ScuDo
Scuola di Dottorato ∽ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (32ndcycle)

Knowledge Graph Embeddings for Recommender Systems

By

Enrico Palumbo

******

Supervisors:
Prof. Elena Baralis, Politecnico di Torino
Dr. Giuseppe Rizzo, LINKS Foundation
Prof. Raphaël Troncy, EURECOM

Doctoral Examination Committee:
Prof. Paolo Cremonesi, Politecnico di Milano (referee)
Dr. Cataldo Musto, Università degli Studi di Bari "Aldo Moro" (referee)
Prof. Alejandro Bellogin, Universidad Autónoma de Madrid
Prof. Silvia Chiusano, Politecnico di Torino
Prof. Paolo Garza, Politecnico di Torino

Politecnico di Torino
2020

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Enrico Palumbo
2020

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

Recommender systems are ubiquitous on the Web, improving user satisfaction and experience by providing personalized suggestions of items they might like. In the past years, knowledge-aware recommender systems have shown to generate high-quality recommendations, combining the best of content-based and collaborative filtering. The crucial point to leverage knowledge graphs to generate item recommendations is to be able to define effective features for the recommendation problem. Knowledge graph embeddings learn a mapping from the knowledge graph to a feature space solving an optimization problem, minimizing the time-consuming endeavor of feature engineering and leading to higher quality features. Thus, the main pillar of this thesis investigates the use of knowledge graph embeddings for recommender systems. In this thesis, we introduce entity2rec, which learns user-item relatedness for item recommendation through property-specific knowledge graph embeddings. entity2rec has been benchmarked with a set of existing knowledge graph embeddings algorithms (translational models, node2vec) that we have applied to the recommendation problem and with popular collaborative filtering algorithms on three standard datasets. entity2rec has shown to generate accurate and non-obvious recommendations, achieving high accuracy, serendipity, and novelty, and to be particularly effective when the dataset is sparse and has a low popularity bias. Furthermore, entity2rec is based on a recommendation model that encodes the semantics of the knowledge graph and can thus be interpreted and configured for a particular recommendation problem. entity2rec has also been tested in a cold start scenario with real new users through a web application called TinderBook. TinderBook is a web application that recommends books to users, given a single book that they like, leveraging an item-item relatedness measure based on entity2rec.

In addition to defining effective features, a crucial element for the quality of knowledge-aware recommender systems is the quality of the knowledge graph itself. Typically, when building a knowledge graph from a set of heterogeneous data sources,

duplicates are a major source of noise in the data. Thus, the second part of the thesis deals with the entity matching problem in the process of knowledge graph generation. In this thesis, we introduce “STEM: Stacked Threshold-based Entity Matching”. STEM is a machine learning layer that can be ‘stacked’ upon existing threshold-based classifiers to improve their precision and recall for the entity matching task. STEM has been tested on three datasets from different domains (finance, music) using two different threshold-based classifiers (linear and Naive Bayes), significantly improving the quality of the entity matching. STEM has also been applied in the context of the European research project 3cixty in the creation of a tourist knowledge graph encompassing places and events of a city, enhancing the deduplication process, and consequently, the quality of the knowledge graph.

Finally, this thesis deals with the extension of the recommendation problem to temporal sequences, i.e. with Sequence-Aware Recommender Systems (SARS). Specific attention is devoted to the problem of learning to recommend tourist paths, sequences of tourist activities that can be of interest for a user. We propose the Path Recommender, an approach based on a Recurrent Neural Network (RNN) trained on sequences of user check-ins collected from Foursquare. The Path Recommender shows to outperform a set of competing sequence-aware algorithms (bigram, Conditional Random Fields) on a set of relevant metrics. An extended and ad-hoc version of the Path Recommender architecture is devised for the problem of automated playlist continuation and tested in the context of the RecSys2018 challenge, achieving the 14th position out of 33 participants in the creative track and the 36th position out of 113 participants in the main track. The Foursquare dataset that has been collected and the evaluation framework for SARS that has been defined in this work have become public resources available to the research community.

Acknowledgements

My PhD was a unique opportunity and a great experience and I would like to thank the people who made it possible.

First, I would like to thank my supervisor Dr. Giuseppe Rizzo from LINKS Foundation for introducing me to the research field, for coming up with the idea of this PhD path as an international collaboration among three excellent research institutions (LINKS Foundation, EURECOM, Politecnico di Torino), and for constantly guiding me through these years. Then, I would like to thank my supervisor Prof. Raphaël Troncy for giving me the possibility of spending a great year in EURECOM and for thoroughly following and reviewing my work. Finally, I would like to thank my supervisor Prof. Elena Baralis from the Politecnico di Torino for the insightful discussions and comments and for managing the PhD process. I also would like to thank Dr. Enrico Ferro and Dr. Michele Osella, who guide the research area where I worked in LINKS Foundation, for offering advice and a business-oriented perspective into the impacts of my work.

I would like to thank the referees and the committee members for taking the time to review my manuscript and for sharing their precious comments to improve my work.

Finally, none of this would have been possible without the support and love of my family: my father Angelo Palumbo, my mother Elisabetta Bubba, my brother Pierandrea Palumbo, my girlfriend Roberta Cappa.

Contents

List of Figures

List of Tables

List of Abbreviations

CBF: Content-Based Filtering

CF: Collaborative Filtering

CRF: Conditional Random Field

KG: Knowledge Graph

KNN: K-Nearest Neighbors

FM: Factorization Machine

LOD: Linked Open Data

LBSN: Location-Based Social Network

POI: Point Of Interest

RNN: Recurrent Neural Network

RS: Recommender System

SARS: Sequence-Aware Recommender System

STD: Semantic Trails Dataset

STEM: Stacked Threshold-based Entity Matching

Chapter 1

Introduction

1.1 Less is more: personalization in the digital age

“Less is more” is the motto that the architect Ludwig Mies van der Rohe invented to describe the minimalist aesthetic in architecture. In the digital age, though, abundance is king: 300 hours of video are uploaded to YouTube every minute[1],1,569 TV shows and 4,010 movies are available on Netflix[2], 20 millions of songs are on Spotify[3] and 564 million of items are sold on Amazon only in the US[4]. Indeed, it is estimated that 90% of all the world’s data has been created in the past two years[5]. This abundance of options to choose from within just one-click should make us happier and more satisfied with our decisions, allowing us to find what is just right for our needs. In fact, this is seldom the case. Psychological studies show that humans are quite bad at choosing among many different options, easily feeling overwhelmed, choosing “none of the above” or opting for poor compromises [4]. The studies point out that choosing between a large number of possibilities quickly becomes a burden as a result of complex interaction among psychological processes that permeates our culture. We are surrounded by high expectations and get easily disappointed by choosing any option that is not the very best, we feel the burden of

[1] https://bit.ly/36a6Bym

[2] https://bit.ly/2RpGAHg

[3] https://bit.ly/36gKc2z

[4] https://bit.ly/2rfxlPk

[5] https://bit.ly/2Yr6YSm

opportunity cost (‘if I had bought X rather than Y, would have that been better?’), we often regret what we chose (‘I should have never bought this!’), we are averted to trade-offs (‘I want a fast, beautiful and new car that is also very cheap’) and so on. On top of this, we have extremely little patience in making choices. Consumer research suggests that a typical Netflix user loses interest after a time interval of 60 to 90 seconds of choosing, skimming through 10 to 20 titles on one or two screens [5]. Providing relevant content to users in a short time is crucial to keep users engaged.

Recommender Systems (RSs) are defined as software tools and techniques providing suggestions for items to be of use to a user [6]. A RS can be seen as a personalized filter that help users in reducing the number of available options, pre-filtering a subset of items that are deemed relevant, and thus reducing the burden of the decision-making process. Nowadays, all of major web companies make use of recommender systems to engage their customers and provide them with tailored content (e.g. Netflix [5], Amazon [7], Youtube [8]). The business value of recommender systems for these platform-based companies is enormous. For instance, Netflix has estimated that its recommender system, by avoiding customers’ unsubscriptions and maximizing the effective catalog size, allows them to save more than 1B$ per year [5].

Artificial Intelligence is the buzzword of the moment and it is considered by many experts as the technological revolution of the century. Looking at its history, the quest for AI dates back to the origin of computer science, and many attribute its birth as a discipline to the Darthmouth workshop in 1956 [9]:

“We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved

for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer."

Ever since, AI has encountered phases of strong optimism (*within a generation ... the problem of creating artificial intelligence will substantially be solved*, Marvin Minsky, 1967 [10]) and phases of pessimism (so-called 'AI winters'), during which skepticism grew stronger as the problem turned out to be much more complex than it first appeared. In recent years, AI has fed on the explosion of data available on the Web. Machine Learning (ML), and more specifically Deep Learning and Neural Networks, the fields that studies algorithms that are able to learn from data generalized rules in a bottom-up fashion, and Semantics, the field that studies how to better structure and organize data in a semantic way in a top-down fashion, have led the development of AI. Both ML and Semantics are applied nowadays in designing Recommender Systems that support our choices. ML is applied to extract from the user's history general patterns, which are then used to provide personalized suggestions. If an Amazon user has bought a Los Angeles Lakers jersey and a Space Jam DVD, chances are that we are talking about a basketball fan. If a Netflix user has watched Manhattan, Match Point and Vicky Cristina Barcelona, it is likely to be a Woody Allen lover. If a Spotify user has listened to Metallica, Iron Maiden and Led Zeppelin, he/she will probably listen to hard rock and metal music.
On the other hand, Semantics has allowed to create web-scale knowledge bases, often called Knowledge Graphs (KG), which contain a huge amount of information related to the items of the recommendations [11]. Who are the starring actors of Pulp Fiction? What is the abstract of the book 1984? Who wrote the song Wish You Were Here? What do Fargo and The Big Lebowski have in common? All of these questions can have an answer thanks to the data openly and publicly available on the Web in the Linked Open Data cloud [12] and can be used to generate more accurate recommendations, to enrich items descriptions and to gain transparency into recommendations (Sec. 2.2.2).

This thesis is located at the intersection between the Recommender Systems and the Semantics research fields, showing how machine learning algorithms can be used to learn features from knowledge graphs to make recommendations. This approach allows to leverage the advancements in Machine Learning in learning features for prediction problems and the wealth of data available on the Web in the structured form of Knowledge Graphs. In the next section, we outline the research challenges and the contributions of the thesis to this overarching research goal.

1.2 Research challenges and contributions

RSs are typically divided into two families: content-based recommender systems and collaborative filtering recommender systems [13] (see also Sec. 2). RSs collect users' preferences, which are either explicitly expressed, e.g., as ratings for products, or are inferred by interpreting user actions, e.g. views and clicks on a web page. Content-based recommender systems recommend items that are similar to the ones that the user has liked in the past. Content-based recommender systems require the definition of a feature vector describing the item content, and suffer the overspecialization problem, i.e. they tend to recommend similar items over and over again. On the contrary, collaborative filtering recommender systems leverage the preferences of other users to identify suitable items for the target user. In its simplest formulation, collaborative filtering recommends to the target user items that users with similar preferences have liked in the past [14]. A major appeal of collaborative filtering is that it does not require an item model and can perform recommendations across different item types (e.g. can recommend books given preferences about shoes). Furthermore, by relying on the preferences of other 'similar' users, it does not suffer from the overspecialization problem. For these reasons, collaborative filtering is probably the most popular and widespread approach to create recommender systems. However, collaborative filtering suffers from the data sparsity problem [13], i.e. it does not generate accurate recommendations when few data about users' preferences is available in the system. Another important limitation of collaborative filtering is that it cannot recommend 'new items', i.e. items that have never been consumed. Collaborative filtering algorithms also have a limited capability of generating explanations for the recommendations, as they cannot leverage the item content for this purpose [15]. Hybrid recommender systems combine content-based and collaborative filtering in the attempt to address the limitations of both approaches and received a lot of attention in the past years [16]. Knowledge graphs are an ideal data structure for hybrid recommender systems. Thanks to their flexibility, knowledge graphs can easily model heterogeneous entities, such as users, items, and other entities, and different types of interactions between them, such as the preference of a user for an item or the relation of an item with a particular entity. In the past years, several studies have made of use of knowledge graphs for hybrid recommender systems, and a few of them have started to use knowledge graph embeddings for this task (see Paragraph 2.2.2). We believe that the use of knowledge graph embeddings is quite promising and interesting for recommender systems, as it allows to leverage the effectiveness of state-of-the-art feature learning algorithms from machine learning research to obtain feature vectors from knowledge graphs to make recommendations. In this context, we formulate the first research challenge of this work (Chapter 3):

RQ1 *How can knowledge graph embeddings be used to create accurate, non-obvious and semantics-aware recommendations based on both collaborative and content-based filtering?*

To address this question, we devise a knowledge graph model including users, items (entities that are the object of recommendations) and other entities (e.g. starring actors of a film, author of a book). The knowledge graph includes a set of different relations, such as user-item interactions (explicit or implicit feedback) through the 'feedback' property and item-entity relations through a diversity of properties (e.g. 'starring', 'director', 'author'). Then, a diversity of approaches can be used to generate recommendations using knowledge graph embeddings. Thus, within this major research question, we formulate and address four sub-research questions:

RQ1.1 *how can translational models for knowledge graph embeddings be used to generate recommendations from a knowledge graph?*

We show that translational models (TransE, TransH, TransR) can be applied directly on the knowledge graph model, embedding all entities and relations into vectors. Then, the recommendation problem is addressed as a link prediction problem of the 'feedback' property.

RQ1.2 *how can the graph embedding algorithm node2vec be used to generate recommendations from a knowledge graph?*
We show that node2vec can be applied directly on the knowledge graph model, embedding all entities into vectors. The recommendation problem is then addressed retrieving the closest items to users in the vector space.

RQ1.3 *how can a recommender system use the effectiveness of node2vec in learning features from graphs, while encoding the semantics of multiple properties with the purpose of making recommendations?*
To address this question, we introduce entity2rec, which generates property-specific knowledge graph embeddings for item recommendation. entity2rec generates property-specific embeddings using node2vec on property-specific subgraphs and computes user-item property-specific relatedness scores using a relatedness function in the vector spaces. Then, property-specific relatedness scores are aggregated into a single relatedness score that is used to retrieve relevant items for the user. In this way, entity2rec extends node2vec to multi-relational graphs, allowing to improve the quality of recommendations even in high sparsity and low popularity bias datasets and to encode the semantics of the properties in the recommendation model. The semantic information that is encoded in the recommendation model can be used to better interpret and explain recommendations.

RQ1.4 *how can entity2rec generate recommendations to new users?*
We deploy entity2rec in Tinderbook, a web application that recommends books to users, given a single book that they like. Tinderbook does not require any log-in or previous information about the user, but only a single book they like. Property-specific knowledge graph embeddings built through entity2rec are used to create an item relatedness function, which is leveraged to retrieve the closest items to the one provided by the user in the onboarding phase. Tinderbook shows that entity2rec can be easily extended to generate recommendations to new users.

A critical element for the quality of recommendations generated using knowledge graphs is the quality of the knowledge graph itself. Typically, knowledge graphs are built integrating data from different data sources, which can contain the same entity (e.g. the same event from different event databases). Although at a first glance the problem might appear trivial, identifying and removing duplicates from heterogeneous data sources can be very challenging, as a consequence of the noise in the data, different formats, ambiguous names, several names referring to the same entity and so on. The problem is known as entity matching (Sec. 2.1.2). Entity matching (also known as instance matching, data reconciliation or record linkage) is the process of finding non-identical records that refer to the same real-world entity among a collection of data sources [17]. Entity matching calls for algorithms that are able to automatically assess, with a certain degree of confidence, whether two records refer to the same real world entity. A common way to approach the problem is the use of threshold-based classifiers [18], such as Silk [19]. The basic idea of threshold-based classifiers is that of computing a set of property-specific similarities between pairs of records (e.g. string similarity between names, geographic distance between locations), deriving a global score that indicates the degree of confidence of the classifier in saying that the records are indeed the same entity. The confidence score is compared with a decision threshold in order to obtain the final decision. The higher the threshold, the more selective the classifier will be. Thus, the threshold introduces a trade-off between the precision and recall of the algorithm and is typically manually tuned to maximize the F-score of the algorithm. After a set of preliminary experiments conducted in the context of the FEIII challenge [20], we observed that combining a set of decision thresholds using simple ensemble techniques such as majority or union voting breaks the trade-off between precision and recall, slightly improving both at the same time. In light of this finding, we have moved forward to investigate whether more sophisticated ensemble techniques based on machine learning, could do even better. Stacked Generalization (os simply stacking) [21], i.e. the use of the predictions of an ensemble of classifiers as a feature vector for an additional supervised classifier, seemed like an excellent candidate for this purpose. Thus, we have formulated the following research challenge (Chapter 4):

RQ2 *Can ensemble learning algorithms such as stacked generalization improve the performance of threshold-based classifiers in the entity matching process?*

To address this question, we introduce STEM (Stacked Threshold-based Entity

Matching). STEM is a machine learning layer that can be stacked upon any threshold-based classifier to enhance the quality of the entity matching. More specifically, STEM combines the predictions of different thresholds through a supervised classifier, breaking the trade-off between precision and recall, and improving the entity matching in terms of F-score. Within RQ2, we formulate three sub-research questions:

RQ2.1 *Does STEM improve the F-score of threshold-based classifiers in a significant and consistent way?*
We present experimental tests on three datasets and using two different threshold-based classifiers, showing that, in all cases, STEM improves the performance of the threshold-based classifier, up to 43% of F-score.

RQ2.2 *How does STEM perform when little training data is available?*
We assess the F-score of STEM varying the amount of available training data, comparing it with that of a set of supervised classifiers directly trained on property-specific similarity values computed between pairs of records. The experiments show that STEM consistently performs better with respect to competitors and is less sensitive to the amount of training data.

RQ2.3 *How can STEM be applied in the process of building a knowledge graph containing Points of Interests (POI) and events for tourists?*
We report on the application of STEM in the context of the 3cixty European research project. 3cixty had the goal of creating an application to guide tourists in the exploration of a city by means of a comprehensive knowledge graph containing places and events of a city. STEM has been applied in the process of generation of the 3cixty knowledge graph, improving the entity matching process and thus the final quality of the knowledge graph.

So far, we have always referred to recommender systems that recommend a single item. We have also assumed that the history of user preferences is the source of information to learn from. However, not always long-term information about the user is available, and, at the same time, short-term information is extremely relevant in certain applications. A clear example is the tourist domain. We have all probably experienced the feeling of being lost in a city as tourists and the need of organizing a tour that maximizes our satisfaction in the city exploration. In the past, the creation of tourist paths has often been addressed as a specific application of the *orienteering problem* [22], i.e. a problem of optimization with constraints whose goal is to find a path that visits a number of points optimizing a global score. Recently, some studies have turned the problem upside down, opting for an inductive approach where 'optimal' tourist paths are learned from real user's data and recommended to the user. Location-based Social Networks (LBSN) represent a great source of data about users' movement in a city, which can be leveraged to make personalized recommendations. LBSN allow users to *check-in* in a Point-of-Interest (POI)[6] and share their activities with friends, providing publicly available data about their behavior. However, a RS that suggests activities to do in a city ought to take into account where the user was to effectively recommend where to go next. Suppose the user is in an *Irish Pub* at 8 PM: is the user more likely to continue her evening in a *Karaoke Bar* or in an *Opera House*? Better a *Chinese Restaurant* or an *Italian Restaurant* for dinner after a *City Park* in the morning and a *History Museum* in the afternoon? We define such a sequence of activities as a 'path' or a 'trail'.

The problem of recommending tourist paths calls for techniques that are specifically meant to deal with sequential data. These systems are known as Sequence-Aware Recommender Systems (SARS) and they are receiving a lot of attention in domains such as music, tourism, e-commerce. SARS are based on a variety of machine learning approaches, but only few studies have used Deep Learning (DL) models (Sec. 2.2.3), in spite of their effectiveness in modelling sequential data in other domains such as Natural Language Processing [23]. Furthermore, many works conducted in this field lack a common definition of the problem that they are trying to address, standard evaluation protocols and datasets specifically conceived for sequences [24]. In this thesis, we aim to contribute to fill these gaps by addressing the following research challenge (Chapter 5):

[6] The term venue is used interchangeably with POI in this work to describe an entity that has a somewhat fixed and physical extension as defined by http://schema.org/Place

RQ3 *How can we create a recommender system that learns to recommend tourist paths from LSBN data, effectively leveraging the temporal correlation among tourist activities?*

To answer this research question, we introduce the Path Recommender, a RNN that learns to model and generate sequences of tourist activities from a dataset of Foursquare check-ins. In this context, we introduce three sub-research questions:

RQ3.1 *How can we benchmark different SARS, improving the comparability and reproducibility of experiments?*

We introduce an evaluation framework called Sequeval, which defines a set of metrics and evaluation protocols for sequence-aware recommender systems, and a new dataset of check-ins that we have collected, processed and publicly released.

RQ3.2 *How do deep learning methods perform compared to other more traditional modelling approaches in the generation of tourist paths?*

We compare the RNN approach of the Path Recommender to a set of sequence-aware algorithms, showing that it is able to generate accurate and non-obvious recommendations.

RQ3.3 *How can we extend the Path Recommender to deal with the automated music playlist continuation task?*

We describe the extension of the Path Recommender elaborated for the task of automated music playlist continuation in the context of the RecSys2018 challenge. The extended model contains a number of improvements specifically meant to deal with the task, such as the capability of handling song titles and lyrics.

In the next section, we summarize the thesis structure.

1.3 Thesis structure

As illustrated in Fig. 1.1, the thesis is divided in six chapters, addressing research challenges within the Recommender Systems and Semantics research fields.

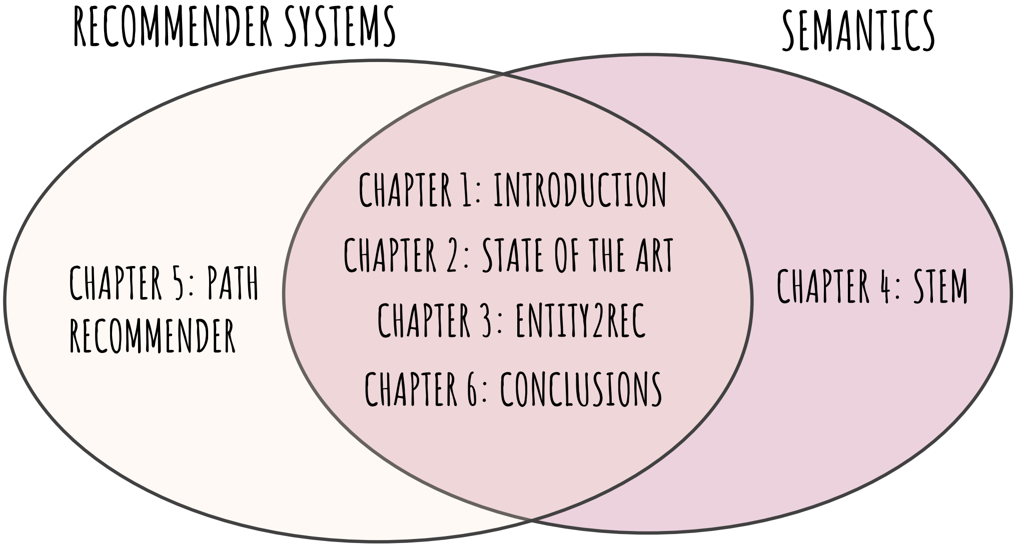


Fig. 1.1 The thesis is divided in six chapters, addressing research challenges in the fields of Recommender Systems and Semantics.

Chapter 1 provides the general context in which the thesis is grounded, describes the research challenges and contributions of the thesis and provides an outline of the work.

In Chapter 2, the state-of-the-art is described, going from general notions about RS and Semantics to the most recent and advanced works in the field.

Chapter 3 contains the theoretical and experimental work concerning the use of translational models [25, 26], node2vec [27] and entity2rec [3] to create knowledge graph embeddings for recommender systems.

Chapter 4 describes the STEM (Stacked Threshold-based Entity Matching) approach [28], its experimental validation and the use-case of the 3cixty research project [29].

Chapter 5 describes the Path Recommender [30], the evaluation framework Sequeval and the collection of the check-in dataset [31, 32], and the extension of the Path Recommender to the music domain [33].

In Chapter 6, we summarize the findings and highlights, outline the future work, and draw the main conclusions of the thesis.

Chapter 2

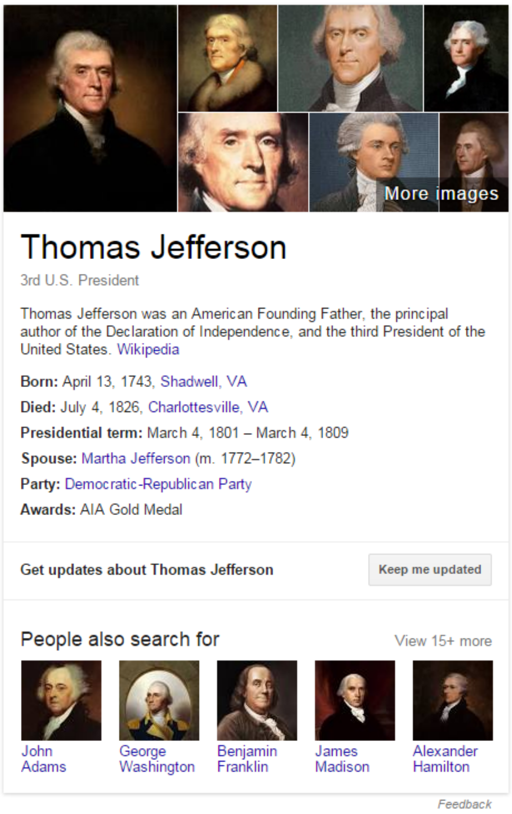
State of the art

This chapter aims to introduce a set of notions that are important for the understanding of the context in which the thesis is located. We provide basic notions on well-established concepts such as Knowledge Graphs and Recommender Systems, as well as summaries of the most recent work in the scientific literature related to what is presented in this thesis.

2.1 Knowledge Graphs

2.1.1 Background

Knowledge Graphs (KG) are graph-structured knowledge bases (KB) that store factual information in the form of relationships between entities [34]. Graph-structured representations have a long history in the fields of logic and artificial intelligence, often under the name of ‘semantic networks’ [35]. Knowledge graphs enable the modeling of real world entities and their relations, powering search engines [36], as well as natural language understanding systems [37]. The term ‘Knowledge Graph’ has been popularized by Google in 2012 [36] to describe their graph-structured knowledge base containing hundreds of millions of entities and relations. The Google Knowledge Graph has a major role in the company’s search engine, providing atomic answers to queries and showing a side panel with specific information



Thomas Jefferson

3rd U.S. President

Thomas Jefferson was an American Founding Father, the principal author of the Declaration of Independence, and the third President of the United States. Wikipedia

Born: April 13, 1743, Shadwell, VA

Died: July 4, 1826, Charlottesville, VA

Presidential term: March 4, 1801 – March 4, 1809

Spouse: Martha Jefferson (m. 1772–1782)

Party: Democratic-Republican Party

Awards: AIA Gold Medal

Get updates about Thomas Jefferson

Fig. 2.1 Google Knowledge Graph information for the entity Thomas Jefferson. The panel shows a short textual description of the entity, a link to Wikipedia and a set of important properties that describe him. Then, related entities are shown. By Google - Google web search, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=37997885

concerning the entity mentioned in the user query. See in Fig. 2.1 an example of the side panel from the Google Knowledge Graph for the entity ‘Thomas Jefferson’. The panel shows a short textual description of the entity, a link to Wikipedia and a set of important properties that describe him. Then, related entities in the KG in terms of what the users typically also search for are shown.

Most of the major tech companies have built proprietary knowledge graphs to power their search functions, e.g. Microsoft Satori[1] or Amazon’s Product Graph[2]. KGs constitute a fundamental ingredient for Question Answering systems and for today’s voice assistants (Amazon’s Alexa, Microsoft’s Cortana, Google’s Assistant, Apple’s

[1] https://bit.ly/2DU3rCX

[2] http://lunadong.com/talks/PG.pdf

Siri)[3].

Knowledge graphs are also the backbone of the Linked Open Data cloud [12] and the graph-based representation is the core of the RDF standard [38] and the Semantic Web [39]. In the past years, several publicly available KGs have been created, such as DBpedia [11], Wikidata [40] or YAGO [41]. DBpedia and Wikidata are probably the most well known public knowledge graphs, and they are both derived from Wikipedia. DBpedia is built by extracting information from Wikipedia infoboxes and currently contains information about 38.3 million things[4]. Wikidata is a more recent project, started in 2012 by the Wikimedia foundation, collaboratively edited and created by the community. Although it is younger than DBpedia and not as mature from some viewpoints such as the programmatic access to data, it is gaining momentum, and currently contains information about 69.7 million things[5]. In this thesis, we often rely on DBpedia to create a KG for recommender systems (Chap. 3). The major convenience of using DBpedia is that it has often been used to create semantics-aware RS and that public mappings to standard RS datasets have been publicly released [42].

Knowledge graphs are graphs in the sense that they store facts under the form of links between entities. For example, consider the fact that Obama was born in Hawaii. Both ‘Obama’ and ‘Hawaii’ are represented as nodes of the graph, whereas the property ‘birth place’ is represented by a typed edge connecting the two nodes. A fact is thus represented by a triple: *(subject, predicate, object)*, e.g. *(Obama, birthPlace, Hawaii)*. Both entities, such as Obama and Hawaii, and predicates such as ‘birthPlace’ are identified by unique identifiers, and classified in an ontology, which pre-defines all possible entity and relation types. Consider the example of the DBpedia Ontology: each entity is a ‘resource’, marked by the prefix ‘dbr’ standing for DBpedia resource, and relations are defined by the prefix ‘dbo’ standing for DBpedia ontology. Thus, a valid triple of DBpedia is: *(dbr:Obama, dbo:birthPlace, dbr:Hawaii)* (Fig. 2.2). A more formal definition of knowledge graph and ontology is provided in Sec. 3.1.

[3] https://bit.ly/2RrE1nU

[4] https://wiki.dbpedia.org/about

[5] https://www.wikidata.org/wiki/Wikidata:Statistics

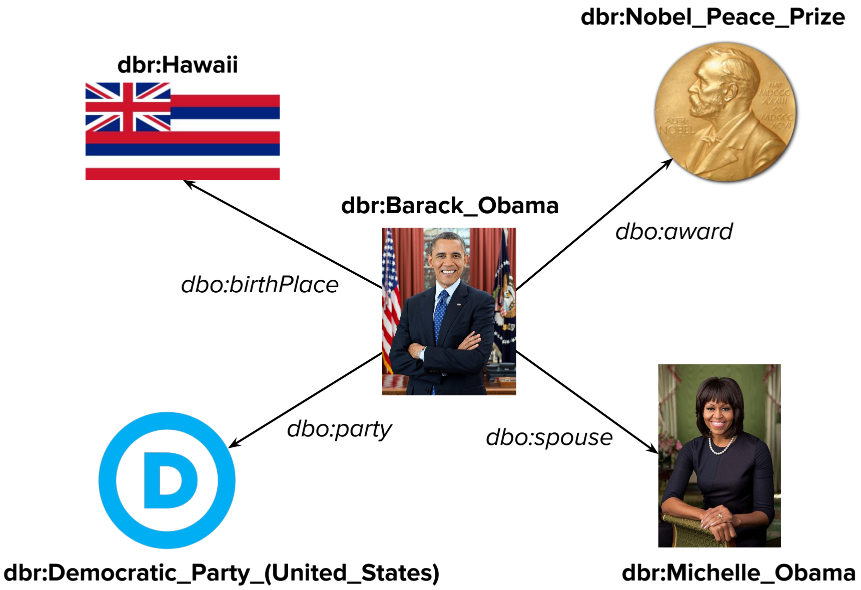


Fig. 2.2 An excerpt from DBpedia representing the entity Barack Obama. Properties are represented as typed edges connecting the entity to other entities.

Although the interpretation of existing triples in a KG is straight-forward, that of non-existing triples is less so. Two alternative assumptions are normally proposed:

1. **Closed World Assumption (CWA):** non-existing triples indicate false relationships. For instance, if there is no triple indicating that Obama is married, it means that he is not married.

2. **Open World Assumption (OWA):** non-existing triples do not mean that the relationship is false or true, but simply unknown. In the former example, the information about Obama's marriage might simply be missing from the KG.

In some cases, a Local Closed World Assumption (LCWA) [43] is formulated, which assumes that a KG is only locally complete. Given an observed triple (s,p,o), it assumes that any other triple $(s,p,\hat{o})$ with $\hat{o} \neq o$ is false. Note that this is strictly true for 1-to-1 relations such as $birthPlace$, but not necessarily for 1-to-N relations such as $literaryGenre$. Given that large-scale web KGs are incomplete, Semantic Web and RDF follow the OWA, which we also follow in this thesis, unless specified otherwise.

The completeness and the quality of a KG are crucial points, which strongly depend on the approaches used to construct them. KGs can be created through totally *curated approaches*, where experts manually input all the necessary information, leading to very high quality graphs. This process is however not scalable and very time-consuming. *Collaborative approaches* for KG creation, such as the one used in Wikidata, attempt to improve the scalability of the process by distributing the effort on the members of the community. This approach preserves the quality of the graph, but depends on the good will of a number of individuals who need to be coordinated and motivated in the endeavor. *Automated structured approaches* attempt to reduce the human effort in the process, by extracting information from semi-structured sources (e.g. Wikipedia infoboxes) using ad-hoc rules, and often results in high-quality KGs such as DBpedia. Also structured sources, such as existing open databases, are often used and integrated in the construction of a KG. As we will see in the next section, this leads to a complicated issue that is known as *entity matching*. However, most of the information on the Web is unstructured. For this reason, recently, some works have developed *automated unstructured approaches* that attempt to extract structured information from text or other media of web pages using machine learning [43]. Although scalable and with a high coverage, these methods are still far from being as accurate as the other approaches.
In the following section, we introduce a problem that has been dealt with in this thesis and that is crucial for the construction of knowledge graphs from structured sources: the problem of entity matching.

2.1.2 Entity Matching

Knowledge graphs are often built integrating a set of heterogeneous structured data sources, aiming to increase their coverage and comprehensiveness. Typically, this leads to the fact that different records from different data sources (or even from the same data source) refer to the same real world entity. Suppose that you need to create a knowledge graph containing all the restaurants of a city, and, in order to do so, you decide to integrate a data source X and a data source Y. Now, suppose that data source X contains a record for a restaurant named ‘Da Luigi’, located in

'Via Napoleone, 32' and data source Y contains a record for a restaurant named 'Da Luigi' located in 'Via Monti, 10'. Will they be two different restaurants? Or is the address of one of the two restaurants wrong? The problem of deciding whether two records refer to the same real world entity is called 'entity matching'[6]. Entity matching is a crucial task for data integration [44] and probabilistic approaches able to handle uncertainty have been proposed since the 60s [45].

A survey of frameworks for entity matching is reported by Köpcke in [18], where a classification of several entity matching frameworks is done by analyzing the entity type, i.e. how the entity data is structured, blocking methods, i.e. the strategy employed to reduce the search space and avoiding the comparison of each possible pair of records, the matching method, i.e. the function utilized to determine if a pair of records represents the same real world entity and the training selection, i.e. if and how training data is used. By taking into account the matching method, entity matching frameworks may be divided in frameworks without training [46–48], in which the model needs to be manually configured, training-based frameworks [49, 50], in which several parameters are self-configured through a learning process on an annotated training set, and hybrid frameworks, which allow both manual and automatic configuration [51, 52].

The authors of the survey thoroughly compare different frameworks on a set of key performance indicators and highlight a research trend towards training-based and hybrid approach, which, in spite of the dependence on the availability, size and quality of training data, significantly reduce the effort of manual configuration of the system. The most commonly used supervised learners are Decision Trees and SVM. Training can help for different purposes, such as learning matching rules or in which order matchers should be applied, automatically setting critical parameters and/or determining weights to combine matchers similarity values [53, 54, 50, 51]. In [55], a comparison among the most common supervised (training-based) learning models is reported together with an experimental evaluation. The authors report a high degree of complementarity among different models which suggests that a combination of different models through ensemble learning approaches might be an effective strategy. The idea of ensemble learning is to build a prediction model by

[6] Also known as 'Entity resolution' when it includes the final step of merging the two entities into a new one to be input in the KG

combining the strengths of a collection of simpler base models. Ensemble learning can be broken down into two tasks: developing a population of base learners from the training data, and then combining them to form the composite predictor [56]. In [53] the authors report that an ensemble of base classifiers built through techniques such as bagging, boosting or stacked generalization (also known as stacking) generally improves the efficiency of entity matching systems. Another evidence of the efficiency of ensemble approaches to entity matching is reported in [57]. Ensemble learning has also been successfully applied to other fields relevant to the Semantic Web community, such as Named Entity Recognition [58, 59]. In Chapter 4, we show that stacking is also effective when applied to a single threshold-based classifier using the predictions of several decision thresholds as features. This allows stacking to be an incremental improvement, a generic layer that can be used on top of any existing threshold-based classifier already in use by researchers and practitioners.

In the past years, the Linked Data [12] research community has shown a great deal of interest for Entity Matching. More specifically, Entity Matching (or Instance Matching) can be seen as a part of the process of Link Discovery. Link Discovery has the purpose of interlinking RDF data sets that are published on the Web, following the evidence of recent studies that show that 44% of the Linked Data datasets are not connected to other datasets at all [60]. Link Discovery can be seen as a generalization of Entity Matching, because it can be used to discover other properties than an equivalence relation between instances. Furthermore, as remarked in [61], in Link Discovery resources usually abide by an ontology, which describes the properties that resources of a certain type can have as well as the relations between the classes that the resources instantiate. The authors of [61] report a comprehensive survey of Link Discovery frameworks, which shows that modern framework such as Silk [62, 63], LIMES [64], EAGLES [65] combine manually defined match rules with genetic programming and/or active learning approaches to automatize the configuration process. A different approach is proposed by WOMBAT [66], which relies on an iterative search process based on an upward refinement operator. WOMBAT learns to combine atomic link specifications into complex link specifications to optimize the F-score using only positive examples. Another line of work is that of collective

entity matching (or resolution) systems, which are not based on pairwise similarity comparison, but rather on the attempt to capture the dependencies among different matching decisions [67–72].
Some approaches have modeled the problem of entity matching as a link prediction problem approached building knowledge graph embeddings, where the property to predict is 'equal to' [73]. In [74], knowledge graph embeddings are used to obtain a distance measure between entities that is used to determine whether they represent the same entity or not. The next section specifically deals with knowledge graph embeddings.

2.1.3 Knowledge Graph Embeddings

'Embedding' is a buzzword nowadays. In recent years, there has been an explosion of works concerning the topic, especially for what concerns word embeddings. Word embeddings are vector representations of a word, typically in a Euclidean space, which preserve the semantics of the word. Word embeddings have become extremely popular after the release of the Word2Vec model [75]. Word2vec efficiently learns word embeddings by training a shallow neural network to predict the context of a word, defined by a sliding window of amplitude c. Thus, given a word w_t, the context is defined by the surrounding words $w_{t-c}, w_{t-c+1}, ..., w_{t+c-1}, w_{t+c}$. Two different architectures are proposed, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model.

Continuous Bag-of-Words (CBOW) Model: the CBOW model implements a neural network where the input corresponds to the context words $w_{t-c}, w_{t-c+1} ... w_{t+c-1}, w_{t+c}$ and the output to predict is the target word w_t (Fig. 2.3). Given T training words $w_1 ... w_T$, the learning process is defined as an optimization problem of the average log-probability of observing the target word w_t given context words $w_{t-c}, w_{t-c+1} ... w_{t+c-1}, w_{t+c}$ with respect to the parameters of the neural network model W:

$$W = arg\ max_W \frac{1}{T} \sum_{t=1}^{T} \log p_W(w_t | w_{t-c} ... w_{t+c}) \tag{2.1}$$

where $p_W(w_t|w_{t-c}...w_{t+c})$ is modeled by the softmax function:

$$p_W(w_t|w_{t-c}...w_{t+c}) = \frac{exp(\hat{v}v'_{w_t})}{\sum_{w=1}^{V} exp(\hat{v}v'_w)} \tag{2.2}$$

where v'_{w_t} is the output vector of the target word w_t, V is the whole vocabulary of words, and $\hat{v}$ is the average of the context vectors $v_{w_{t-c}}..v_{w_{t+c}}$.

Skip-Gram Model: the Skip-Gram model implements a two layer neural network where the input corresponds to the target word w_t and the output to the context words $w_{t-c}, w_{t-c+1}...w_{t+c-1}, w_{t+c}$ (Fig. 2.3). Given T training words $w_1...w_T$, the optimization problem is defined as:

$$W = arg\ max_W \frac{1}{T} \sum_{t=1}^{T} \sum_{j=-c,\neq 0}^{c} \log p_W(w_{t+j}|w_t) \tag{2.3}$$

where the probability $p_W(w_{t+j}|w_t)$ is modeled by the softmax function:

$$p_W(w_O|w_I) = \frac{exp(v'_{w_O} v_{w_I})}{\sum_{w=1}^{V} exp(v'_w v_{w_I})} \tag{2.4}$$

where v'_w and v_w are the input and output representations of the word w and V is the complete vocabulary of words. Given the computational cost of summing over all the words in the vocabulary, in both CBOW and Skip-gram models, the denominator of Eq.2.2 and Eq.2.4 is normally approximated using either hierarchical softmax or negative sampling [75]. In recent years, more models of word embeddings have been proposed, the most famous being FastText [76], which pushes the Word2Vec model to account for sub-word information.

DeepWalk [77] has shown that language models such as Word2Vec [75] can be extended to graph structures. We have seen that Word2Vec works by training a neural network to predict neighboring words in a text. But what is the neighborhood of a node in a graph? The key idea of DeepWalk is that of defining the neighborhood of a node as the nodes that are sampled in a random walk process. In this way, DeepWalk

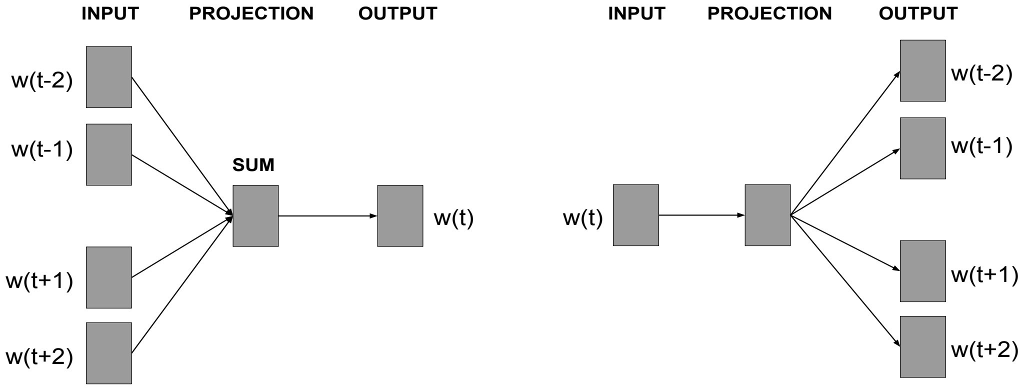


Fig. 2.3 Comparing the CBOW (left) and the Skip-Gram (right) architecture [1]

.

is converting the graph into a ‘text’ by performing a set of random walks on the graph, making the sequences of nodes sampled during walks become ‘sentences’ of the text. Then, neural language models originally devised for text, such as Word2Vec, can be applied on the unrolled graph, learning node embeddings as if they were ‘words’ of a document. In DeepWalk, a random uniform random walk is used, but different graphs have different connectivity patterns that should be taken into account when building node representations.

node2vec [78] has built upon this insight, introducing a more sophisticated random walk strategy that can be more easily adapted to a diversity of graph connectivity patterns, outperforming DeepWalk. node2vec has two additional hyper-parameters (p,q) with respect to DeepWalk, which can be set to fine-tune the 2nd order random walk exploration strategy to the structural properties of the graph. p is called the *return parameter* and governs the likelihood of returning to the previous node, q is called the *in-out parameter* and governs the likelihood of moving one step away from the previous node (Fig. 2.4). These two hyper-parameters allow to create random walks that mix Breadth-First-Search and Depth-First-Search. Then, similarly to DeepWalk, Word2Vec can be applied on the unrolled graph, generating node embeddings. Node embeddings generated through DeepWalk and node2vec can be used for a variety of tasks such as link prediction, multi-label classification, node relatedness. In Chapter 3, we show the application of node2vec on knowledge graphs to generate recommendations.

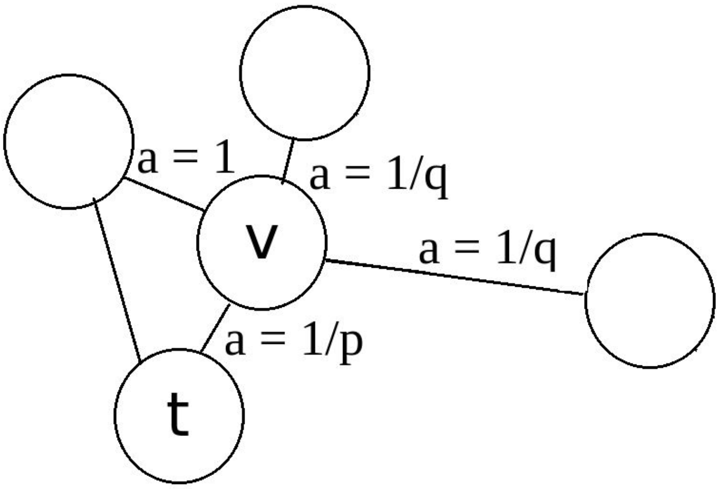


Fig. 2.4 node2vec random walk transition probabilities a, given that the walk is currently in v and it just visited t. If p is low, the walk is likely to go back to the previous node. If q is low, the walk is likely to move one step away from the previous node.

In knowledge graphs, though, not only the graph structure has to be encoded into the features, but also properties and/or entity types. Thus, knowledge graph embeddings are vector representations of entities and/or relations that attempt to preserve the structure and the semantics of the knowledge graph. They are generated using feature learning algorithms that project entities and/or relations into a vector space by solving a learning problem (unsupervised or supervised). Typically, generating KG embeddings involves three steps: (1) representing entities and relations (2) defining a scoring function (3) learning entity and relation representations [79]. According to [79], KG embeddings can be roughly categorized in: *semantic matching models* and *translational distance models*.

Semantic matching models are based on a similarity-based scoring functions that measures the plausibility of a triple by matching the semantics of the latent representations of entities and relations. RESCAL [74] is a semantic matching model, based on a tensor factorization method that explains triples via pairwise interactions of vector representations of entities:

$$f(s,p,o) = \sum_{i=1}^{d} \sum_{j=i}^{d} s_i M_{pij} o_j \tag{2.5}$$

where M_p is the matrix representing the relation p and s_i and o_j are the latent vector representations of entities s and o. Entities are represented by vectors and relations by matrices. Thus, RESCAL requires $O(d^2)$ parameters per relation, where d is the dimension of the embedding vector. Other examples of semantic matching models are: DistMul [80], which simplifies RESCAL by representing relations with diagonal matrices, thus reducing its complexity; ComplEX [81], which extends DistMul using complex numbers in place of real numbers; NTN (Neural Tensor Network) [82], which is an expressive neural network that learns representations using non-linear layers. Recent work has dropped the assumption of embedding in a Euclidean space, showing that using hyperbolic spaces can lead to better performance, especially in modelling hierarchies [83, 84].

On the other hand, translational distance models are based on a scoring function that measures the plausibility of a triple by measuring distances in the vector space, typically after performing a translation operation. Popular examples are TransE [85], TransH [86], and TransR [87]. The basic idea of translational models is that modelling relations in a KG as translations in a vector space. In TransE, for instance, it is assumed that $s+p \approx o$ and the scoring function becomes:

$$f(s,p,o) = D(s+p,o) \tag{2.6}$$

where D is a distance function in the vector space such as the $L1$ or $L2$ norm. In this thesis, we have applied translational models to the recommendation problem and we provide a detailed description of this process in Chapter 3.2.1. We focus on translational models, as they have shown to be intuitive, computationally efficient and accurate at the same time. However, as a future research line, semantic matching models might as well be applied to the recommendation problem.

Another popular knowledge graph embedding approach is RDF2Vec [88]. RDF2Vec is based on a neural language model similar to DeepWalk, and it has been applied to small and large scale RDF graphs with a number of different applications such as link prediction, entity relatedness and entity recommendation. In general, knowledge graph embeddings have been applied to address several tasks such as link prediction, triple classification, entity classification, entity resolution, relation extraction, question answering, and recommender systems [79]. A specific paragraph of the next section is dedicated to describing published work on knowledge graph embeddings for recommender systems (Sec. 2.2.2), which are the central topic of this thesis. Before delving into that, though, we provide a general overview of basic concepts in the Recommender Systems fields as well as on the research trends in the field.

2.1.4 Summary

In this section, we have introduced the concept of Knowledge Graph and discussed the state-of-the-art in two important related research topics: Entity Matching and Knowledge Graph Embeddings. We have seen that Knowledge Graphs are graph-based knowledge bases, which have gained popularity in the past years thanks to their richness and flexibility, both as proprietary assets of tech companies and as Open Data publicly released on the Web through the Linked Open Data movement. We have discussed the troublesome endeavor of building a KG, with a specific focus on the Entity Matching problem when integrating data from heterogeneous sources. We have then presented one of the hottest research topics related to KGs, i.e. the learning of KG vector representations using KG embeddings. After introducing the general concept of embeddings and reviewed the famous Word2Vec model for learning word embeddings, we have described popular approaches that are specifically meant to learn embeddings of (knowledge) graphs.

2.2 Recommender Systems

2.2.1 Background

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [6]. As discussed in Chapter 1, RSs have gained popularity on the Web as effective tools to deal with the problem of information overload and in supporting the choices of users among large sets of available options. RSs are typically based on the history of the user's preferences, which can be expressed in two ways:

Explicit feedback: the user explicitly communicates the value of an item, for instance using a graded scale. A typical form of explicit feedback are *ratings*. Ratings have been used substantially in the past years by the research community, also thanks to the Netflix prize [89] whose task was developing algorithms that are able to predict movie ratings. The task is known as *rating prediction* and its goal is predicting the rating value that a user will assign to a certain item. However, the final objective of the RS is providing a small subset of relevant items for the user, which can be shown in the limited space of a web interface. This problem is known as *top-N item recommendation* (see Sec. 3.1 for a formal definition). In the past, models trained for rating prediction were used for top-N item recommendation, using the predicted rating to rank a set of items that are presented to the user. This approach is far from optimal, as studies have shown that RSs optimized for rating prediction do not necessarily perform well for top-N item recommendation [90]. For this reason, recently, more attention has been placed on addressing the top-N item recommendation problem using models that optimize directly the ranking quality, rather than the error on ratings. Many platforms (e.g. Netflix) have also switched from a five stars rating mechanism to a thumb-up/thumb-down explicit feedback, as binary feedback can be easily used for assessing the quality of ranking models and requires less cognitive effort to the user. In this thesis, we always consider an experimental scenario where we have binary feedback and we directly address the top-N item recommendation problem (Sec. 3). Explicit feedback is precious, as it provides a direct and easy way to understand the user's taste. However, the users typically only rate a small portion of the items that they consume. Thus, explicit feedback is extremely sparse and in many cases it is useful to resort to implicit feedback.

Implicit feedback: the user interacts with an item and assumptions are made on her opinion on the item. Examples are clicks, views, purchases, browsing history, mouse movements. No explicit information about the appreciation of the item is provided, but educated guesses are possible. For instance, if a user frequently listens to an artist, it is reasonable to assume that she likes the artist. Or similarly, if a user has bought an item and does not return it, we can assume that he likes it. All of these assumptions make implicit feedback less reliable than explicit feedback. However, implicit feedback is abundant, and can provide a lot of information on the user's behavior. For this reason, in the past years, many RSs have started to encompass implicit feedback in their models, typically accounting for different confidence levels [91].

As mentioned in Sec. 1, RSs are typically divided in content-based and collaborative filtering systems. We will now give an overview of the most commonly used techniques in content-based and collaborative filtering. In the following, we will generally refer to a utility function r_{ui}, which models the usefulness of item i for user u. In this perspective, the goal of the RS is that of finding the items i that maximize the utility function r_{ui} for u.

Content-based filtering (CBF) is based on the principle of recommending to the user items that are similar to the ones he/she liked in the past. Content-based recommenders require an item model, i.e. a feature vector describing an item. For instance, in the case of a song recommender, each song could be modelled as a vector including the artist, the album, the year of publishing, or the record label. Some approaches then proceed by creating an annotated corpus, labeling as positive examples items that are liked by the user, and train supervised classifiers to identify them [92]. This requires, however, a good number of annotations to provide good performance, as they require to train one classifier per user of the system. Other approaches use a nearest neighbor model, where the utility r_{ui} of user u for item i is given by:

$$r_{ui} = \frac{\sum_{j \in N_u(i)} s_{ij} r_{uj}}{\sum_{j \in N_u(i)} |w_{ij}|} \tag{2.7}$$

where s_{ij} is a measure of content similarity between item i and j and $N_u(i)$ are the nearest items to i among those already consumed by u.

Often, content-based filtering is used for textual documents. Text documents can be represented through keywords or through more sophisticated approaches based on semantics [93, 94]. Recently, different works have used embeddings to learn item representations for content-based recommender systems [95, 96]. Semantic-aware approaches show generally better performance with respect to keywords-based ones,

as they allow to disambiguate textual information and to model semantic relations among words and sentences [92].
One of the advantages of content-based recommender is that they can generate recommendations using only the target user feedback, with no need to look into the behavior of other users ('user independence'). Another important advantage of content-based recommenders is that they can recommend new items, if they are similar in content to the ones that the user has liked ('new items'). Furthermore, content-based recommendations are easy to explain to the user, as the dimensions of the content that are most similar to the user profile can be highlighted together with the recommendation, e.g. "we suggest Pulp Fiction because it stars John Travolta" ('Explainability'). However, content-based recommenders typically end up in a loop where only items of the same genre are recommended, resulting in a lack of serendipity ('Overspecialization'). Furthermore, they need an item model, which requires domain knowledge (e.g. which actors are starring in a movie), strongly hinders recommendations between different types of items (e.g. hard to find content similarities between a song and a pair of shoes) and, in some cases, requires advanced modeling techniques based on semantics and/or on machine learning to create effective item representations ('Limited content analysis').

Collaborative filtering (CF) recommends to the target user items that users with similar preferences have liked in the past [14]. The traditional approach to CF is based on neighborhood models [97]. User-based systems, such as GroupLens [98], predict the interest of a user u for an item i using the ratings given to i by other users that have similar rating patterns ('neighbors'). The neighbors of user u are typically the users v whose ratings on the items rated by both u and v are most correlated to those of user u (Fig. 2.5). More formally, the predicted utility of user u for item i is given by:

$$r_{ui} = \frac{\sum_{v \in N_i(u)} w_{uv} r_{vi}}{\sum_{v \in N_i(u)} |w_{uv}|} \tag{2.8}$$

where w_{uv} is a similarity function between users and $N_i(u)$ are the nearest neighbors of user u who rated item i. Several possible similarity functions are possible, such as the cosine similarity or the Pearson correlation similarity [97]. Note that, in pure CF, the similarity function does not depend on user's characteristics, such as

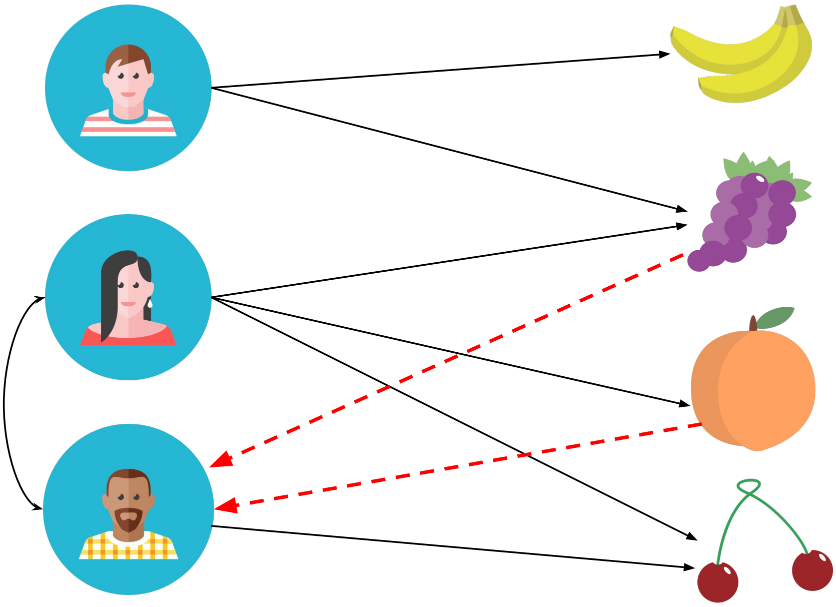


Fig. 2.5 User-based collaborative filtering. Similar users to the active user in terms of consuming patterns are identified ('neighbors'). Then, items consumed by the neighbors that the active user has not consumed yet are presented as recommendations.

demographics, but only on his/her consuming patterns. In a nutshell, in user-based collaborative filtering, r_{ui} is a weighted average of the appreciation of the item i given by similar users.

On the other hand, item-based systems [99], predict the interest of a user u for an item i based on the ratings of u for items similar to i (Fig. 2.6):

$$r_{ui} = \frac{\sum_{j \in N_u(i)} w_{ij} r_{uj}}{\sum_{j \in N_u(i)} |w_{ij}|} \tag{2.9}$$

where w_{ij} is a similarity function between items and $N_u(i)$ are the most similar items, among those already consumed by u, to the item i. Note that the equation resembles that of a content-based KNN (Eq. 2.7). Differently from content-based systems, though, the similarity of two items is high if several users of the system have rated these items in a similar fashion and not if they are similar in content. In fact, CF does not need item models at all.

Traditional neighborhood models such as [99] are still quite widespread in industry, because of their simplicity, efficiency (they require no training phase) and stability

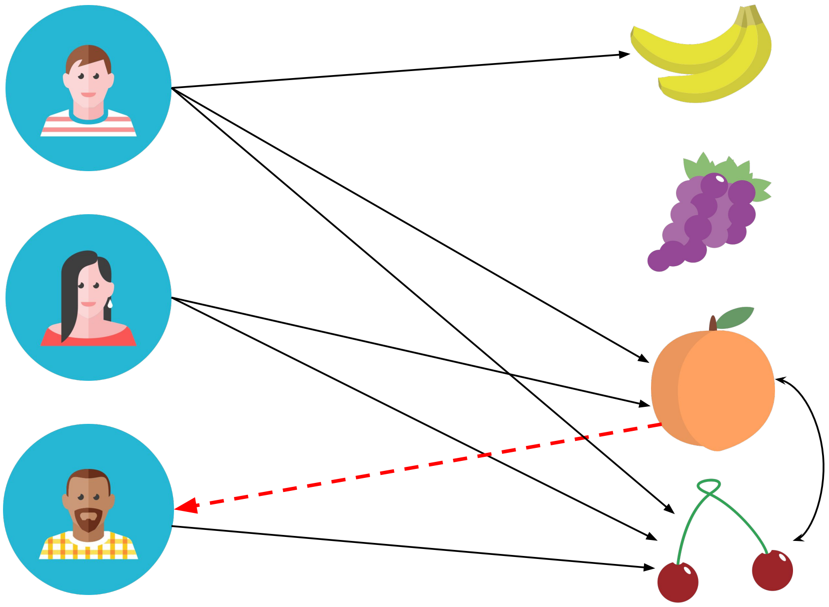


Fig. 2.6 Item-based collaborative filtering. Similar items, in terms of consuming patterns, to the items consumed by the active users are identified and presented as recommendations.

(item-based systems can make recommendations to new users once item similarities have been computed), and for their moderate transparency (in item-based approaches neighboring items can be shown to the user as explanations). However, neighborhood models have limited coverage, as they only rely on co-rated items, and are sensitive to sparse data. Latent-factor models address these issues and usually generate more accurate recommendations with respect to neighborhood models, as the Netflix prize has shown [100]. Latent-factor models are typically based on matrix factorization and learn user and item representations as vectors in a latent space from the user-item interaction matrix [101]. Take as an example Fig. 2.7, users and items are projected in the latent space. For instance, users that tend to like highly caloric fruits, will score highly on that dimension, similarly to highly caloric fruits. Users who like small fruits will score low on that specific dimension, similarly to small fruits. In this way, users end up close together in the vector space to fruits they like or may like. Note that latent dimensions do not have an explicit semantics and are often not easy to interpret. In its basic formulation, the utility function r_{ui} is modeled as a dot

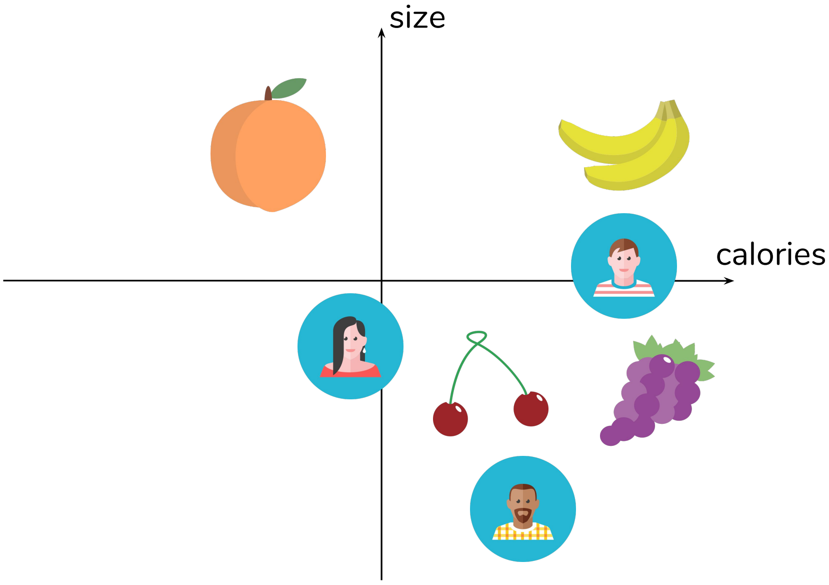


Fig. 2.7 Matrix Factorization individuates latent dimensions to explain the consumption patterns of the users.

product of the user and the item latent vector:

$$\hat{r}_{ui} = q_i^T * p_u \tag{2.10}$$

Then, the loss function becomes:

$$L(p,q) = \sum_{u,i} (r_{ui} - q_i^T * p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.11}$$

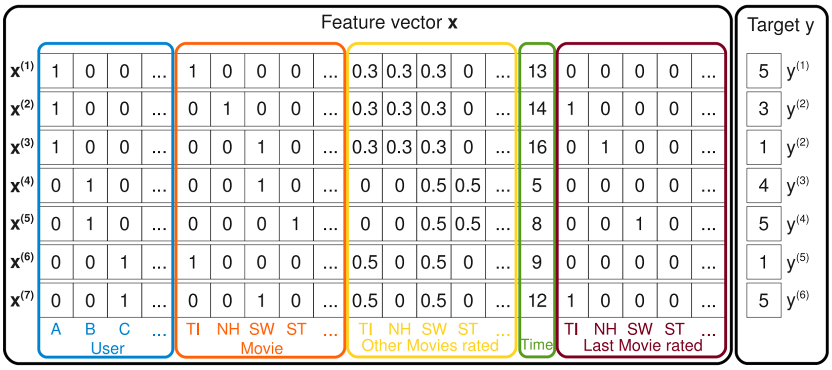
where the first term represents the squared difference between the actual ratings r_{ui} and the predicted ratings $q_i^T * p_u$ and the second term is a regularization that prevents latent factors from becoming arbitrarily large. The loss can have many additional terms, accounting for users and items biases for instance, and can be minimized using stochastic gradient descent or Alternating Least Squares [100]. Specific models have been proposed to deal with implicit feedback and to endow matrix factorization with weights accounting for different levels of confidence in the preferences of the users (Weighted Regularized Matrix Factorization) [91]. Then, since recommender systems usually provide a ranked set of items, other optimization criteria have been proposed to directly address the learning of latent factors as a ranking problem [102].

A very popular algorithm for collaborative filtering that combines the strength of neighborhood-based and model-based approaches is the Sparse LInear Method (SLIM) [103]. SLIM uses the same linear prediction function of an ItemKNN model, but the item similarity matrix W is learned by minimizing a loss function L. The loss function includes a least-square term to obtain accurate recommendation, an L1 regularization term to have a sparse W and an L2 regularization term to prevent overfitting.

Collaborative filtering is arguably the most widespread recommendation approach in industry. Nevertheless, collaborative filtering typically suffers the cold start problem, namely it cannot generate recommendations for new users and items and it does not perform well when the dataset is very sparse [100]. Take as an example Fig. 2.6, since no user has consumed grapes, grapes are a new item and cannot be recommended to any user using CF. On the other hand, content-based filtering could recommend it, was it similar in content to fruits that the user has liked. Besides, collaborative filtering systems can only generate explanations in terms of similar items, whereas content-based systems can leverage the item content. Consider the example of Fig. 2.6. The model could easily come up with an explanation such as: “People who like cherries are also interested in oranges”. This type of explanation is quite common in modern e-commerce portals, as it is easy to generate using CF systems. However, explanations that take into account the item content, such as “This orange comes from your favorite farmer” or “This orange is biological”, cannot be generated.

Hybrid recommender systems put together content-based and collaborative filtering logics in the attempt to take the best of both worlds [16, 104]. Different strategies have been experimented to create hybrid recommender systems [13]. In some cases, predictions of CBF and CF systems are combined *a posteriori*, for instance using a voting scheme [105]. In other cases, content-based profiles are used to boost collaborative filtering systems [106] and viceversa [107]. Most researchers are currently working on a single unified model that combines both content-based and collaborative filtering [108, 109]. The work presented in this thesis belongs to this family.

Factorization Machines (FMs) are popular algorithms to create hybrid recommender



	Feature vector **x**																				Target y	
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...		
	User				Movie					Other Movies rated						Last Movie rated						

Fig. 2.8 Factorization Machine matrix model. User-item interactions are modeled as feature vectors x that contain an index of the user, an index of the item, additional information such as user and/or item properties or contextual information such as the time of the interaction. Picture is taken from the original paper [2].

systems using a single unified model [2]. FMs work with an extremely sparse matrix, containing a one-hot encoding of users, items and additional information about users and items (Fig. 2.8). Then, the model equation is:

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} < v_i, v_j > x_i x_j \tag{2.12}$$

where $w_0 \in \mathbb{R}$, $w \in \mathbb{R}^n$, and:

$$< v_i, v_j > = \sum_{f=1}^{k} v_{if} v_{jf} \tag{2.13}$$

are the parameters to estimate. k is a hyper-parameter of the model that determines the dimensionality of the factorization. Note that FMs reduce to matrix factorization, if the matrix of Fig 2.8 only contains the first and second group, i.e. the user and item indexes. For this reason, factorization machines can be seen as an extension of matrix factorization methods, which can also encompass additional user and item information in the form of feature vectors, and are thus ideal to create hybrid recommender systems. Factorization machines have also been used to model other information about the user-item interaction, such as contextual information [110].

Another popular algorithm for creating hybrid recommender systems is the Sparse LInear Method with Side Information (S-SLIM) [111], an extension of the SLIM method that includes side information about the items. S-SLIM uses information about the item content to create constraints on the learning of the item similarity matrix W that is used to make recommendations. A first approach ('Collective') assumes that, if two items are similar in content, they should also be similar in co-purchase behavior. Thus, the same similarity matrix W that predicts purchase behavior should also predict item features. An alternative approach assumes that the item-item similarity matrix W can be in turn derived from a weighted combination of the item features, where the weights are learned from the optimization problem.

Knowledge graphs are an ideal structure for hybrid recommender systems. They are able to model the interactions of multiple users with multiple items, which can be used for collaborative filtering, and interactions between items and other entities, which can be used as an 'item model' for content-based filtering. In the next section, we describe the most relevant and recent works on RSs using knowledge graphs, i.e. 'Knowledge-aware Recommender Systems'.

2.2.2 Knowledge-aware Recommender Systems

In the past years, several works have shown the effectiveness of using knowledge graphs to generate recommendations. Many also make use of Linked Open Data to create KGs for recommender systems, as they represent a wealth of multi-domain knowledge about the items of the recommendations [112]. As mentioned earlier in Sec. 2.2.1, a typical application of semantics and KGs in recommender systems is the construction of an item model. For instance, in [94], the authors extend the popular Vector Space Model (VSM) [113] from the Information Retrieval research field to model items in a KG. In VSM, a textual document d is represented as a n-dimensional vector:

$$d = (w_1, w_2, \dots, w_n) \tag{2.14}$$

where w_1 are *term frequency-inverse document frequency* (TF-IDF) weights associated to the words in the document and n is the size of the vocabulary. The intuition of [94] is that in a KG, two entities s_1 and s_2 are related if there are many triples with the same property and object. Thus, each item i, under a specific property p, is represented as :

$$i^p = (w_1^p, w_2^p, \dots, w_n^p) \tag{2.15}$$

w_j^p are TF-IDF weights, where the TF of a term is defined as the frequency of node j as object of the triple (i, p, j). The similarity between items is given by a weighted average of the cosine similarity between the property-wise representation of items, which becomes part of a nearest neighbor model similar to Eq. 2.7.

Many works have also used KGs to create hybrid recommenders. For instance, the same extended VSM of [94] is combined with collaborative filtering for a LOD-based event recommender systems in [114].

In [115], the authors propose HeteRec, an approach based on a hetereogeneous information network including user-item interactions and item relationships to other entities. They use the graph to diffuse the observed user implicit feedback along different meta-paths (i.e. the sequence of types of the edges of a path) to generate recommendation candidates. Then, they apply matrix factorization to compute user and item latent factors, which are combined in a global recommendation model.

In [116], the authors propose three methods to generate KG-based recommendations based on probabilistic logic programming. The first, EntitySim, uses only the links of the graph, the second, TypeSym also uses the types of the entities, and the third, GraphLF, combines graphs with matrix factorization.

In SPrank [117, 42], the authors start from a knowledge graph including user-item interactions and item relations to other entities. They define a set of features based on the number of meta-paths connecting users to items, which are then combined in a global score through a learning to rank algorithm.

Knowledge graphs are also valuable for their ability of generating effective explanations. As discussed in Sec. 2.2.1, explanations can be generated in terms of neighboring items, users, or item content and KGs can generate all of these explanations, as they typically include all the necessary information. In [15], the authors propose a method that jointly ranks items and entities in the KG so that entities can be used as an explanation of the recommendations. In [118], the authors propose ExpLOD, a framework that generate explanations of recommendations creating a graph connecting items liked by the user in the past with the recommended items through the properties available in the KG. In [119], the authors augment an autoencoder

architecture with semantic information from DBpedia to generate content-based explanations for recommendations. In [120], the authors propose a method that jointly learns association rules between items in a knowledge graph and a recommender model to generate explainable recommendations.
These systems are based on manually defined vector models and similarity functions. In the next section, we discuss approaches based on knowledge graph embeddings.

Knowledge graph embeddings for recommender systems In the past couple of years, some works have applied knowledge graph embeddings to recommender systems. In [121], the authors use translational models to create knowledge graph embeddings that are then combined with embeddings of the items' content (textual and visual knowledge) to initialize a matrix factorization. This paper represents an interesting example of how knowledge graph embeddings can be combined with other representations build from multimedia content in order to foster the recommendation quality.

In [95], RDF2Vec embeddings are used to create item representations for a content-based Item-KNN recommender system. The model can be seen as an improvement over the model of [94], where the VSM model of the item has been replaced by an entity embedding that preserves the structure of the graph in the vector space. RDF2Vec embeddings have also been used to create a hybrid recommender system as feature vectors of a Factorization Machine in [88].

In [122], the authors use recurrent neural networks to learn representations of different meta-paths connecting users and items that are then aggregated through a pooling and a fully connected layer.

In [123], the authors experiment with different graph embedding techniques on a knowledge graph, showing that the combination of collaborative and content-based information leads to better result in all cases. In [124], the authors create metapath-aware embeddings using DeepWalk [77], which are then combined using an aggregation function and used to initialize a matrix factorization for rating prediction. This work shares some similarities with the work presented in this thesis (Chapter 3), in the way it builds KG embeddings using a graph embedding algorithm using one metapath at the time and then merges them. The first important difference is that it is based on DeepWalk rather that on node2vec, which is more primitive in its random walk strategy (see the discussion above in Sec. 2.2.1). Furthermore, the use of metapaths rather than single properties leads to a more complex recommendation model, not as easy to interpret and configure as entity2rec (see Sec. 3.4.3), and to a complexity that scales with the number of metapaths rather than with the number of properties.

2.2.3 Sequence-aware Recommender Systems

As described in Chapter 1, in many domains, such as the tourist one, it is important to generalize the discussion to RSs that recommend sequences of items, i.e. Sequence-Aware Recommender Systems (SARS) [24].
LBSN represent a great source of data about users' movement in a city, which can leveraged by RSs to provide personalized recommendations. One of the distinctive features of LBSN data with respect to traditional location prediction systems, which are mainly based on GPS data and focus on physical mobility [125], is the rich categorization of POIs in consistent taxonomies, which attribute an explicit semantic meaning to users' activities. The availability of venue categories has opened new research lines, such as statistical studies of venues peculiarities [126], automatic creation of representations of city neighborhoods and users [127, 128], definition of semantic similarities between cities [129]. Most importantly, venue categories play an important role in POI recommender systems, as they enable to model user interests and personalize the recommendations [130]. These works, though, do not account for temporal correlations among the visits of POIs, and are thus not suitable to generate tourist paths. On the other hand, the goal of the next POI prediction problem is to predict a sequence of venues ('a tourist path') that can be interesting for a given user, given some training sequences of previously liked POIs. For instance, in [131] the authors propose a matrix factorization method including personalization and geographic constraints that attempts to predict the next check-in of the user based on her past activities and geographical factors. Feng *et al.* [132] proposed a metric embedding algorithm that captures both the sequential information and the

individual preference. Such metric is used to create a Markov chain model capable of representing the transition probabilities between a given POI and the next one. The key features that are implicitly considered in the embedding creation phase are the conceptual similarity and the geographical distance of the analyzed venues. These two studies directly develop a model to recommend the next POI, while in [133] the authors focus on modeling sequences of venue categories. They propose a framework that uses a mixed Hidden Markov Model to predict the most likely next venue category and recommend POIs belonging to the most likely next category in the neighborhood of the user. Another interesting work is that of [134], where the authors address the next POI prediction problem using a personalized Markov Chain with latent patterns optimized using Bayesian Personalized Ranking.
As we mentioned in Chapter 1, the problem of next POI prediction share important features with that of itinerary recommendation, which aims at recommending sequences of POIs by solving the orienteering problem. Typically, to each POI a score is assigned based on popularity and/or personal preferences, travel times between POIs are inferred from data, and the problem of itinerary recommendation is tackled as an optimization problem where the objective is to maximize the total possible score of the itinerary while complying with the constraints [135, 22, 136].

Another common application domain for SARS is music. The task of generating music playlists was already discussed and presented in a seminal work by Herlocker *et al.* in 2004 [137]. The authors suggested that it would be interesting to be able to suggest, in the music domain, not only the songs that will be probably liked by a certain user, but an entire playlist of songs that is globally pleasing. This problem was later addressed by Chen *et al.* [138], who designed and implemented a recommender system capable of generating personalized playlists by modeling them as Markov chains. The problem of playlist generation has received a lot of attention lately, thanks to the ever increasing use of streaming platforms for music content, such as Spotify, Amazon Music or Apple music, which allow users to create, edit and listen to music playlists. For these reasons, the RecSys2018 challenge [139], sponsored by Spotify, has dealt with the *automated music playlist continuation* task. The results of the challenge are summarized in [140] and provide important insights on the best performing approaches to the problem. The challenge has shown that most of the top-performing systems use a combination of well known methods for RS such as matrix factorization and learning to rank models, often in a two-stage approach. A high recall model is first used to identify a large set of potential candidate songs for the music playlist, and then a second model, typically based on learning to rank, is used to rank the candidates and obtain the final suggested songs. Neural networks were also used to learn playlist representations using autoencoders, Convolutional Neural Networks (CNN) or RNNs. In this thesis, we describe our participation to the RecSys2018 challenge, where we propose a RNN architecture (Sec. 5.5), which achieved a good ranking. Recurrent Neural Networks (RNNs) have received a great deal of attention in machine learning research lately [141], as, thanks to their improved architectures [142, 143] and the advancements in computational power, they are able to effectively model sequences. For this reason, they have been used successfully for tasks such as speech recognition [144], sentiment analysis [145], image captioning [146] and neural language models [23]. One of the typical applications of RNN in the field of language modeling is that of generating text by recursively predicting the next word in a sentence [147].

One of the oldest applications of SARS is that of predicting Web navigation patterns. Zhou *et al.* [148] describe a web RS based on a sequential pattern mining algorithm. The training set is given by the access logs of a website and the goal of the system is to predict the next web page that a user will visit, given her previously visited pages. Interactions between users and web pages are also the typical object of prediction of session-based recommenders [149]. In session-based recommender systems, no long-term information is available about the user, and the goal is to predict the next interaction of the user with the system, given a set of short-term interactions. A common example is predicting where the user will click next, given the history of the session's clicks. This is a very active line of research, as it deals with a very common problem in websites for which users normally do not authenticate with a personal profile. Interestingly, it is one of those problems where RNNs have most commonly been used in the RS research field [24].

Another application of SARS is the market basket analysis, where the goal is to use the sequence of previous transactions in order to predict what a customer is going to buy next [150]. For example, Rendle *et al.* [151] proposed a method based on personalized transition graphs over Markov chains, while Wang *et al.* [152] designed a recommender capable of modeling both the sequential information from previous purchases and the overall preferences by a hybrid representation.

2.2.4 Evaluation of Recommender Systems

Most of the research in the RS field has focused on developing new algorithms for recommendations, and as discussed in the previous sections, a plethora of approaches exist nowadays to generate recommendations. How to decide which algorithm suits best an application among the large number of available systems? Typically, such a decision is made through experimental evaluations. Evaluations can be divided into three families: *offline experiments*, *user studies* and *online evaluations* [153].

In offline experiments, a pre-collected dataset of user's choices is used and an evaluation protocol is set up with the aim of simulating the behavior of real users. For instance, the Movielens1M [154] dataset contains 1M ratings of users on movies. It can be split into a training set to train the model and into a validation set to test the model's performance, simulating a real recommender systems.

In user studies, a set of test subjects is selected and asked to perform a number of tasks while interacting with the recommender system. For instance, they can be asked to express their appreciation for a set of recommended items. User studies are much more flexible than offline experiments, as they can be designed to answer questions that cannot be trivially simulated (e.g. 'How surprised were you in receiving this recommendation?'). Furthermore, they are closer to a real production environment, where the ultimate goal of the RS is to improve the satisfaction of users. However, user studies are more time-consuming and expensive to run than offline experiments and are impossible to repeat in the same exact conditions, hindering comparability and reproducibility. Also, great care has to be put in the selection of the subjects in reproducing the population of the real RS.

In online evaluations, data is gathered from the users' behavior while they interact with the real application. Online evaluations are meant to directly measure the system's goal, such as the improvement in the click-through rate given by a specific RS technique. These tests in real production environments are risky, as they expose users to techniques that are still under development. Thus, typically offline experiments and user studies are run before running real online evaluations.

In this thesis, we mostly use offline experiments, as they are the most widespread practice among researchers, thanks to their reproducibility and relative ease of implementation.

The first ingredient for the offline evaluation of a RS is the definition of a set of purposeful metrics. Different properties of a RS can be tested and prediction accuracy is the most widely used, as it is reasonable to assume that, the more a RS is accurate in its predictions, the more it will be appreciated by the user. On top of this, prediction accuracy is simple to measure in offline experiments and several metrics have been devised for this purpose. For the task of *rating prediction* (see Sec. 2.2.1), commonly used metrics are the Root Mean Squared Error (RMSE) or the Mean Absolute Error (MAE) between the predicted and the actual ratings. For the *top-N item recommendation problem* (see Sec. 2.2.1), metrics from Information Retrieval (IR) such as precision@N, recall@N, Non-Discounted Cumulative Gain (NDCG@N) [155] are used to assess the accuracy of the ranking of items. When using IR metrics, an important point to be specified is the candidate generation process, i.e. what are the candidate items that are ranked by the RS [156]. Three protocols are commonly used [157]:

- **All Items:** for each user, all items in the catalog are candidate items for the recommendations, both in the training and in the test set. This protocol assumes that items can be consumed multiple times by the user. It also assumes that items that are not appearing in the test set of the user are negative examples.

- **All Unrated Items:** for each user, all items in the catalog that he/she has not already rated in the training set are candidate items for the recommendations. This protocol is similar to All Items, but assumes that the user can only consume the item once. The All Unrated Items is described in detail in Sec. 3.3, being the one used in this thesis.

- **Rated Test Items:** for each user, only the rated items in his/her test are used as candidates. This allows to not make the assumption that unrated items are negative examples. However, it does not reproduce well a real environment, where the goal of the RS is to rank all the items in the catalog, rather than those that the user has already consumed.

Metrics of the IR family are gaining popularity in the evaluation of RS [90] and are used in this thesis (Sec. 3.3), together with metrics that go beyond pure accuracy. Indeed, in a survey on RS evaluation [137], the authors review several metrics, suggesting that accuracy alone is not sufficient to measure the RS quality. For instance, a user might find very accurate, but obvious and boring, the suggestions of a RS proposing only very popular Oscar-winning movies. For this reason, they also review and discuss metrics such as coverage, novelty, serendipity, and confidence. Coverage measures the ability of the RS of spanning the entire catalog of items in the recommendations. Novelty measures the ability of the RS of recommending items that appear as novel to the user. Serendipity measures the ability of recommending accurate and unexpected items to the user. Confidence measures the degree of confidence of the RS in the recommendation that it makes. These metrics are typically computed user-wise and then averaged across all the users of the system (except for other cases, where additional information about users is available and other partitions can be created [158]). Formal definitions of these metrics are provided in Sec. 3.3 for the entity2rec experimental setup and in Sec. 5.3.2 for Sequeval.

Another important element of an offline experiment is the splitting protocol. Jannach *et al.* [159] compared several recommendation algorithms in an offline experiment, analyzing their performance by relying on a comprehensive evaluation framework. The authors considered different splitting protocols and metrics, designed to characterize both the accuracy, in terms of rating and ranking, and the coverage of the suggested items. The results of the experimental trails suggest that some common algorithms, despite their high accuracy, tend to only recommend popular items that are probably not very interesting for the users of a real system. This problem is related to the popularity biases introduced by the offline evaluation protocol: for this reason, it is not advisable to compare different algorithms by relying only on measures related to their accuracy. In addition, different splitting protocols produce significantly different and non-comparable outcomes.

Several software tools are available with the purpose of simplifying the process of comparing the performance of recommendation algorithms. They typically include some evaluation protocols and a reference implementation of well-known techniques. However, the comparability between the results of these framework is still an issue. Said and Bellogín [160] compared several of these tools in order to check if their results are consistent. They discovered that the values obtained with the same dataset and algorithm may vary significantly among different frameworks. For this reason, it is not feasible to directly compare the scores reported by these tools, because they are obtained relying on several protocols. The discrepancies reported by the authors are mainly caused by the data splitting protocol, the strategy used to generate the candidate items, and the implementation choices related to the evaluation metrics. In light of these considerations, in this thesis, we describe in detail the experimental setup, specifying the choice in the generation of candidates for the ranking, the metrics, the splitting protocol, the exact version of the dataset used as well as the software implementations (Sec. 3.3).

2.2.5 Summary

In this section, we have introduced a set of basic notions related to the field of Recommender Systems, illustrating the different types of feedback used, the different possible formulations of the recommendation problem and the most commonly used models for RSs. We have highlighted the pros and cons of the different approaches, showing how the research trend is directed towards hybrid systems that combine the best of collaborative and content-based filtering. We have shown how KGs are ideal to build hybrid recommenders and reviewed the state-of-the-art in KG-aware recommender system, with a specific attention on methods that use KG embeddings. Then, we have discussed the extension of the recommendation problem to sequences of items and presented the state-of-the-art in sequence-aware recommender systems and their possible applications. Finally, we have discussed the thorny matter of the evaluation of RSs, remarking the importance of choosing metrics and evaluation protocols that well fit with the measurement of specific properties of interest of the RSs under analysis.

Chapter 3

entity2rec: Property-specific Knowledge Graph Embeddings for Item Recommendation

In the last decade, research on recommender systems has shown that hybrid systems generally outperform collaborative or content-based systems, addressing problems such as data sparsity, new items or overspecialization [13]. Knowledge graphs are an ideal data structure for hybrid recommender systems, as they allow to easily represent user-item interactions, and item and user properties as typed edges connecting pairs of entities. Recommender systems based on knowledge graphs have shown to generate high quality recommendations that are also easier to interpret and explain (see Sec. 2.2.2). The crucial point to use knowledge graphs to perform item recommendations is to be able to effectively define a measure of user-item relatedness on the graph. At the present time, most knowledge-aware recommender systems are based on manually engineered features based on path counting and/or random walks [112]. However, machine learning research has shown that feature engineering, in addition to being a cumbersome and time-consuming endeavor, leads to features that are task-specific and do not perform as well as those devised by feature learning algorithms [161]. Feature learning algorithms applied to a knowledge graph generate ‘knowledge graph embeddings’, which have proven to be very effective for

prediction tasks such as knowledge graph completion (see Sec. 2.1.3).
In this chapter, we address the following research questions:

RQ1 *How can knowledge graph embeddings be used to create accurate, non-obvious and semantics-aware recommendations based on both collaborative and content-based filtering?*
Within this major research question, we formulate and address four sub-research questions:

RQ1.1 *how can translational models for knowledge graph embeddings be used to generate recommendations from a knowledge graph?*

RQ1.2 *how can the graph embedding algorithm node2vec be used to generate recommendations from a knowledge graph?*

RQ1.3 *how can a recommender system use the effectiveness of node2vec in learning features from graphs, while encoding the semantics of multiple properties with the purpose of making recommendations?*

The remainder of the chapter is structured as follows. In Sec. 3.1 we provide a set of basic definitions that are used throughout the chapter, in Sec. 3.2 we describe how translational models and node2vec are applied to generate recommendations from knowledge graphs and we describe the entity2rec approach, in Sec. 3.3 we report the experimental setup, in Sec. 3.4 we describe the results of the experimental comparison between KG embeddings recommender systems and collaborative filtering systems, in Sec. 3.5 we describe the TinderBook use-case and how entity2rec can be used in a cold-start scenario. Finally, in Sec. 3.6 we summarize the chapter.
Part of the work described in this chapter has been published in the proceedings of the European Semantic Web Conference (ESWC) and itssatellite events [26, 26] and of the ACM conference on Recommender Systems (RecSys) [3].

3.1 Definitions

In this section, we introduce some definitions that will be used in the remaining of this thesis.

Definition 1 *A knowledge graph is a set $K = (E,R,O)$ where E is the set of entities, $R \subset E \times \Gamma \times E$ is a set of typed relations between entities and O is an ontology. The ontology O defines the set of relation types ('properties') Γ, the set of entity types Λ, assigns entities to their type $O : e \in E \rightarrow \Lambda$ and entity types to their related properties $O : \varepsilon \in \Lambda \rightarrow \Gamma_\varepsilon \subset \Gamma$.*

Definition 2 *Users are a subset of the entities of the knowledge graph, $u \in U \subset E$.*

Definition 3 *Items are a subset of the entities of the knowledge graph, $i \in I \subset E$. Users and items form disjoint sets, $U \cap I = \emptyset$.*

Definition 4 *A triple is an edge of the knowledge graph, i.e. $(i, p, j) \in R$ where $i \in E$ and $j \in E$ are entities and $p \in \Gamma$ is a property.*

Definition 5 *The property 'feedback' describes an observed positive feedback between a user and an item. Feedback only connects users and items, i.e. $(u, feedback, i)$ where $u \in U$ and $i \in I$.*

A depiction of the knowledge graph model for the specific case of movie recommendation is provided in Fig. 3.1.

The problem of top-N item recommendation is that of selecting a set of N items from a set of possible candidate items. Typically, the number of candidates is order of magnitudes higher than N and the recommender system has to be able to identify a short list of very relevant items for the user. More formally:

Definition 6 *Given a user $u \in U$, the set of candidate items $I_{candidates}(u) \subset I$ is the set of items that are taken into account as being potential object of recommendation.*

Definition 7 *A ranking function $\rho(u,i) : (u,i) \in U \times I_{candidates}(u) \rightarrow \mathbb{R}$ is a function that takes as inputs a user and a candidate item and assigns a score to them. The higher the score, the more relevant the item is deemed to be for user u.*

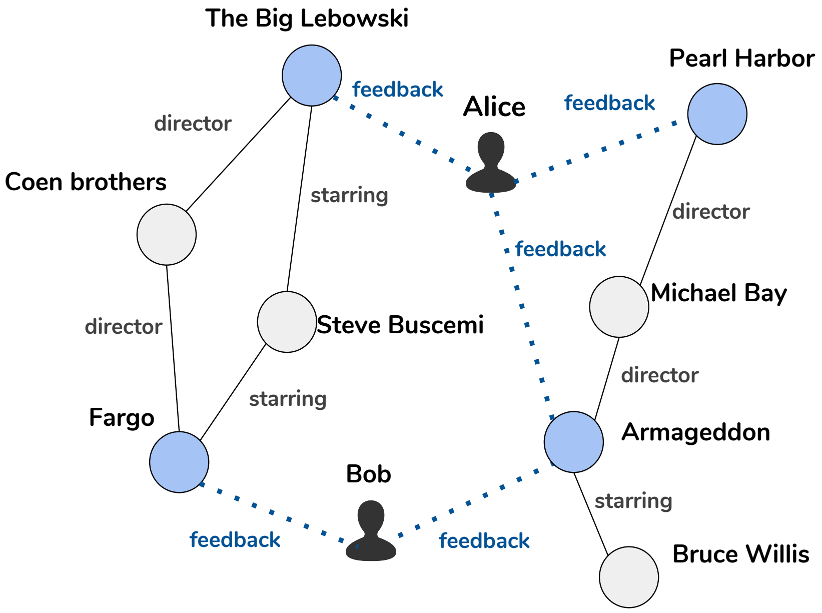


Fig. 3.1 Knowledge graph represents user-item interactions through the special property 'feedback', as well as item properties and relations to other entities. Items are represented in blue, whereas other entities are represented in grey. The knowledge graph allows to model both collaborative and content-based interactions between users and items. In this depiction, 'starring' and 'director' properties are represented as an example, more properties are included in the experiments.

Definition 8 *A ranking function $\rho(u,i)$ induces a permutation of integers corresponding to sorting the list of items $I_{candidates}(u)$ according its score:*

$$\pi(u, I_{candidates}(u)) = \{i_1, i_2, \dots, i_L\} \tag{3.1}$$

where $L = |I_{candidates}(u)|$ and $\rho(u,i_j) > \rho(u,i_{j+1})$ for any $j = 1,\dots,L-1$.

Definition 9 *Top-N item recommendation provides to each user $u \in U$ the recommended items $R(u)$, i.e. the first $N \leq L$ elements of $\pi(u, I_{candidates}(u))$:*

$$R(u) = \pi(u, I_{candidates}(u))_1^N = \{i_1, i_2, \dots, i_N\} \tag{3.2}$$

where $\rho(u,i_j) > \rho(u,i_{j+1})$ for any $j = 1,\dots,N-1$.

3.2 Knowledge Graph Embeddings Models

3.2.1 Translational Models

In this subsection, we address RQ1.1: *how can translational models for knowledge graph embeddings be used to generate recommendations from a knowledge graph?* In order to predict missing relations in a knowledge graph, most algorithms rely on feature learning approaches that are able to map entities and relations into a vector space, generating knowledge graph embeddings. Recommendations on a knowledge graph can be seen as predicting the missing feedback property (Fig. 3.2), which models the user appreciation for an item, as described in Sec. 3.1. Translational models embed entities and relations in a vector space, by seeing relations as translations in the vector space. Different models have been proposed in the past years, we compare the following:

- **TransE** [85]: learns representations of entities and relations so that $h + l \approx t$ where $(h,l,t) \in R$ is a triple. h is the 'head' entity, l is the relation and t is the 'tail' entity. The score function for a triple is thus $f(h,l,t) = D(h+l,t)$ where

D is a distance function such as the L_1 or the L_2 norm. TransE has a space complexity $O(nd+md)$ and a time complexity $O(d)$.

- **TransH** [86]: first extension of TransE, enables entities to have different representations when involved in different relations by projecting entities on a hyperplane identified by the normal vector w_l. The score function becomes: $f(h,l,t) = D(h_\perp + l, t_\perp)$, where $h_\perp = h - w_l^T h w_l$ and $t_\perp = t - w_l^T t w_l$ and D is a distance function such as the L_1 or the L_2 norm. TransH has twice more parameters to learn relations than TransE $O(nd+2md)$ and a time complexity $O(d)$.

- **TransR** [87]: enables entities and relations to be embedded in a separate vector space through a matrix M_l associated to any relation l that performs projections of vectors from entity to relation space. The score function is: $f(h,l,t) = D(h_l + l, t_l)$ where $h_l = hM_l$ and $t_l = tM_l$ and D is a distance function such as the L_1 or the L_2 norm. TransR, by introducing a separate vector space of dimension k, has a higher time and space complexity: $O(dk)$ and $O(nd+mdk)$ respectively.

TransE is the oldest and simplest among the three methods, and it is considered to be a seminal work, opening the research line and showing its promising results. TransH and TransR are conceived to allow an entity to have different representations depending on the relation that involves them (Fig. 3.2). Generally, TransH and TransR perform better than TransE, at the cost of a higher model complexity. Since results may change depending on the dataset and on the problem at hand, we experiment with all of them for the recommendation problem.

The models are trained through the minimization of a pairwise ranking loss function L that measures the total difference between the scores of ‘positive triples’ D^+ and ‘negative triples’ D^-, plus regularization terms such as the margin γ and other constraints:

$$L = \sum_{(h,l,t)\in D^+} \sum_{(h',l,t')\in D^-} max(0, \gamma + f_l(h,t) - f_l(h',t')) \tag{3.3}$$

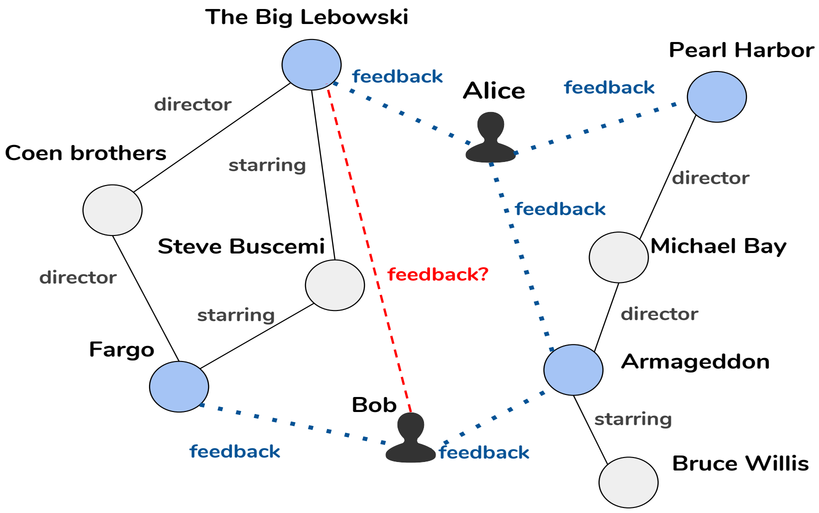
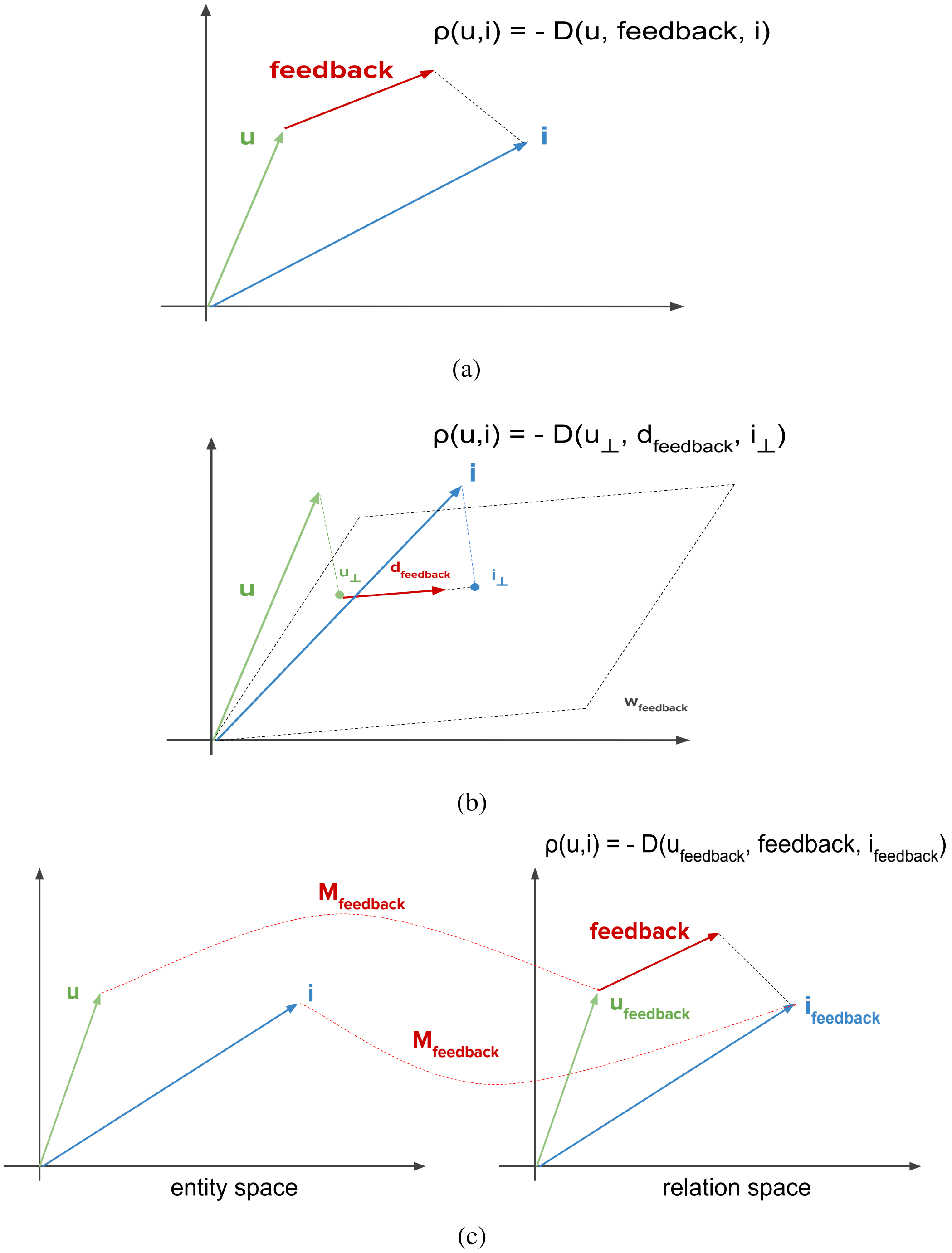


Fig. 3.2 Recommending items as a knowledge graph completion problem. Translational models are used to predict the 'feedback' property.

Positive triples D^+ are triples of the knowledge graph K, whereas negative triples D^- are obtained by 'corrupting' positive triples replacing the head or tail entities with other entities. Notice that this strategy can produce false negatives, as knowledge graphs are known to be incomplete and missing triples can still be valid facts. In order to reduce this risk, we adopt the strategy described in [86], which considers non-uniform sampling probabilities depending on the type of relation. The core idea of using translational models for item recommendation is that of using the negative score assigned to a triple $f(u, feedback, i)$ as the ranking function, i.e. $\rho(u,i) = -f(u, feedback, i)$ (Fig. 3.3).

3.2.2 node2vec

In this subsection, we address the RQ1.2: *how can the graph embedding algorithm node2vec be used to generate recommendations from a knowledge graph?*
node2vec [78] learns representations of nodes in a graph through the application of the word2vec model on sequences of nodes sampled through random walks (Fig. 3.4). The innovation brought by node2vec is the definition of a random walk exploration that is flexible and adaptable to the diversity of connectivity patterns that a network may present. Given a knowledge graph K encompassing users U, items I (the object



(a)

(b)

(c)

Fig. 3.3 (a): in TransE, user, items and relations are embedded in the same space and the ranking function is defined through the distance between the $u + feedback$ and i (b): in TransH, translations are performed on the hyperplane $w_{feedback}$ and thus the ranking function is defined through the distance between $u_{\perp} + d_{feedback}$ and $i_{\perp}$ (c): in TransR, entities and relations are embedded in different vector spaces and thus the ranking function is defined through the distance between $u_{feedback} + feedback$ and $i_{feedback}$

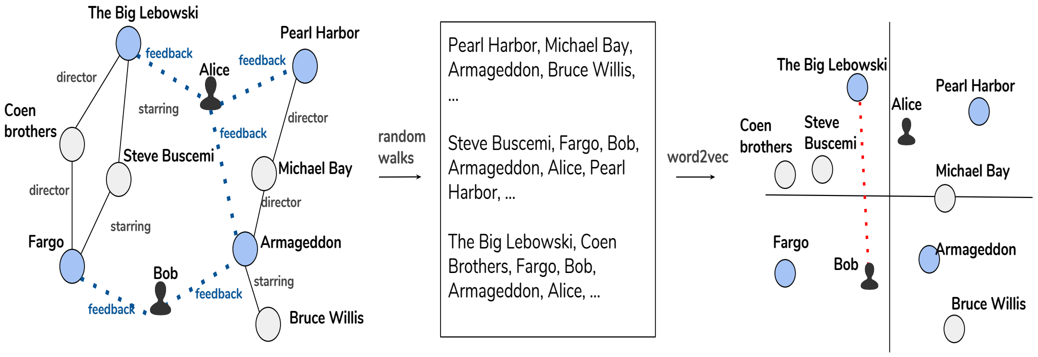


Fig. 3.4 Node2vec for item recommendation using the knowledge graph. node2vec learns knowledge graph embeddings by sampling sequences of nodes through random walks and then applying the word2vec model on the sequences. The ranking function for item recommendation is then given by the node relatedness in the vector space.

of the recommendations, e.g. a movie) and other entities E (objects connected to items, e.g. the director of a movie), node2vec generates vector representations of the users x_u and of the items x_i (and of other entities x_e). More specifically, node2vec learns a mapping $x : e \in K \to \mathbb{R}^d$, optimizing the node2vec objective function [78]:

$$\max_x \sum_{e \in K} (-\log Z_e + \sum_{n_i \in N(e)} x(n_i) \cdot x(e)) \tag{3.4}$$

where $Z_e = \sum_{v \in K} \exp(x(e) \cdot x(v))$ is the per-node partition function and it is approximated using negative sampling [75], and $N(e) \in K$ is the neighborhood of the entity e defined by the node2vec random walk. Thus, we propose to use as a ranking function the relatedness between the user and the item vectors: $\rho(u,i) = s(x(u), x(i))$ where s is the cosine similarity in this work. Note that, although in general the knowledge graph K is a directed graph, in this case we neglect the direction of properties in the random walks, as the final objective is the estimation of a symmetric user-item relatedness function.

3.2.3 entity2rec

The crucial point to leverage knowledge graphs to perform item recommendations is to be able to effectively model user-item relatedness from this rich heterogeneous network. To this end, it is highly desirable to opt for approaches that are able to automatically learn user and item representations from an optimization problem on this graph of interactions, minimizing the time-consuming endeavour of feature engineering and leading to better performance [161]. This justifies the choice of knowledge graph embeddings. At the same time, though, it is also beneficial to use a recommendation model whose features have a straight forward interpretation and that can thus be easily adapted to a specific recommendation problem. node2vec [78] has recently shown to be particularly effective at learning vectorial node representations and can be used to generate recommendations, as discussed in the previous section. However, node2vec cannot account for the diversity of semantic properties of a knowledge graph. Films can be related in terms of starring actors and not in terms of subject, can share the same director but not the same writer. Processing the whole knowledge graph altogether neglecting the semantics of the properties would not allow to account for these variations.

In this subsection, we address RQ1.3: *how can a recommender system use the effectiveness of node2vec in learning features from graphs, while encoding the semantics of multiple properties with the purpose of making recommendations?*

To address this question, we start by learning property-specific vector representation of nodes considering one property at the time, i.e. creating property-specific subgraphs K_p. Then, for each K_p independently, we learn a mapping $x_p : e \in K_p \to \mathbb{R}^d$, optimizing the node2vec objective function [78]:

$$\max_{x_p} \sum_{e \in K_p} (-\log Z_e + \sum_{n_i \in N(e)} x_p(n_i) \cdot x_p(e)) \tag{3.5}$$

where $Z_e = \sum_{v \in K_p} \exp(x_p(e) \cdot x_p(v))$ is the per-node partition function and it is approximated using negative sampling [75], and $N(e) \in K_p$ is the neighborhood of the entity e defined by the node2vec random walk. The optimization is carried out using stochastic gradient ascent over the parameters defining x_p and it attempts to maximize the dot product between vectors of the same neighborhood, i.e. to embed them close together in vector space. Similarly to what has been in done in 3.2.2, we consider K as an undirected graph. The global approach is summarized in Fig. 3.5.

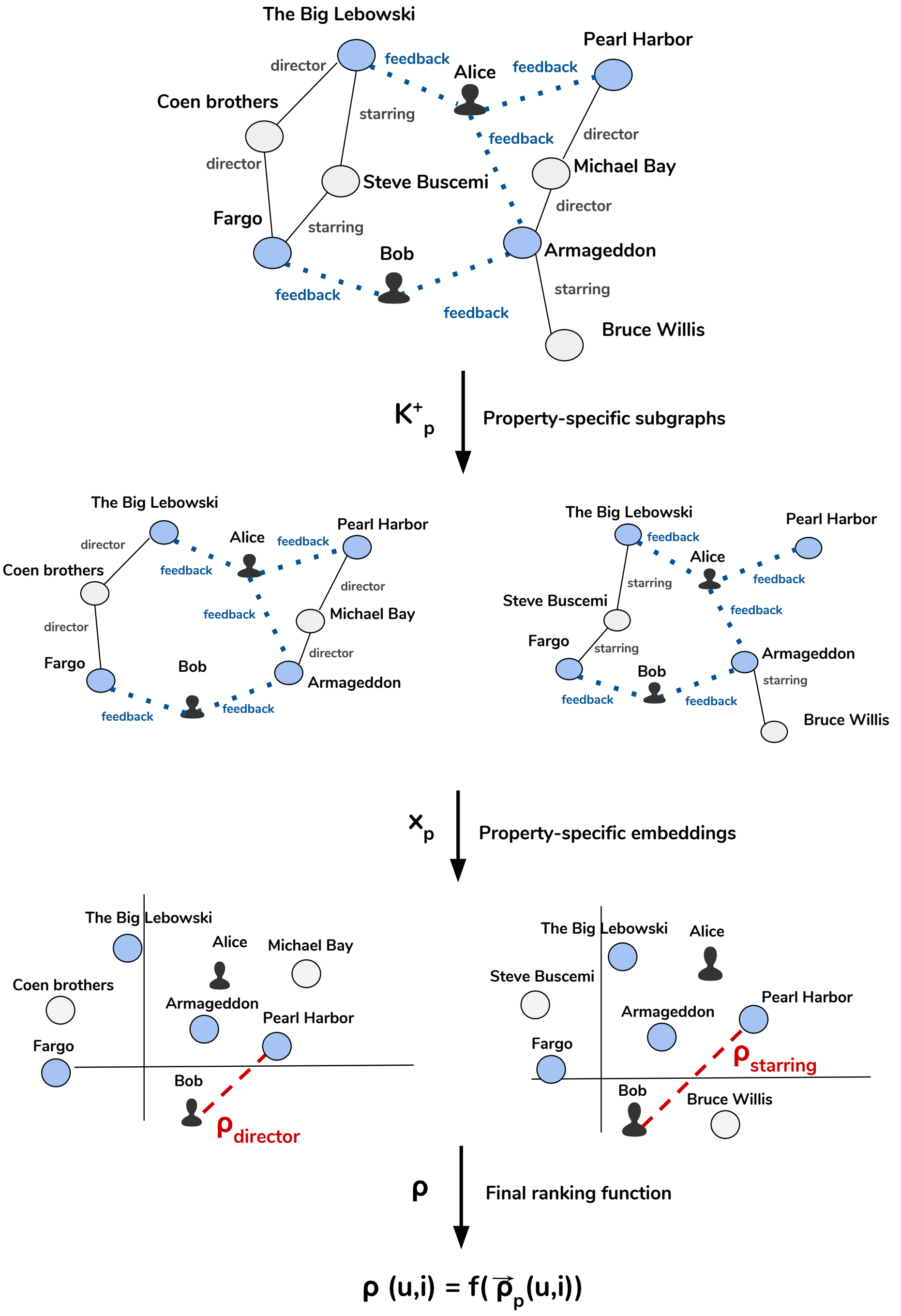


Fig. 3.5 entity2rec creates property-specific subgraphs, computes property-specific embeddings and derives property-specific relatedness scores. The property-specific relatedness scores are then aggregated to obtain a global relatedness score, which is used as a ranking function for item recommendation. The figure illustrates the case in which hybrid property-specific subgraphs are used, as described in Section 3.2.3, and where only the starring and the director properties are considered.

Property-specific subgraphs

We consider two different strategies to create property-specific subgraphs. The first one is the one presented in [3], which keeps separated collaborative and content-based information. We shall note with K_p the property-specific subgraphs created with this strategy and we will refer to it as entity2rec (2017) in Section 5.5.6. The latter is the one presented in this work, which considers hybrid property-specific subgraphs, in the sense that each of them contains both collaborative and content-based information. We refer to the hybrid subgraphs as $K^+{}_p$ and to the whole approach as entity2rec, as it has proven to be the most effective one among the two (see Section 3.4).

Collaborative-content subgraphs For each property $p \in \Gamma_\varepsilon$, we define a subgraph K_p as the set of entities connected by the property p, i.e. the triples (i, p, j). For example, if p = 'starring', we have edges connecting movies to their starring actors, e.g. (Fargo, starring, Steve_Buscemi), if p = 'subject' we have edges connecting movies to their category, e.g. (Fargo, subject, American_crime_drama_films). The only subgraph K_p containing users is that corresponding to p = 'feedback', where triples represent user-item interactions, e.g. (user201, feedback, dbr:Fargo). From the vector representations x_p, property-specific relatedness scores can be defined as follows:

$$\rho_p(u,i) = \begin{cases} s(x_p(u), x_p(i)) & \text{if } p = \text{`}feedback' \\ \frac{1}{|R_+(u)|}\sum_{i' \in R_+(u)} s(x_p(i), x_p(i')) & \text{otherwise} \end{cases}$$

where $R_+(u)$ denotes a set of items liked by the user u in the past and s denotes a measure of vector similarity. In this work, we consider s as the cosine similarity. The features include both collaborative and content information and have a straightforward interpretation. When considering p = 'feedback', K is reduced to the graph of user-item interactions and thus $\rho_{feedback}(u,i)$ models collaborative filtering. $\rho_{feedback}(u,i)$ will be high when $x_{feedback}(u)$ is close to the item $x_{feedback}(i)$ in vector space, i.e. when i has been liked by users who have liked the same items of u in the past and are thus tightly connected in the $K_{feedback}$ graph. On the other hand, when

p corresponds to other properties of the ontology O, the features encode content information. For instance, if p is 'starring', $\rho_{starring}(u,i)$ will be high if $x_{starring}(i)$ is close to items $x_{starring}(i')$, i.e. when i shares starring actors with items that the user u has liked in the past. For 'new items', i.e. with no feedback from users, we are still able to compute all the content-based features.

Hybrid subgraphs The largest issue with the use of property-specific subgraphs that consider collaborative and content-based information as separated is that often these graphs have poor connectivity, as a consequence of the fact that many properties connect one item to a few or even a single entity. This is clearly undesirable for feature learning algorithms based on random walks such as [3, 78, 77]. Consider the example of the property p = 'director'. Since most films have only one director, $K_{director}$ is similar to a set of disconnected star graphs, where each director is connected to his/her movies. In order to overcome this limitation and maintain the interpretability and explainability of the approach of using one property at the time to create features, we propose to use the 'feedback' property as a *pivot* property to create bridges between different parts of the graph, i.e. to replace K_p with $K_p^+ = K_p \bigcup \, (u, feedback, i)$, where $u \in U$ and $i \in I$. Therefore, we move from an approach where collaborative ($K_{feedback}$) and content-based (K_p with $p \neq feedback$) information is well distinguished to a set of property-specific graphs that are hybrid, as they contain both the 'feedback' information and one specific item property (Fig. 3.6).

In this case, $\Gamma_\varepsilon = \Gamma_\varepsilon \setminus feedback$. From the subgraphs K_p^+, vector representations x_p can be learned as described above and property-specific relatedness scores between a user $u \in U$ and an item $i \in I$ can be defined as follows:

$$\rho_p(u,i) = s(x_p(u), x_p(i)) \tag{3.6}$$

where s is the cosine similarity. Note that since users are now part of every subgraph K_p^+, it is no longer necessary to distinguish between collaborative features, where the relatedness score can be computed directly as in Eq. 3.6, and content-based features that require to look at the items that the user u has rated in the past, as done in [3]. For ‘new items’, i.e. with no feedback from users, we are able to compute all features.

Global user-item relatedness

For each user-item pair, we are now able to compute all property-specific relatedness scores $\vec{\rho}(u,i) = \{\rho_p(u,i)\}_{p\in\Gamma}$, either using content/collaborative subgraphs as described in Section 3.2.3 or using hybrid subgraphs as described in Section 3.2.3. We aim to consider these scores as features of a global user-item relatedness model that can be used to provide item recommendation. To this end, we experiment both an unsupervised and a supervised approach.

Unsupervised approach

In the unsupervised approach, user-item property-specific relatedness scores are combined into a single user-item relatedness score used as ranking function $\rho(u,i)$ through different possible functions such as:

$$entity2rec_{avg}(u,i) = avg(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{3.7}$$

$$entity2rec_{min}(u,i) = min(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{3.8}$$

$$entity2rec_{max}(u,i) = max(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{3.9}$$

Supervised approach

In the supervised setting, we define the global user-item relatedness $\rho(u,i;\theta) = f(\vec{\rho}(u,i);\theta)$ as a function f of the property-specific scores $\vec{\rho}(u,i)$ and of a set of parameters θ. The goal is that of finding the parameters θ that optimize top-N item recommendation as a supervised learning to rank problem [162].

Training data Given the set of users $U = \{u_1, u_2, \dots, u_N\}$, each user u_k is associated with a set of items from feedback data $\vec{i_k} = \{i_{k1}, i_{k2}, \dots, i_{kn(k)}\}$, where $n(k)$ denotes the number of feedback released by the user u_k, and a set of labels

$\vec{y_k} = \{y_{k1}, y_{k2}, \ldots, y_{kn(k)}\}$ denoting the ground truth relevance of items $\vec{i_k}$ (e.g. ratings with explicit feedback or boolean values with implicit feedback). The training set is thus represented as $\tau = \{(u_k, \vec{i_k}, \vec{y_k})_{k=1}^{N}\}$. In the case of implicit feedback, negative examples are obtained by random sampling, i.e. by randomly selecting items that the user has not interacted with.

Sorting $\rho(u,i;\theta)$ is a ranking function, meaning that, for each user u_k, the corresponding items $\vec{i_k}$ are sorted according to its score. $\rho(u,i;\theta)$ induces a permutation of integers $\pi(u_k, \vec{i_k}, \theta)$, corresponding to sorting the list of items $\vec{i_k}$ according its score (see Section 3.1).

Loss The agreement $A(\pi(u_k, \vec{i_k}, \theta), \vec{y_k})$ between the permutation $\pi(u_k, \vec{i_k}, \theta)$ induced by $\rho(u,i;\theta)$ and the list of ground truth relevance of items $\vec{y_k}$ can be measured by any information retrieval metric that measures ranking accuracy, such as P@k [163]. From this score, a loss function can be easily derived as:

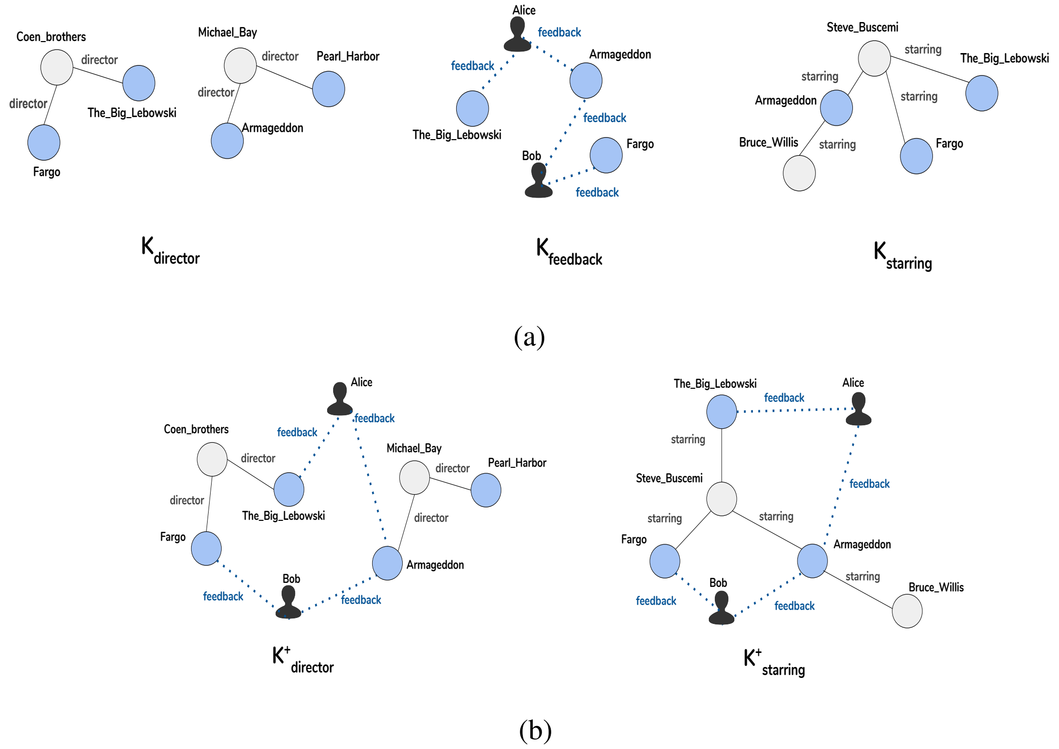
$$C(\theta) = \sum_{k=1}^{N} (1 - A(\pi(u_k, \vec{i_k}, \theta)) \tag{3.10}$$

Optimization The learning process has thus the objective of finding the set of parameters θ that minimize the loss function C over the training data:

$$\hat{\theta} = \arg\min_{\theta} C(\theta) \tag{3.11}$$

In this work, we use LambdaMart [164], a listwise learning to rank algorithm that has state-of-the-art results in learning to rank and has shown to achieve the best scores in previous work such as [3, 117]. We define:

$$entity2rec_{lambda}(u,i) = LambdaMart(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{3.12}$$



(a)

(b)

Fig. 3.6 (a): Property-specific subgraphs as defined in [3]. Collaborative and content information are separated. User-item relatedness scores can be computed directly only for $K_{feedback}$, whereas for content properties it is necessary to average the distance with respect to the items that a user has rated in the past. For properties such as “director”, the connectivity is poor. (b): Property-specific subgraphs as defined in this work. Feedback from user to items is included in every graph, improving connectivity and allowing to measure directly user-item relatedness for all properties.

Computational complexity

In this section, we derive the computational complexity of entity2rec in terms of the number of users U and the number of items I. The computational complexity of entity2rec can be divided into a component required for training and one required for testing:

$$T_{entity2rec} = T^{train}_{entity2rec} + T^{test}_{entity2rec} \tag{3.13}$$

The training of entity2rec can be in turn expressed as the training time of node2vec repeated for p times, where p is the number of distinct properties in the graph, which is the time required to generate the embeddings and the time required to learn the global relatedness score:

$$T^{train}_{entity2rec} = pT^{train}_{node2vec} + T^{train}_{global} \tag{3.14}$$

node2vec is mainly composed of three steps: one in which transition probabilities are pre-computed for each edge, one in which nodes are sampled through random walks, and one in which the word2vec model is applied to learn the embeddings [78]:

$$T^{train}_{node2vec} = T_{preprocessing} + T_{walks} + T_{word2vec} \tag{3.15}$$

The time for pre-processing transition probabilities depends on the number of edges $T_{preprocessing} = O(E)$. Since a fixed number of random walks is performed for each node and a single random walk is done in unitary time, $T_{walks} = O(N)$. Learning the embeddings with word2vec using the Skip-gram model can be done in $T_{word2vec} = O(Nlog_2N)$ as explained in [1]. Summing the three components, we obtain that:

$$T^{train}_{node2vec}(N,E) = O(E) + O(N) + (N\log_2 N) \tag{3.16}$$

Given that the number of edges $E \sim UI$ and the number of nodes $N \sim U + I$, we have:

$$T^{train}_{node2vec}(U,I) = O(UI) + O(U+I) + O((U+I)\log_2(U+I)) \tag{3.17}$$

Now, in the unsupervised case:

$$T_{global}^{train} = O(1) \tag{3.18}$$

and the total training complexity:

$$T_{entity2rec}^{train}(p,U,I) = O(pUI) + O(p(U+I)) + O(pU\log_2(U+I)) + O(pI\log_2(U+I)) \tag{3.19}$$

The time for testing is:

$$T_{entity2rec}^{test} = O(pUI) \tag{3.20}$$

as for each pair of user and items we need to compute p scores. The total complexity for entity2rec becomes:

$$T_{entity2rec}(p,U,I) = O(pUI) + O(p(U+I)) + O(pU\log_2(U+I)) + O(pI\log_2(U+I)) \tag{3.21}$$

meaning that for large values of U and I:

$$T_{entity2rec}(p,U,I) \sim O(pUI) \tag{3.22}$$

entity2rec is linear in the number of users, linear in the number of items and linear in the number of properties in the graph. Note that the dependence on p can be removed using an embarrassingly parallel implementation that distributes the generation of the embeddings on different machines, obtaining:

$$T_{entity2rec}^{parallel}(U,I) \sim O(UI) \tag{3.23}$$

3.3 Experimental setup

In this section, we describe the experimental setup, providing information about the three datasets used in the experiments, how the knowledge graph has been created, and how we have configured and evaluated the systems.

3.3.1 Datasets

The first dataset is MovieLens 1M.[1] MovieLens 1M [154] is a well known dataset for the evaluation of recommender systems and it contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users. The second dataset is the LastFM dataset,[2] which contains 92,834 listen counts of 1,892 users of 17,632 musical artists [165]. The third dataset is LibraryThing,[3] which contains 7,112 users, 37,231 books and 626,000 book ratings ranging from 1 to 10. For these three datasets, their items have been mapped to the corresponding DBpedia entities [42] and we make use of these publicly available mappings[4] to create the knowledge graphs using DBpedia data [11]. The mappings have been created semi-automatically, using DBpedia look up[12] and then checking manually suspicious cases, such as entities referenced multiple time, and items with no match. We select the most frequently occurring properties p from the DBpedia Ontology[5] (see Appendix A), and for each item property p, we include all the triples (i, p, e) where $i \in I$ and $e \in E$, e.g. (dbr:Pulp_Fiction, dbo:director, dbr:Quentin_Tarantino) in $K_{Movielens1M}$.[6] We finally add the 'feedback' property, modeling user-item interactions. Similarly to what has been done in previous work [166, 88], we add a 'feedback' edge for all movie ratings where $r \geq 4$ in X_{train}, all user-item interactions in LastFM as the dataset does not contain explicit feedback, and ratings where $r \geq 8$ for LibraryThing.

We split the data into a training X_{train}, validation X_{val} and test set X_{test} containing, for each user, respectively 70%, 10% and 20% of the ratings. Users with less than 10 ratings are removed from the dataset, as well as items that do not have a corresponding entry in DBpedia. In this process, we lose 674 movies from Movielens1M, 27 users and 7867 musical artists from LastFM, 323 users and 27305 books for LibraryThing. The final datasets statistics after this processing are reported in

[1] https://grouplens.org/datasets/movielens/1m/

[2] https://bit.ly/2s2No2Q

[3] https://www.librarything.com

[4] http://sisinflab.poliba.it/semanticweb/lod/recsys/datasets/

[5] https://wiki.dbpedia.org/services-resources/ontology

[6] dbo stands for DBpedia Ontology, dct stands for Dublin Core Terms and dbr stands for DBpedia resource.

Dataset	**Domain**	**Feedback**	**Users**	**Items**	$\rho_{\mathbf{f}}$	H
Movielens 1M	Movie	948976	6040	3226	95.13	7.17
LastFM	Music	78633	1865	9765	99.57	7.77
LibraryThing	Book	410199	6789	9926	99.39	8.26

Table 3.1 Datasets statistics. ρ_r represents the sparsity of the user-item interactions, H is the entropy of the positive feedback distribution.

Table 3.1. The sparsity of the feedback matrix ρ_f is defined as:

$$\rho_f = \frac{F}{|U||I|} \tag{3.24}$$

where F is the number of user-item interactions (e.g. ratings or implicit feedback), $|U|$ is the number of users and $|I|$ is the number of items in the dataset. The sparsity measures how many interactions, out of all the possible interactions, have actually taken place between users and items of the system. Given that typically users interact with a very limited fraction of all the items available, the sparsity of datasets for recommender systems is generally very high [167]. We observe that Movielens 1M has a much lower sparsity with respect to LastFM and LibraryThing. The entropy of the dataset is defined in terms of the distribution of positive feedback assigned by users to items:

$$H = -\sum_{i \in I} P^+(i) \log P^+(i) \tag{3.25}$$

where $P^+(i) : I \to [0,1]$ is the fraction of positive feedback attributed to the item i. The entropy of the dataset is a useful measure of the popularity bias, i.e. the effect according to which most of the positive feedback is concentrated on a few very popular items [90]. Being the entropy maximum when the distribution is uniform, a low entropy value indicates a strong concentration of feedback on very popular items, i.e. high popularity bias, whereas a high entropy value points at the contrary effect. In Figure 3.7, we represent the $P^+(i)$ distributions for the three datasets to visualize this effect. Movielens 1M and LastFM have a strong popularity bias, whereas this effect is relatively weaker for LibraryThing.

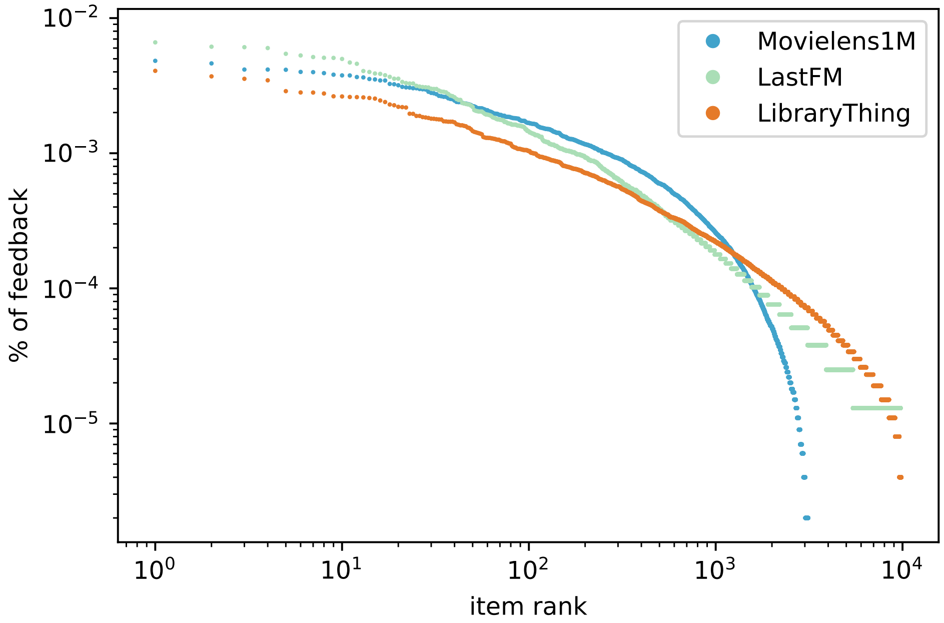


Fig. 3.7 Showing how the positive feedback is distributed among the items of the three datasets, in a log-log scale. LastFM has a strong concentration in the top-100 items, but it has a significant long tail compared to Movielens 1M. In LibraryThing, the popularity bias is weaker and the feedback is more evenly distributed among the items. These considerations are consistent with the values of entropy reported in Table 3.1.

3.3.2 Evaluation

We use the evaluation protocol known as AllUnratedItems [157], i.e. for each user, we select as possible candidate items, all the items present in the training or in the test set that he or she has not rated before in the training set:

$$I_{candidates}(u) = I \setminus \{i \in X_{train}(u)\} \tag{3.26}$$

Items that are not appearing in test set for user u are considered as negative examples, which is a pessimistic assumption, as users may actually like items that they have not seen yet. Scores are thus to be considered as a worst-case estimate of the real recommendation quality. We measure standard information retrieval metrics such as precision (P@k) and recall (R@k).

$$\mathrm{P}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\mathrm{hit}(i_j, u)}{k} \tag{3.27}$$

$$\mathrm{R}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\mathrm{hit}(i_j, u)}{|\mathrm{rel(u)}|} \tag{3.28}$$

where the value of *hit* is 1 if the recommended item i is relevant to user u, otherwise it is 0, and $rel(u)$ is the set of relevant items for user u in the test set. Differently from $P(k)$, $R(k)$ accounts for the fact that different users can have a different number of relevant items, e.g. for a user who is highly selective and likes fewer items, finding relevant items is harder.

In addition to these accuracy-focused metrics, we also decided to measure the serendipity and the novelty of the recommendations. Serendipity can be defined as the capability of identifying items that are both attractive and unexpected [168]. [169] proposed to measure it by considering the precision of the recommended items after having discarded the ones that are too obvious. Eq. 3.29 details how we compute this metric. *hit_non_pop* is similar to *hit*, but top-k most popular items are always counted as non-relevant, even if they are included in the test set of user u. Popular items can be regarded as obvious because they are usually well-known by most users.

$$\mathrm{SER}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\mathrm{hit\_non\_pop}(i_j, u)}{k} \tag{3.29}$$

In contrast, the metric of novelty is designed to analyze if an algorithm is able to suggest items that have a low probability of being already known by a user, as they belong to the long-tail of the catalog. This metric was originally proposed by [170] in order to support recommenders capable of helping users to discover new items. We formalize how we computed it in Eq. 3.30. Note that this metric, differently from the previous ones, does not consider the correctness of the recommended items, but only their novelty.

$$\mathrm{NOV}(k) = -\frac{1}{|U| \times k} \cdot \sum_{u \in U} \sum_{j=1}^{k} \log_2 \mathrm{P}_{\mathrm{train}}(i_j) \tag{3.30}$$

The function $P_{train} : I \rightarrow [0,1]$ returns the fraction of feedback attributed to the item i in the training set. This value represents the probability of observing a certain item in the training set, that is the number of ratings related to that item divided by the total number of ratings available. In order to avoid considering as novel items that are not available in the training set, we consider $\log_2(0) \doteq 0$ by definition.

We compare entity2rec to a set of state-of-the-art collaborative filtering recommender systems from the MyMediaLite library [171], which has shown to outperform competing libraries in controlled experiments [172]:

- BPRMF: a matrix factorization method where the optimization is performed using Bayesian Personalized Ranking [102].
- BPRSLIM: a Sparse LInear Method where the optimization is performed using Bayesian Personalized Ranking [103].
- ItemKNN: a K-nearest neighbor recommender based on items [99].
- LeastSquareSLIM: a Sparse LInear Method optimized for the ranking elastic net loss [103].
- MostPop: a simple baseline algorithm where the top-N popular items are recommended to every user.

- WRMF: the Weighted Regularized Matrix Factorization is a matrix factorization method where a weighting matrix is used to account for different confidence levels in the user-item feedback [91].

Furthermore, we include in the evaluation translational-based models (Sec. 3.2.1) and node2vec (Sec. 3.2.2). Finally, as a non KG-based hybrid algorithm we add the Ranking Factorization Machine (RankingFM) [2] with side information using the Turi Create library[7]. As side information, we use the entities connected to the specific items from the DBpedia KG. All the baselines have been trained on the user ratings contained in X_{train} in the original matrix format and tested on X_{test}. The implementations of the translational based embeddings[8] and of node2vec[9] are available online. entity2rec is also publicly available on GitHub.[10]

3.3.3 Configuration

We have configured entity2rec hyper-parameters by assessing the P@5 of the model on the validation set, using grid and manual searches. We have optimized the dimension of the embeddings d, the maximum length of the random walk l, the context size for the optimization c, the return parameter p, the in-out parameter q, the number of random walks per each node of the graph n (see [78] for more information on these parameters). More in detail, we started to search for hyper-parameters on the Movielens 1M dataset, making a grid search and evaluating the model on the validation set, using the ranges $p \in \{0.25, 1, 4\}$, $q \in \{0.25, 1, 4\}$, $d \in \{200, 500\}$, $l \in \{10, 20, 30, 50, 100\}$, $c \in \{10, 20, 30\}$, $n \in \{10, 50\}$. We found the optimal configuration in this range to be $C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$, and we observed that the performance was improving when increasing l and c. Thus, we have run a configuration $C2 = \{p : 4, q : 1, d : 200, l : 100, c : 50, n : 100\}$, which achieved better performance on the validation set. For LastFM, we started from the configuration $C1$, and then explored the ranges: $p \in \{1, 4\}$, $q \in \{1, 4\}$, $c \in \{30, 40, 50, 60\}$, $l \in \{60, 100, 120\}$, $n \in \{50, 100\}$. We found the configuration

[7] https://github.com/apple/turicreate
[8] https://github.com/thunlp/KB2E
[9] https://github.com/aditya-grover/node2vec
[10] https://github.com/D2KLab/entity2rec

$C3 = \{p:4, q:4, d:200, l:100, c:60, n:100\}$ to be optimal on the validation set in this range. For LibraryThing, we have explored the ranges $p \in \{1,4\}$, $q \in \{1,4\}$, $c \in \{30,50\}$, keeping $l = 100$, $n = 100$, $d = 200$. The optimal configuration is: $C4 = \{p:1, q:1, d:200, l:100, c:50, n:100\}$. The hyper-parameters of the LambdaMart [164] algorithm have been left to their default values as reported in the implementation that has been used for this work.[11] In general, we can observe that, for all datasets in consideration, using long walks ($l = 100$), many walks per entity ($n = 100$), and a large context size such as $c = 50$ improves the quality of the recommendations. This kind of configuration ought to be used as a starting point in configuring entity2rec with new datasets.

3.4 entity2rec experimental results

3.4.1 Hybrid property-specific subgraphs

In this section, we compare the new version of entity2rec based on hybrid property specific subgraphs to the one proposed in [3], i.e. entity2rec to entity2rec (2017). In Table 3.2, Table 3.3 and Table 3.4, we report statistics for the property-specific subgraphs K_p and K_p^+ for Movielens 1M, LastFM and LibraryThing respectively. It can be noticed that for many content properties, the average degree is small, indicating that items are connected to few entities, as discussed in Section 3.2.3. On the other hand, hybrid property-specific subgraphs always have a better connectivity in terms of average degree of the nodes.

We have compared entity2rec to entity2rec (2017) for the three datasets under analysis, measuring P@5, R@5, SER@5, and NOV@5. We have also added $entity2rec_{feedback}$, which only uses the feedback graph and is thus equivalent to node2vec, as a baseline to see whether entity2rec or entity2rec (2017) are more effective in using content properties. The results are reported in Table 3.5, Table 3.6 and Table 3.7. The first finding is that entity2rec consistently obtains better scores for different configurations of the hyper-parameters for the three datasets, proving

[11] https://github.com/jma127/pyltr

	$\mathbf{K_p}$			$\mathbf{K_p^+}$		
property	**N**	**M**	κ	**N**	**M**	κ
dbo:cinematography	2757	2136	1.6	9881	382636	77.4
dbo:director	4607	3142	1.4	10881	383642	70.5
dbo:distributor	3180	3180	2.0	9594	383680	80.0
dbo:editing	2292	1809	1.6	9851	382309	77.6
dbo:musicComposer	3682	3031	1.7	10322	383531	74.3
dbo:producer	4358	3817	1.8	11135	384317	69.0
dbo:starring	8969	13990	3.1	15375	394490	51.3
dbo:writer	5044	3836	1.5	11668	384336	65.9
dct:subject	9809	49897	10.2	15967	430397	53.9
feedback	9119	380500	83.5	n/a	n/a	n/a

Table 3.2 Network stats for Movielens 1 M for K_p and K_p^+. $N =$ n_nodes, $M =$ n_edges, $\kappa =$ average_degree.

	$\mathbf{K_p}$			$\mathbf{K_p^+}$		
property	**N**	**M**	κ	**N**	**M**	κ
dbo:associatedBand	15212	19492	2.6	20169	74444	7.4
dbo:associatedMusicalArtist	15211	19491	2.6	20168	74443	7.4
dbo:bandMember	9768	7587	1.6	17113	62539	7.3
dbo:birthPlace	5650	5581	1.9	12992	60533	9.3
dbo:formerBandMember	8870	7481	1.7	16621	62433	7.5
dbo:genre	10258	26064	5.1	13034	81016	12.4
dbo:hometown	9646	12386	2.6	13870	67338	9.7
dbo:instrument	2236	4411	3.9	11137	59363	10.7
dbo:occupation	2211	3246	2.9	10970	58198	10.6
dbo:recordLabel	12159	20279	3.3	15938	75231	9.4
dct:subject	25827	88375	6.8	27946	143327	10.3
feedback	10147	54952	10.8	n/a	n/a	n/a

Table 3.3 Network stats for LastFM for K_p and K_p^+. $N =$ n_nodes, $M =$ n_edges, $\kappa =$ average_degree.

	$\mathbf{K_p}$			$\mathbf{K_p^+}$		
property	**N**	**M**	κ	**N**	**M**	κ
dbo:author	13132	9757	1.5	20564	194416	18.9
dbo:country	2015	1921	1.9	16657	186580	22.4
dbo:coverArtist	1990	1438	1.4	17126	186097	21.7
dbo:language	1975	1906	1.9	16630	186565	22.4
dbo:literaryGenre	7211	8526	2.4	17292	193185	22.3
dbo:mediaType	4632	5539	2.4	16676	190198	22.8
dbo:previousWork	4845	3261	1.3	17559	187920	21.4
dbo:publisher	8696	8241	1.9	17878	192900	21.6
dbo:series	3317	2562	1.5	17319	187221	21.6
dbo:subsequentWork	5928	3930	1.3	18245	188589	20.7
dct:subject	18903	51324	5.4	26138	235983	18.1
feedback	16501	184659	22.4	n/a	n/a	n/a

Table 3.4 Network stats for LibraryThing for K_p and K_p^+. N = n_nodes, M = n_edges, κ = average_degree.

the effectiveness of using hybrid property-specific subgraphs as suggested in this work. The second finding is that using learning to rank ($entityrec_{lambda}$) is crucial to obtain good recommendations for entity2rec (2017), but it is no longer useful for entity2rec, especially if hyper-parameters are properly optimized. In order to interpret this result, we remind the reader that, as shown in [3], the most relevant information to make predictions comes from the user-item interaction, i.e. from the 'feedback' property. In entity2rec (2017), the feedback property was contained only in the 'feedback' subgraph, and thus the learning to rank was fundamental to attribute different weights to the properties. On the other hand, for hybrid property-specific subgraphs, user-item interactions are present for all the properties, and the feature learning process, with an appropriate configuration of the hyper-parameters, is able to encode effectively all the necessary information to make recommendations, so that the learning to rank algorithm appears redundant and even damaging, and a simple unsupervised approach such as averaging the features is more effective. It is also worth noticing that entity2rec (2017) might appear to have a higher novelty than entity2rec. However, this is likely due to the fact that it has a lower accuracy rather to the algorithm itself. In fact, in Tab. 3.5 if you compare the configuration

System	P@5	R@5	SER@5	NOV@5
$entity2rec_{lambda}$ (C2)	0.2125	0.0967	0.1913	9.654
$entity2rec_{avg}$ (C2)	0.2372	0.1045	0.2125	9.577
$entity2rec_{min}$ (C2)	0.2198	0.0976	0.1946	9.466
$entity2rec_{max}$ (C2)	0.2206	0.0951	0.2038	10.046
$entity2rec_{lambda}$ (2017) (C2)	0.1836	0.0748	0.1640	9.948
$entity2rec_{avg}$ (2017) (C2)	0.0578	0.0234	0.0523	11.085
$entity2rec_{min}$ (2017) (C2)	0.0166	0.0090	0.0166	11.541
$entity2rec_{max}$ (2017) (C2)	0.0099	0.0023	0.0095	12.129
$entity2rec_{feedback}$ (C2)	0.1801	0.0814	0.1629	9.881
$entity2rec_{lambda}$ (C1)	0.2221	0.0988	0.2021	9.891
$entity2rec_{avg}$ (C1)	0.2265	0.0997	0.2051	9.820
$entity2rec_{min}$ (C1)	0.2020	0.0912	0.1787	9.744
$entity2rec_{max}$(C1)	0.1954	0.0865	0.1825	10.208
$entity2rec_{lambda}$ (2017) (C1)	0.1670	0.0707	0.1521	10.369
$entity2rec_{avg}$ (2017) (C1)	0.0100	0.0052	0.0100	14.582
$entity2rec_{min}$ (2017) (C1)	0.0214	0.0107	0.0213	12.214
$entity2rec_{max}$ (2017) (C1)	0.0069	0.0020	0.0069	12.416
$entity2rec_{feedback}$ (C1)	0.1464	0.0685	0.1329	10.377

Table 3.5 entity2rec outperforms entity2rec (2017) for different configurations of hyper-parameters on Movielens 1M ($C1 = \{p:4, q:1, d:200, l:100, c:30, n:50\}$, $C2 = \{p:4, q:1, d:200, l:100, c:50, n:100\}$). In entity2rec as presented in this work, the learning to rank is no longer useful, as the unsupervised approach appears to be more effective. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision. Through the comparison with $entity2rec_{feedback}$, we see that entity2rec is more effective than entity2rec (2017) at leveraging the item content.

System	P@5	R@5	SER@5	NOV@5
$entity2rec_{lambda}$ (C3)	0.1852	0.1066	0.1512	10.101
$entity2rec_{avg}$ (C3)	0.2062	0.1191	0.1682	10.379
$entity2rec_{min}$ (C3)	0.2055	0.1191	0.1664	9.807
$entity2rec_{max}$ (C3)	0.1693	0.0986	0.1423	10.243
$entity2rec_{lambda}$ (2017) (C3)	0.1469	0.0844	0.1194	11.092
$entity2rec_{avg}$ (2017) (C3)	0.0597	0.0351	0.0574	13.143
$entity2rec_{min}$ (2017) (C3)	0.0002	0.0001	0.0002	13.09
$entity2rec_{max}$ (2017) (C3)	0.1387	0.0801	0.1063	11.426
$entity2rec_{feedback}$ (C3)	0.1542	0.0886	0.1198	11.512
$entity2rec_{lambda}$ (C1)	0.1745	0.1009	0.1405	11.227
$entity2rec_{avg}$ (C1)	0.1505	0.0870	0.1182	12.267
$entity2rec_{min}$ (C1)	0.1699	0.0981	0.1343	11.331
$entity2rec_{max}$ (C1)	0.1295	0.0753	0.1037	10.537
$entity2rec_{lambda}$ (2017) (C1)	0.1054	0.0611	0.081	12.273
$entity2rec_{avg}$ (2017) (C1)	0.0396	0.0238	0.0380	13.689
$entity2rec_{min}$ (2017) (C1)	0.0002	0.0001	0.0002	13.089
$entity2rec_{max}$ (2017) (C1)	0.0952	0.0553	0.0651	12.450
$entity2rec_{feedback}$ (C1)	0.0872	0.0508	0.0600	12.762

Table 3.6 entity2rec outperforms entity2rec (2017) for different configurations of hyper-parameters on LastFM ($C1 = \{p:4, q:1, d:200, l:100, c:30, n:50\}$, $C3 = \{p:4, q:4, d:200, l:100, c:60, n:100\}$). In entity2rec as presented in this work, the learning to rank is no longer useful, as the unsupervised approach appears to be more effective. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision. Through the comparison with $entity2rec_{feedback}$, we see that entity2rec is more effective than entity2rec (2017) at leveraging the item content.

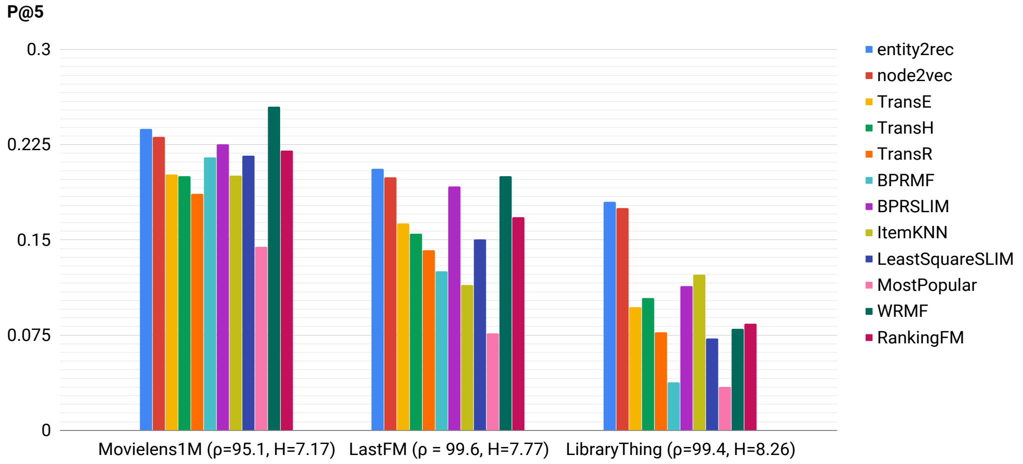
System	**P@5**	**R@5**	**SER@5**	**NOV@5**
$entity2rec_{lambda}$ (C4)	0.1271	0.0803	0.1229	12.469
$entity2rec_{avg}$ (C4)	0.1800	0.1072	0.1736	12.886
$entity2rec_{min}$ (C4)	0.1831	0.1084	0.1757	11.709
$entity2rec_{max}$ (C4)	0.1634	0.0984	0.1591	12.783
$entity2rec_{lambda}$ (2017) (C4)	0.1322	0.0746	0.1285	13.000
$entity2rec_{avg}$ (2017) (C4)	0.0720	0.0495	0.0719	13.481
$entity2rec_{min}$ (2017) (C4)	0.0060	0.0027	0.0060	13.396
$entity2rec_{max}$ (2017) (C4)	0.0319	0.0250	0.0316	14.549
$entity2rec_{feedback}$ (C4)	0.1534	0.0926	0.153	13.007
$entity2rec_{lambda}$ (C1)	0.1396	0.0815	0.1365	13.261
$entity2rec_{avg}$ (C1)	0.1678	0.1014	0.1639	12.731
$entity2rec_{min}$ (C1)	0.1735	0.1041	0.1689	12.301
$entity2rec_{max}$ (C1)	0.1411	0.0865	0.1390	13.629
$entity2rec_{lambda}$ (2017) (C1)	0.1160	0.0689	0.1132	13.444
$entity2rec_{avg}$ (2017) (C1)	0.0661	0.0453	0.0661	13.270
$entity2rec_{min}$ (2017) (C1)	0.0041	0.0018	0.0041	13.545
$entity2rec_{max}$ (2017) (C1)	0.0129	0.0122	0.0128	14.339
$entity2rec_{feedback}$ (C1)	0.1632	0.0976	0.1578	12.410

Table 3.7 entity2rec outperforms entity2rec (2017) for different configurations of hyper-parameters on LibraryThing ($C1 = \{p:4, q:1, d:200, l:100, c:30, n:50\}$, $C4 = \{p:1, q:1, d:200, l:100, c:50, n:100\}$). In entity2rec as presented in this work, the learning to rank is no longer useful, as the unsupervised approach appears to be more effective. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision. Through the comparison with $entity2rec_{feedback}$, we see that entity2rec is more effective than entity2rec (2017) at leveraging the item content

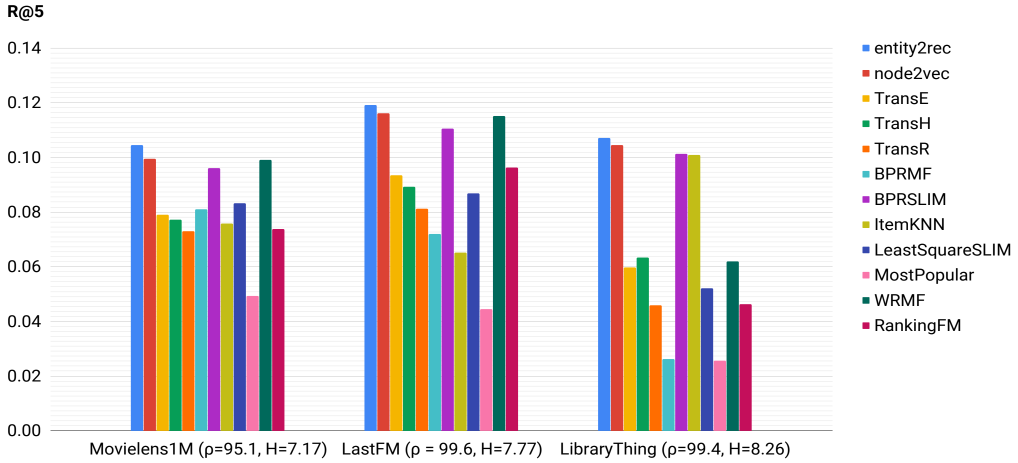
with the worst accuracy of the new approach $entity2rec_{max}(C1)$ with the one with the best accuracy of the old approach $entity2rec_{lambda}(C2)(2017)$, you can see that the novelty of $entity2rec_{max}(C1)$ is higher than $entity2rec_{lambda}(C2)(2017)$. Similar observations can be done for the other two datasets. Finally, the comparison with $entity2rec_{feedback}$, which is equivalent to using node2vec on the collaborative graph only, allows us to draw some observations on the effect of content properties. Results show that, for all datasets and all configurations, entity2rec using hybrid property-specific subgraphs has better scores than $entity2rec_{feedback}$, implying that, content information when encoded in hybrid property-specific subgraphs, is useful to enhance recommendations. For $entity2rec(2017)$ this is true for Movielens1M and for the (C1) configuration of LastFM, where $entity2rec_{lambda}(2017)$ outperforms $entity2rec_{feedback}$, but not for the (C3) configuration of LastFM and for LibraryThing. We can therefore conclude that, in agreement with our argument of the graph connectivity, the most effective way to encode content-based information to improve the quality of recommendation is that of the hybrid property-specific subgraphs.

3.4.2 Comparison with other recommender systems

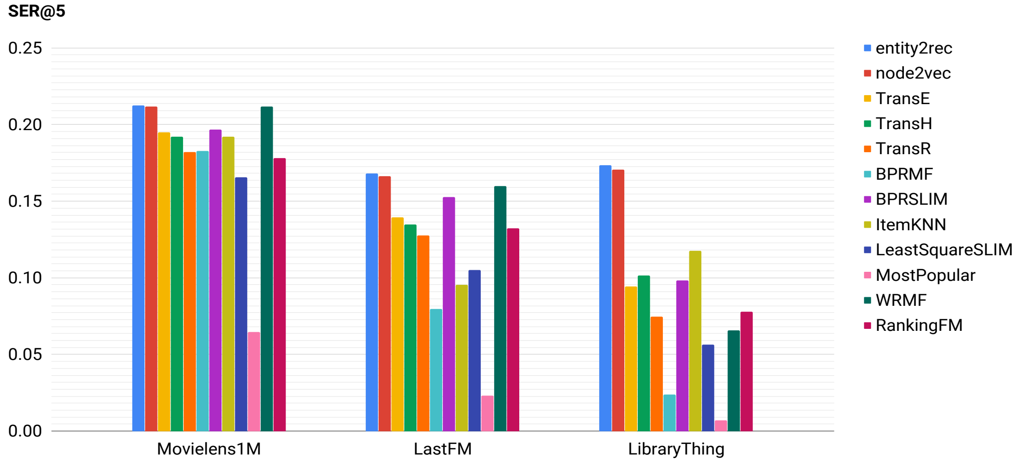
We have measured P@5, R@5, SER@5, and NOV@5 for all the recommender systems and the three datasets under analysis. We can see from Figure 3.8a, Figure 3.8b, and Figure 3.8c that entity2rec outperforms competing systems for all datasets for P@5, R@5, and SER@5, except for P@5 in Movielens 1M where WRMF is performing better. As we can see from Figure 3.8d, WRMF is characterized by low novelty of the recommendations, i.e. it tends to recommend very popular items. This proves to be effective in Movielens 1M and LastFM that have a high popularity bias as shown by the entropy value (Table 3.1), the distribution of items (Figure 3.7), previous literature [90] and the effectiveness of the MostPopular baseline. Looking at SER@5, i.e. considering the top-5 items as non relevant, entity2rec has a slightly better score than WRMF. Movielens 1M is also characterized by a lower sparsity with respect to the other datasets, and this favors collaborative filtering systems, which are known to suffer from data sparsity [13]. In fact, all of collaborative filtering systems, even ItemKNN that does not perform dimensionality reduction, achieve scores above


P@5
0.3
0.225
0.15
0.075
0
Movielens1M (ρ=95.1, H=7.17)
LastFM (ρ = 99.6, H=7.77)
LibraryThing (ρ=99.4, H=8.26)
entity2rec
node2vec
TransE
TransH
TransR
BPRMF
BPRSLIM
ItemKNN
LeastSquareSLIM
MostPopular
WRMF
RankingFM

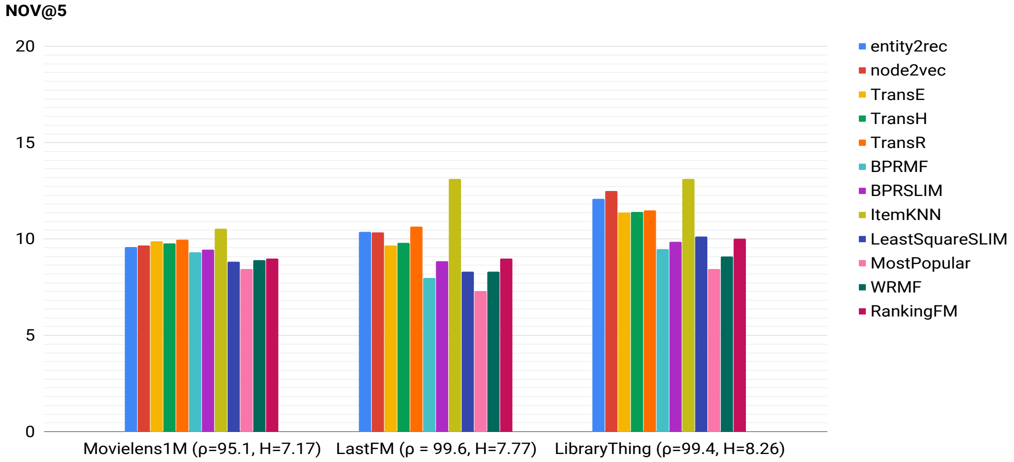
(a)


R@5
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Movielens1M (ρ=95.1, H=7.17)
LastFM (ρ = 99.6, H=7.77)
LibraryThing (ρ=99.4, H=8.26)
entity2rec
node2vec
TransE
TransH
TransR
BPRMF
BPRSLIM
ItemKNN
LeastSquareSLIM
MostPopular
WRMF
RankingFM

(b)



(c)



(d)

Fig. 3.8 Results on Movielens 1M, LastFM, and LibraryThing datasets for P@5, R@5, SER@5, and NOV@5. entity2rec performs well for all datasets, but it is especially effective for LibraryThing, where sparsity and entropy are high. Scores are reported in tabular form in Appendix B. entity2rec refers to $entity2rec_{avg}(C1)$, $entity2rec_{avg}(C2)$, and $entity2rec_{avg}(C3)$ for Movielens 1M, LastFM, and LibraryThing respectively. node2vec refers to $node2vec(C1)$, $node2vec(C2)$ and $node2vec(C3)$ for Movielens 1M, LastFM, and LibraryThing respectively. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

20%. LastFM is much sparser than Movielens 1M, and this affects the performance of ItemKNN, whereas matrix factorization based techniques maintain good scores. In LibraryThing, where the popularity bias is weaker (Figure 3.7), entity2rec is significantly more effective than competing systems and matrix factorization techniques are much less well performing with respect to the other two datasets. ItemKNN, on the other hand, is outperforming matrix factorization techniques. Looking at R@5 (Figure 3.8b) we can see that entity2rec outperforms all competing systems, also in the Movielens 1M dataset, where WRMF has a better precision. Since R@5 is weighting the number of hits by the number of relevant items for the user, this shows that entity2rec generally works better than other systems with users having fewer relevant items.

In terms of novelty, ItemKNN is the best performing system (Figure 3.8d), but it is not comparable to competing systems in terms of P@5, R@5 and SER@5. Recommender systems based on knowledge graph embeddings (entity2rec, node2vec, translational models) have better novelty with respect to collaborative filtering systems. We also observe that entity2rec outperforms node2vec for all the datasets and for P@5, R@5, and SER@5, justifying the creation of property-specific embeddings that are aggregated in a later stage, rather than embedding the whole knowledge graph. Furthermore, we observe that using knowledge graph embeddings approaches based on neural language models such as entity2rec and node2vec is more effective than using translational models.

In general, we can say that entity2rec generates accurate (high precision and recall) and non-obvious (high serendipity, good novelty) recommendations and it is particularly effective with respect to state-of-the-art systems when the sparsity and the entropy of the datasets are high (e.g. LibraryThing). A tabular representation of the scores on the three datasets is reported in Appendix B.

3.4.3 Model Interpretability

In this section, we address the question of the interpretation of the entity2rec model. We focus on $entity2rec_{avg}$, as it proved to be more effective on the three datasets. $entity2rec_{avg}$, as described in Section 3.2.3, is the average of property-specific relatedness scores:

$$entity2rec_{avg}(u,i) = avg(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{3.31}$$

where the property-specific relatedness scores are obtained from the embeddings of the property-specific subgraphs, containing user-item interactions and item relations p to other entities. With respect to most knowledge-aware recommender systems based on metapaths, the interpretation is thus easier, as it considers one property at the time. We report in Table 3.8, Table 3.9, and Table 3.10 the scores of using a single property ρ_p as a ranking function. We can see that for all datasets, the information coming from “dct:subject" is quite relevant. Then, for movies, the starring actors and the director appear to be strong features, for musical artists the record label and the genre, and for books previous and subsequent work. In general, none of the features individually outperforms the average of the features.

The simplicity of the final ranking function and the ability to interpret the model in an easy way has several positive consequences. An in-depth discussion of this point is matter of future work, but we mention two big advantages. The first is that entity2rec can be easily used in an interactive interface, as a sort of multicriteria recommender system, replacing Eq. 3.31 with a weighted average of the property-specific relatedness score. For example, the user might assign more weight to the “director" property and recommendations could change accordingly. The second is the possibility of explaining recommendations along different dimensions. Explanations can be generated in terms of related items, looking at what items of the users' profile are more similar to the recommended ones using the global $\rho(u,i)$ (e.g. “Because you liked Titanic"); in terms of properties, by comparing the property-specific relatedness scores $\rho_p(u,i)$ and using the highest scores to unravel specific properties of the item to which the user is more related (e.g. “Because you may like the cast"); in terms of item content, since property-specific relatedness scores $\rho_p(u,e)$ can be measured between a user u and any entity of the knowledge graph $e \in E$, not just for items $i \in I$ (e.g. “Because you may like Steve Buscemi").

$\mathbf{K_p^+}$	**P@5**	**R@5**	**SER@5**	**NOV@5**
dbo:cinematography	0.1847	0.0813	0.1675	9.835
dbo:director	0.1913	0.0842	0.1741	9.859
dbo:distributor	0.1846	0.0805	0.1673	9.894
dbo:editing	0.1829	0.0810	0.1668	9.855
dbo:musicComposer	0.1861	0.0817	0.1691	9.891
dbo:producer	0.1777	0.0826	0.1603	10.349
dbo:starring	0.2113	0.0937	0.1965	9.957
dbo:writer	0.1808	0.0822	0.1652	**10.393**
dct:subject	**0.2249**	**0.0958**	**0.2044**	9.831

Table 3.8 Feature evaluation for Movielens 1M dataset. The most effective features appear to be the subject, the starring actors and the director of the movie. The writer and the producer introduce more novelty. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

$\mathbf{K_p^+}$	**P@5**	**R@5**	**SER@5**	**NOV@5**
dbo:associatedBand	0.1539	0.0894	0.1253	11.05
dbo:associatedMusicalArtist	0.1575	0.0915	0.1299	10.95
dbo:bandMember	0.1511	0.0873	0.1217	11.60
dbo:birthPlace	0.1612	0.0925	0.1287	11.08
dbo:formerBandMember	0.158	0.0909	0.1274	**11.48**
dbo:genre	0.1801	0.1042	0.1466	10.33
dbo:hometown	0.1708	0.0979	0.1371	10.36
dbo:instrument	0.1601	0.0919	0.1270	11.25
dbo:occupation	0.1457	0.0844	0.1103	10.96
dbo:recordLabel	0.1856	0.1076	0.1532	10.42
dct:subject	**0.1954**	**0.1131**	**0.1655**	10.19

Table 3.9 Feature evaluation for LastFM dataset. The most effective features appear to be the subject, the record label and the genre. The instruments and the former band players introduce more novelty. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

K_p^+	P@5	R@5	SER@5	NOV@5
dbo:author	0.1603	0.0972	0.1556	**12.736**
dbo:country	0.1629	0.0976	0.1572	12.360
dbo:coverArtist	0.1625	0.0973	0.1571	12.362
dbo:language	0.1619	0.0971	0.1558	12.350
dbo:literaryGenre	0.1633	0.0978	0.1583	12.353
dbo:mediaType	0.1610	0.0956	0.1559	12.411
dbo:previousWork	0.1688	0.1001	0.1630	12.523
dbo:publisher	0.1643	0.0979	0.1588	12.326
dbo:series	0.1635	0.0976	0.1581	12.460
dbo:subsequentWork	0.1687	0.1007	0.1634	12.520
dct:subject	**0.1733**	**0.1037**	**0.1684**	12.031

Table 3.10 Feature evaluation for LibraryThing dataset. The most effective features appear to be the subject, the previous and following works. The author introduces more novelty. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

3.5 Use-case: the Tinderbook application

In this section, we show how entity2rec deals with the *new user* problem, i.e. we address RQ1.4. We introduce the problem of book recommendation in a cold start scenario and describe the Tinderbook application, how entity2rec works within it, its evaluation and the lessons learned from the experimentation. Tinderbook is available online at: www.tinderbook.it.

3.5.1 Cold start: item-based book recommendations

In recent years, the explosion of information available on the Web has made ever more challenging the task of finding a good book to read. In 2010, the number of books in the world was more than one hundred millions[12] and approximately 2,210,000 new books are published every year[13]. At the same time, a survey shows that in the US a reader typically reads 4 books in one year[14] and a study shows that on average, fewer than half of the books are finished by the majority of readers and

[12] https://bit.ly/36akNY8

[13] https://bit.ly/2DRSr94

[14] https://bit.ly/2Lz6uo9

that most readers typically give up on a book in the early chapters[15]. These figures show the importance and the complexity of the process of selecting a book to read among the enormous amount of available options. Recommender systems (RS) have provided a great deal of help in this task, using algorithms that predict how likely it is for a user to like a certain item, leveraging the history of the user preferences [6]. Most of the existing book recommender systems are typically based on collaborative filtering, which suffers from the cold start problem [13], and are thus based on long onboarding procedures, requiring users to log-in and to rate a consistent number of books (Section 3.5.5).

The goal of the Tinderbook application is that of recommending books to read, given a single book that the user likes. In a more formal way, we need to define a measure of item relatedness $\rho(i_j, i_k)$ which estimates how likely it is that the user will like the book i_k, given that the user likes the book i_j. The item relatedness $\rho(i_j, i_k)$ is used as a ranking function, i.e. to sort the candidate items $i_k \in I_{candidates}(u)$ given the 'seed' item i_j. Then, only the top N elements are selected and presented to the user. The approach to define the measure of item relatedness $\rho(i_j, i_k)$ is based on entity2rec. We apply entity2rec to generate property-specific knowledge graph embeddings, but then, we focus on item-item relatedness, rather than on user-item relatedness, given the absence of the user profile. In light of the experimental results described in Sec. 3.4, we use the 'average' as the final aggregation function. Property-specific item-item relatedness scores are then averaged to obtain a global item-item relatedness score that is used as a ranking function (Figure 3.9). We define:

$$\rho_{entity2rec}(i_j, i_k) = avg(\{\rho_p(i_j, i_k)\}_{p \in \Gamma}) \tag{3.32}$$

where $\rho_p(i_j, i_k) = cosine\_sim(x_p(i_j), x_p(i_k))$ and x_p is the property-specific knowledge graph embedding obtained using node2vec on the hybrid property-specific subgraph. The only difference with respect to Eq. 3.31 is that, rather than measuring user-item relatedness, we are now measuring item-item relatedness. We compare this measure of item relatedness with that of an ItemKNN [99], which is a purely collaborative filtering system. The relatedness between the items is high when they

[15] https://nyti.ms/36fk6wy

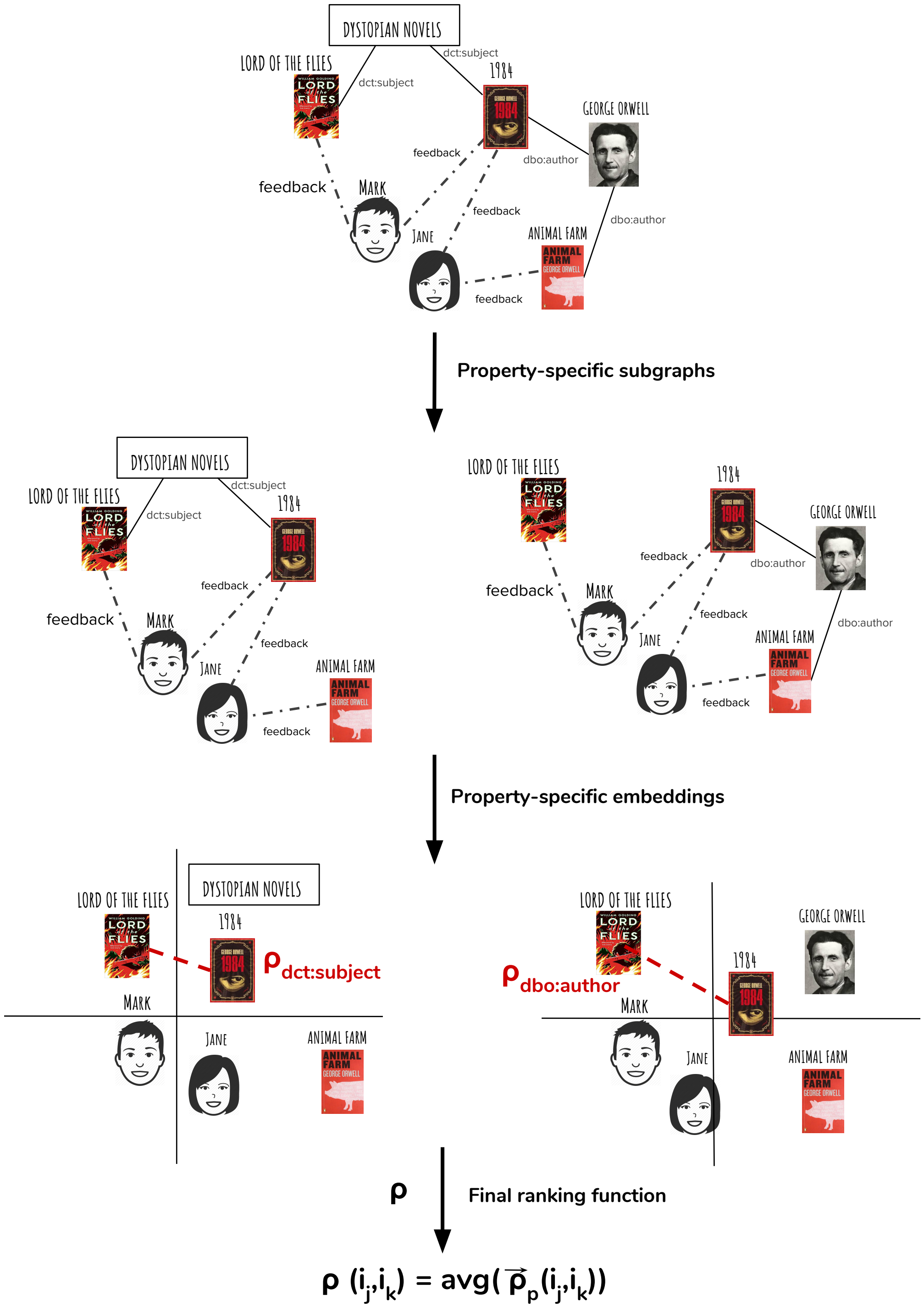


Fig. 3.9 The knowledge graph represents user-item interactions through the special property ‘feedback’, as well as item properties and relations to other entities. The knowledge graph allows to model both collaborative and content-based interactions between users and items. In this figure, ‘dbo:author’ and ‘dct:subject’ properties are represented as an example, more properties are included in the experiments. Property-specific subgraphs are created from the original knowledge graph. Property-specific embeddings are computed, and property-specific item relatedness scores are computed as cosine similarities in the vectors space. Finally, property-specific relatedness scores are averaged to obtain a global item-item relatedness score.

tend be liked by the same users. More formally, we define:

$$\rho_{ItemKNN}(i_j, i_k) = \frac{|U_j \cap U_k|}{|U_j \cup U_k|} \tag{3.33}$$

where U_j and U_k are the users who have liked item i_j and i_k respectively. We also use as a baseline the MostPop approach, which always recommends the top-N most popular items for any item i_j. Then, we compare entity2rec with a content-based measure of item relatedness based on a cosine similarity between TF-IDF vectors (see Sec. 2.2) built using DBpedia properties of an item as if they were "words" of a document:

$$\rho_{TFIDF}(i_j, i_k) = cosine\_sim(TFIDF(i_j)), TFIDF(i_k)) \tag{3.34}$$

Finally, we compare entity2rec with a measure of item relatedness based on knowledge graph embeddings built using RDF2Vec [88]. RDF2Vec turns all DBpedia entities into vectors, including the books that are items of the recommender system. Thus, we simply use as a measure of item relatedness the cosine similarity between these vectors:

$$\rho_{RDF2Vec}(i_j, i_k) = cosine\_sim(RDF2Vec(i_j), RDF2Vec(i_k)) \tag{3.35}$$

where $RDF2Vec(i_j)$ stands for the embedding of the item i_j built using RDF2Vec. Note that this is a purely content-based recommender such as the one implemented in [95], as DBpedia does not contain user feedback.

3.5.2 Offline evaluation

The dataset used for the application and for the offline evaluation is LibraryThing and we use the evaluation protocol known as AllUnratedItems [157]. The offline experiment simulates the scenario in which the user selects a single item he/she likes i_j (so-called 'seed' book) and gets recommendations according to an item-item relatedness function $\rho(i_j, i_k)$, which ranks the candidate items i_k. We iterate through the users of the LibraryThing dataset, and for each user we sample with uniform

System	P@5	R@5	SER@5	NOV@5
entity2rec	**0.0549**	**0.0508**	**0.0514**	11.099
ItemKNN	0.0484	0.0472	0.0463	12.200
TFIDF	0.0322	0.0283	0.0312	12.568
RDF2Vec	0.0315	0.0288	0.0311	**13.913**
mostpop	0.0343	0.0256	0.0070	8.452

Table 3.11 Results for different item-item relatedness measures. entity2rec provides more accurate recommendations with respect to pure collaborative filtering such as ItemKNN and to the Most Popular baseline. It also scores better with respect to the content-based TF-IDF and RDF2Vec, although RDF2Vec has the best novelty. Scores can be considered as without error, as the standard deviation is negligible up to the reported precision.

probability an item i_j that he/she liked in the training set. Then, we rank the candidate items $i_k \in I_{candidates}(u)$ using $\rho_{entity2rec}(i_j, i_k)$, $\rho_{itemknn}(i_j, i_k)$, $\rho_{TF-IDF}(i_j, i_k)$, $\rho_{RDF2Vec}(i_j, i_k)$ and MostPopular, and we measure P@5, R@5, SER@5, NOV@5. The results show that entity2rec obtains better precision, recall and serendipity with respect to competing systems (Table 3.11).

3.5.3 Application

In this section, we describe the Tinderbook application.

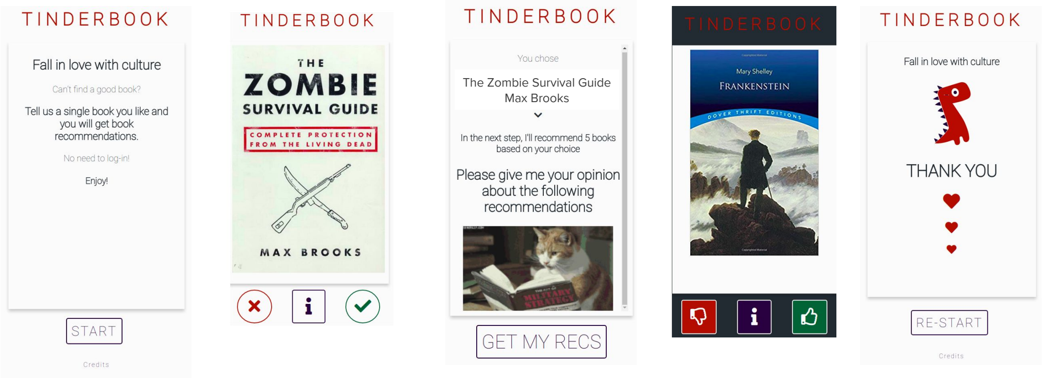
Session

A complete usage session can be divided in two phases (Figure 3.10):

1. **Onboarding**: the user lands on the application and gets books that are sampled with a chance that is proportional to the popularity of the book. More in detail, a book is sampled according to:

$$p(book) \sim P^{+}(book)^{\frac{1}{T}} \tag{3.36}$$

   where P^{+} is the popularity of the book, which is defined as the fraction of positive feedback (ratings $r \geq 8$) obtained by the book in the LibraryThing dataset. T is a parameter called "temperature" that governs the degree of



Fall in love with culture

Can't find a good book?

Tell us a single book you like and you will get book recommendations.

No need to log-in!

Enjoy!

You chose

The Zombie Survival Guide
Max Brooks

In the next step, I'll recommend 5 books based on your choice

Please give me your opinion about the following recommendations

Fall in love with culture

1. ONBOARDING

2. RECOMMENDATIONS

Fig. 3.10 A complete session of use of the application. The user selects a book that she likes, gets book recommendations based on her choice and provides her feedback. User can get info about the book by pressing on the “Info” icon.

randomness in the sampling. $T \to 0$ generates a rich-gets-richer effects, i.e. most popular books become even more likely to appear in the extraction. On the contrary, when T grows the distribution becomes more uniform, and less popular books can appear more often in the sampling. The user has to discard books (pressing “X" or swiping left on a mobile screen) until a liked book is found. The user can get additional information about the book (e.g. the book abstract from DBpedia) by pressing on the “Info” icon.

2. **Recommendations**: after the user has selected a book (“seed book”), she receives five recommended books based on her choice, thanks to the item-item relatedness $\rho_{entity2rec}$ (see Section 3.5.1). The user can provide feedback on the recommended books using the “thumbs up” and “thumbs down” icons, or swiping right or left. The user can again get additional information about the book (book abstract from DBpedia) by pressing on the “Info” icon.

The graphical user interface of Tinderbook aims to engage users using playful interaction on popular like/dislike interaction [173]. The graphical representation of cards and a slot-machine-like interaction engage the users into an infinite swipe left and right loop as the popular Tinder interface [174]. We choose to adopt digital cards interface for Tinderbook because it can be applied to a variety of contexts and, combined with ubiquitous swipe gesture, can alleviate information overload and

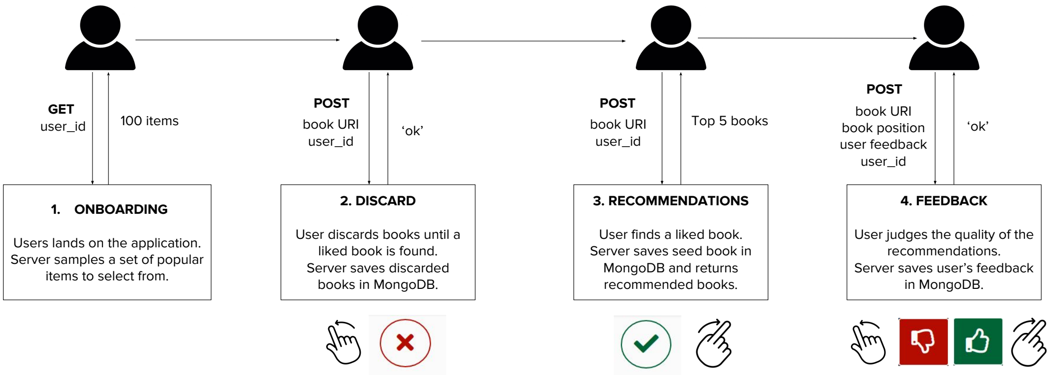

GET
user_id
100 items
1. ONBOARDING
Users lands on the application. Server samples a set of popular items to select from.
POST
book URI
user_id
'ok'
2. DISCARD
User discards books until a liked book is found. Server saves discarded books in MongoDB.
POST
book URI
user_id
Top 5 books
3. RECOMMENDATIONS
User finds a liked book. Server saves seed book in MongoDB and returns recommended books.
POST
book URI
book position
user feedback
user_id
'ok'
4. FEEDBACK
User judges the quality of the recommendations. Server saves user's feedback in MongoDB.

Fig. 3.11 Tinderbook interactions and corresponding API calls. In 1. ONBOARDING, books are sampled with a chance proportional to their popularity, as described in Eq. 3.36. In 2. DISCARD, the user goes through proposed books until he/she finds a liked book. In 3. RECOMMENDATIONS, the user receives five book recommendations related to his/her chosen book. In 4. FEEDBACK, the user judges the quality of the recommendations.

improve the user experience aspect of apps[16]. Moreover, Tinderbook can further leverage engagement data, i.e. each individual user-swipe interaction, to get insights on users' satisfaction in the usage of the application. The interactions of the user are mapped into API calls to the server, as described in Figure 3.11.

Architecture

The overall architecture is presented in Figure 3.12. DBpedia is the main data source for the application. DBpedia is queried to get book title, author and abstract. Google images is queried to retrieve thumbnails for images, using the book title and author extracted from DBpedia to disambiguate the query. The model is a key-value data structure that stores item-item similarities as defined in Eq. 3.32 and it is used to get the five most similar books to the chosen book. MongoDB is used to store the discarded books, seed books, and the feedback on the recommended books ("thumbs up" or "thumbs down"), in order to evaluate the application in the online scenario (Section 5.5.6). Book metadata are collected once for all the books at the start of the server and kept in memory to allow faster recommendations.

[16] https://www.nngroup.com/articles/cards-component/

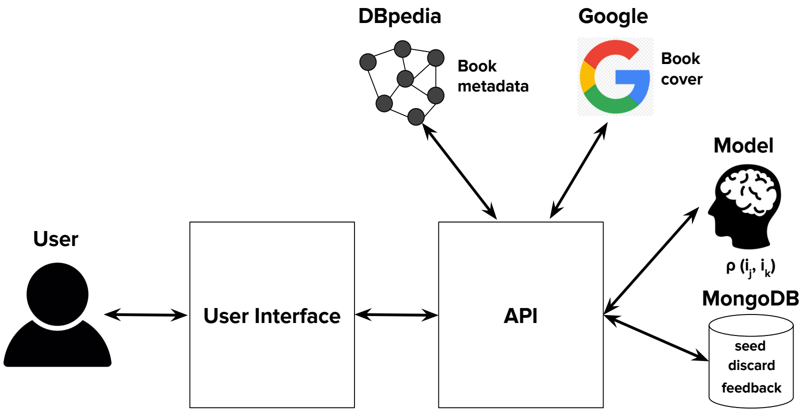


Fig. 3.12 The architecture of the Tinderbook application. The user interacts through the user interface, which makes calls to the API. In turn, the API interacts with the similarity model to get recommendations, DBpedia to enrich book descriptions, Google to get book covers and MongoDB to save books chosen ('seed') and discarded ('discard') in the onboarding phase and the users' feedback in the recommendation phase ('feedback').

3.5.4 Online Evaluation

Tinderbook has been deployed on Nov 22nd, 2018. In this section, we report the results of usage data collected for two weeks, going from Nov, 22nd to Dec, 6th (Table 3.12). In order to evaluate the application, we have defined a set of Key Performance Indicators (KPIs), which are specific to the online scenario, in addition to the metrics defined in Section 3.3.2. In the online experiment, we define the recommendation as a 'hit' if the user provides positive feedback ("thumb up" or swipe right), and as a 'miss' if the user provides negative feedback ("thumb down" or swipe left) in the recommendation phase. Recall cannot be measured in the online experiment, as we do not have a test set to measure $rel(u)$.

Definition 10 *We define **completeness** as the average percentage of rated books per session, given that the user has entered the recommendation phase.*

Definition 11 *We define **discard** as the average number of discarded books in the onboarding phase*

Definition 12 *We define **dropout** as the percentage of users who leave the application during the onboarding phase*

	All	T=0.3	T=1.
tot. # seeds	470	358	112
tot. # feedback	1,936	1495	441
tot. # discarded books	3,668	2263	1405

Table 3.12 Total usage stats for the online experiment for the whole experiment (22 Nov - 6 Dec), for $T = 0.3$ configuration only (22nd, Nov - 29th, Nov) and for the $T = 1.$ configuration only (30th, Nov - 6th, Dec).

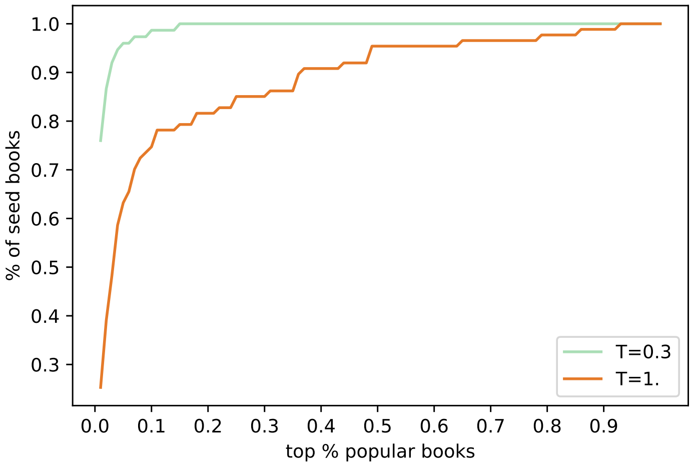


Fig. 3.13 Showing how different values of the temperature affect the popularity of the books chosen as “seeds” for the recommendations in the onboarding phase. In both cases, seeds are strongly concentrated among the most popular books. However, in the case of $T = 0.3$ the effect is stronger, with all of the seeds falling into the top 20% most popular books. In the case of $T = 1.$, roughly 80% of the seed books fall into the top 20% most popular books.

Definition 13 *We define **seed popularity** as the average popularity of the seed books*

Definition 14 *We define **recommendation time** τ as the average time required to provide the list of recommended books in the recommendation phase*

In the first week, we have experimented an onboarding phase with a temperature parameter set to $T = 0.3$. In the second week, we have increased this temperature to the value of $T = 1$. As described in Section 3.5.3, the temperature T governs the degree of randomness in the popularity-driven book sampling of the onboarding phase. The first effect observed as a consequence of the increase in temperature in the onboarding phase was the fact that less popular books were chosen during the onboarding phase. In Figure 3.13, we represent the distribution of seed books falling in the top x% popular items for $T = 0.3$ and $T = 1$. The picture shows that $T = 1$ has made less popular books appear more frequently in the choices of the users in the onboarding phase with respect to the initial configuration $T = 0.3$. However, it is worth noticing that most seed books are still concentrated among the most popular books (80% in the top 20% popular books). The change of temperature has also had effects on the other KPIs. In order to compare the two different onboarding configurations $T = 0.3$ and $T = 1.$, we have measured the KPIs mean values and standard deviations and run a statistical test to assess whether the observed differences were statistically significant or not. More specifically, we have run a Welch's t-student test [175] with a confidence value of $\alpha = 0.05$, only $p < \alpha$ are considered as statistically significant. As shown in Table 3.13, the onboarding configuration $T = 1.0$ decreases the average popularity of the seed books in a statistically significant way. This leads to the fact that users have to discard more items before finding a liked book in the onboarding phase, as it can be noticed by the increase of the average number of discarded books. However, the number of dropouts does not increase in a statistically significant way, meaning that we cannot say that this fact is pushing users to get bored during the onboarding and leave the application more easily. In fact, it shows that users are engaged enough to keep using the application even if they have to discard more books in the onboarding. Interestingly, the configuration with $T = 1.0$ is also increasing the novelty, meaning that less popular books also appear more often in the recommendations. Overall, we can claim that $T = 1.0$ is the best configuration for the application, as it leads to more novelty without significantly increasing the number of dropouts.

The recommendation time is very short, roughly 10 milliseconds, as it involves accessing values from a key-value data store, which can be done in unitary time. More specifically, we measure $\tau = 0.012448 \pm 0.000255$ across the whole experiment.

3.5.5 Competing Systems

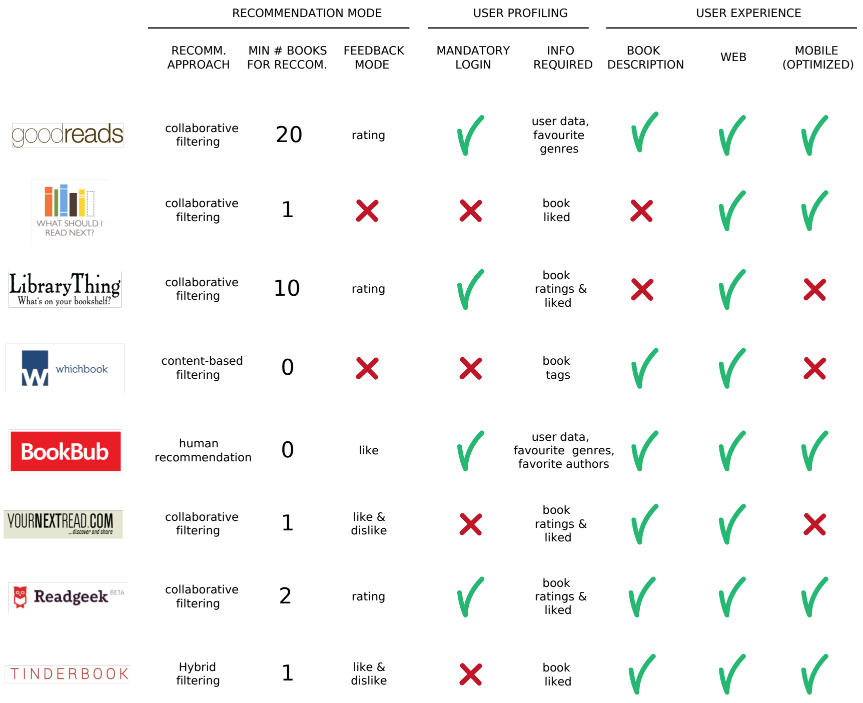
Existing book recommender systems are typically based either on content-based or collaborative filtering [176]. In Figure 3.14, we report a comparison of Tinderbook

	T = 0.3	**T = 1.**	**p value**	**significant**
P@5	0.497368 ± 0.026381	0.495833 ± 0.052701	9.79E-01	no
SER@5	0.417105 ± 0.024892	0.437500 ± 0.047382	7.07E-01	no
NOV@5	8.315443 ± 0.176832	10.095039 ± 0.347261	2.30E-05	**yes**
completeness	0.903947 ± 0.018229	0.937500 ± 0.025108	2.86E-01	no
discard	6.321229 ± 0.663185	12.544643 ± 2.070238	2.09E-03	**yes**
dropout	0.131285 ± 0.019150	0.178571 ± 0.039930	2.45E-01	no
seed pop	0.002626 ± 0.000060	0.000835 ± 0.000086	2.74E-48	**yes**

Table 3.13 Tinderbook KPIs for the $T = 0.3$ configuration only (22nd, Nov - 29th, Nov) and for the $T = 1.$ configuration only (30th, Nov - 6th, Dec). Welch's t-student test is used to compare the KPIs with a confidence value $\alpha = 0.05$.

with existing book recommender systems. The first point that makes Tinderbook stand out from competitors is the recommendation algorithm, a hybrid approach based on knowledge graph embeddings. In the past years, several works have shown the usefulness of knowledge graphs for recommender systems, and more specifically, of Linked Open Data knowledge graphs [112]. More in detail, knowledge graphs are often used to create hybrid recommender sytems, including both user-item and item-item interactions. For instance, in [117] and in [42], the authors use hybrid graph-based data model utilizing Linked Open Data to extract metapath-based features that are fed into a learning to rank framework. Recently, some works have used feature learning algorithms on knowledge graphs, i.e. knowledge graph embeddings for recommender systems, reducing the effort of feature engineering and resulting in high-quality recommendations [124, 121, 88, 26, 27]. In particular, entity2rec [3], on which Tinderbook is based, has shown to create accurate recommendations using property-specific knowledge graph embeddings.

The second point that makes Tinderbook stand out is the Graphical User Interface (GUI) and the quick onboarding process, with no necessity of log-in or account creation. Card-based GUI are a great way to deliver information at a glance. Cards help avoid walls of text, which can appear intimidating or time-consuming and allow users to deep dive into their interests quicker. Many apps can benefit from a card-based interface that shows users enough necessary information to make a quick maybe/no decision [177]. Cards serve as entry points to more detailed information. According to Carrie Cousins, cards can contain multiple elements within a design,



	RECOMMENDATION MODE			USER PROFILING		USER EXPERIENCE		
	RECOMM. APPROACH	MIN # BOOKS FOR RECCOM.	FEEDBACK MODE	MANDATORY LOGIN	INFO REQUIRED	BOOK DESCRIPTION	WEB	MOBILE (OPTIMIZED)
goodreads	collaborative filtering	20	rating	✓	user data, favourite genres	✓	✓	✓
WHAT SHOULD I READ NEXT?	collaborative filtering	1	✗	✗	book liked	✗	✓	✓
LibraryThing What's on your bookshelf?	collaborative filtering	10	rating	✓	book ratings & liked	✗	✓	✗
whichbook	content-based filtering	0	✗	✗	book tags	✓	✓	✗
BookBub	human recommendation	0	like	✓	user data, favourite genres, favorite authors	✓	✓	✓
YOURNEXTREAD.COM ...discover and share	collaborative filtering	1	like & dislike	✗	book ratings & liked	✓	✓	✗
Readgeek BETA	collaborative filtering	2	rating	✓	book ratings & liked	✓	✓	✓
TINDERBOOK	Hybrid filtering	1	like & dislike	✗	book liked	✓	✓	✓

Fig. 3.14 Comparison of existing book recommender systems.

but each should focus on only one bit of information or content [178]. A famous example of card-based GUI is that of the dating application Tinder, and according to Babich: "Tinder is a great example of how utilizing discovery mechanism to present the next option has driven the app to emerge as one of the most popular mobile apps. This card-swiping mechanism is curiously addictive, because every single swipe is gathering information - cards connect with users and offer the best possible options based on the made decisions." [174].

Finally, Tinderbook leverages DBpedia [12] and this allows to leverage a wealth of multi-language data, such as book descriptions, without the cost of creating and maintaining a proprietary database.

3.6 Summary

In this chapter, we have discussed the work relative to knowledge graph embeddings for recommender systems, part of it has been published in [3, 27, 26, 25].

We have introduced a set of crucial definitions, such as knowledge graph and item recommendation. Then, we have described how translational models, node2vec and entity2rec work in generating recommendations through knowledge graph embeddings. We have described a common experimental setup, composed of three datasets, a specific evaluation protocol and a set of metrics to evaluate the quality of the recommendations. We have then compared these systems among each other and with a set of state-of-the-art collaborative filtering systems. entity2rec tend to outperform competing systems, especially when the dataset is strongly sparse and has a low popularity bias. This is when the recommendation problem becomes interesting, because heuristics based on popularity fall short and traditional collaborative filtering algorithms have poor results. We have shown that entity2rec is based on a simple recommendation model, defined by the average of a set of property-specific relatedness score. For this reason, it can be easily intepreted and/or configured for a specific recommendation need, like in a multi-criteria recommender system. Finally, we have introduced Tinderbook, a book recommender system that provides book recommendations given a single book that the user likes. Tinderbook shows how entity2rec works with new users, and highlights the value of using semantic technologies in building recommender systems in an applied scenario.

Chapter 4

STEM: Stacked Threshold-based Entity Matching

In the last decade, we have witnessed to the generation of several knowledge graphs that grant access to an enormous amount of structured data and knowledge. However, the generation of knowledge graphs has required a tremendous manual effort to overcome several challenges. One of the typical issues in the generation of knowledge graphs that integrate data from a collection of heterogeneous sources is that of automatically detecting duplicate records. Entity matching (also known as instance matching, data reconciliation or record linkage) is the process of finding non-identical records that refer to the same real-world entity among a collection of data sources (see Sec. 2.1.2). Entity matching allows to identify redundant data, remove them (*deduplication*) and obtain unambiguous entities. Entity matching is rendered troublesome by the different data models used by the data providers, by possible misspellings, errors and omissions in data descriptions, by the use of synonyms, as well as the presence of implicit semantics in the textual descriptions. Entity matching systems typically define a metric to measure a similarity between entities. This metric can be defined through knowledge of the domain and a trial-and-error process, in a top-down manner [46, 47], or can be learned from annotated examples, in a bottom-up way [63, 65]. Then, the similarity is turned into a confidence score, which represents the degree of confidence in asserting that the pair of entities is a match. Finally, a threshold has to be specified, in order to convert the confidence score into a decision, namely classifying the pair as a match or not. This decision threshold introduces a trade-off between the precision, i.e. the capacity of discriminating false positives, and the recall, i.e. the capacity of individuating true positives, of the algorithm. Indeed, higher values of the threshold lead to a more selective classifier, which tends to incur in false negatives, reducing the recall of the algorithm, while lower values of the threshold produce the opposite effect. Thus, the user typically attempts to find a balance between these two measures, either manually or using more sophisticated approaches that are able to learn a configuration from annotated examples. Independently from the strategy chosen to set the final threshold, state-of-the-art systems typically rely on a single decision threshold. In this thesis, we show that the combination of the predictions of an array of thresholds using ensemble learning is able to break the trade-off between the precision and the recall of the algorithm, increasing both at the same time, and consequently the F-score of the algorithm. The first experiment we have performed used simple ensemble techniques such as majority and union voting, and already yielded good results in the context of the Financial Entity Identification and Information Integration (FEIII) Challenge. challenge [20]. The natural evolution of the work was to use more sophisticated techniques such as stacking, to boost the performance of the entity matching ensemble. We propose a general approach called STEM (Stacked Threshold-based Entity Matching), which can be applied on top of any numerical threshold-based entity matching system. STEM is based on the principle of *Stacking* (or Stacked Generalization) [21], which consists in training a meta-learner to combine the predictions of a number of base classifiers.

In this chapter, we address the following research question:

RQ2 *Can ensemble learning algorithms such as stacked generalization improve the performance of threshold-based classifiers in the entity matching process?*

To address this question, we introduce STEM (Stacked Threshold-based Entity Matching). Within RQ2, we formulate three sub-research questions:

RQ2.1 *Does STEM improves the F-score of threshold-based classifiers in a significant and consistent way?*

RQ2.2 *How does STEM performs when little training data is available?*

RQ2.3 *How can STEM be applied in the process of building a knowledge graph containing Points of Interests (POI) and events for tourists?*

The structure of the chapter is the following. In Sec. 4.1, we introduce a set of definitions for the problem, in Sec. 4.2 we describe the approach of STEM, in Sec. 4.3 we describe the experimental setup, in Sec. 4.4 we report the results obtained by STEM, and finally in Sec. 4.5 we describe the use of STEM in the construction of the 3cixty knowledge graph.
Part of the work discussed in this chapter has been published in the proceedings of the Data Science for Macro-Modeling (DSMM'16) workshop [20], in the Semantic Web journal [28] and in the Journal of Web Semantics [29].

4.1 Definitions

The problem of entity matching can be defined as follows [179]:

Definition 15 *Given two datasets A and B, find the subset of all pairs of entities for which a relation* $\sim$ *holds:* $M = \{e_1 \in A, e_2 \in B, (e_1, e_2) \in AxB : e_1 \sim e_2\}$

In this formulation, we assume that the schema mapping problem is solved, and thus:

Definition 16 *Given a property* π *of the schema of A and a property* ρ *of the schema of B, we assume that a set of* ***mapped properties*** *has been defined* $m_i = \{(\pi_i, \rho_i)\}$ *with* $i = 1..K$*. In the following, when we refer to the properties* $i = 1..K$*, we refer to the components of the mapping* m_i*, i.e. to* π_i *for* $e_1 \in A$ *and* ρ_i *for* $e_2 \in B$*.*

We assume that the comparison between a pair of entities e_1 and e_2 is carried out on a set of **literal values** v_i of properties $i = 1..K$. We assume that the entity matching system is a pairwise numeric threshold-based binary classifier acting on the properties $i = 1..K$. Thus:

Definition 17 *We define the* ***linkage rule*** *as a Boolean function* $\hat{f}:(e_1,e_2)\in AxB\rightarrow\{0,1\}$*, where* $\hat{f}=1$ *indicates that the pair is deemed a match and* $\hat{f}=0$ *indicates that the pair is not deemed to be a match.*

Definition 18 *The comparison of two entities is performed using a* ***comparison vector*** $s(e_1,e_2)=\{s_1(e_1,e_2),s_2(e_1,e_2)..s_K(e_1,e_2)\}$*, where the components* $s_i\in\mathbb{R}^+$ *represent atomic similarities defined over a number of literal values* $s_i(e_1,e_2)=s(v_i(e_1),v_i(e_2))$ *of the properties* $i=1..K$.

Definition 19 *The comparison vector is then turned into a* ***confidence vector*** $c(e_1,e_2)=\{c_1(s_1),c_2(s_2)..c_K(s_K)\}$*, where each component* c_i *represents the degree of confidence in stating that the pair of entities is a match given by the similarity score* $s_i=s_i(e_1,e_2)$.

Definition 20 *We define a* ***confidence function*** $f:c(e_1,e_2)\rightarrow[0,1]$ *which maps the confidence vector* $c(e_1,e_2)$ *into a final score representing the confidence of the entity matching system to state that the pair of entities is a match.*

Definition 21 *We define as* ***threshold-based classifier*** *a linkage rule* $\hat{f}$ *that depends on the confidence function* f *in the following way:*

$$\hat{f}(e_1,e_2;t)=\theta(f(c(e_1,e_2))-t) \tag{4.1}$$

where θ is the Heaviside step function and $t\in[0,1]$ is a given threshold.[1] The linkage rule of a threshold-based classifier has a very intuitive interpretation. A pair of entities is considered to be a match if the degree of confidence f that the pair is a match and f is above a certain threshold t. The threshold t can be defined experimentally or can be learned from a set of examples. Independently from the strategy through which it is set, the threshold t introduces a trade-off between the

[1] We assume that both f and t are normalized in the [0,1] interval, but it is intuitive to see that the same argument holds for any closed interval $[a,b]\in\mathbb{R}$ with $a<b$, as Eq. 4.1 is invariant to any multiplying factor within the θ function.

rate of false positives and false negatives that the algorithm will accept. To see why this is the case, let us first start from the definition of the two types of errors defined in [179], considering the variations of the confidence value $f \in [0,1]$. The first error, which corresponds to the probability of having false positives, occurs when an unmatched comparison is considered as a match:

$$p_{fp} = P(\hat{f} = 1 | e_1 \neq e_2) = \int_0^1 P(\hat{f} = 1 | f) P(f | e_1 \neq e_2) df \tag{4.2}$$

Given that we consider a binary threshold-based classifier, it follows from Eq. 4.1 that:

$$P(\hat{f} = 1 | f) = \theta(f - t) \tag{4.3}$$

which leads to:

$$p_{fp} = P(\hat{f} = 1 | e_1 \neq e_2) = \int_t^1 P(f | e_1 \neq e_2) df \tag{4.4}$$

The second error, which corresponds to the probability of false negatives, occurs when a matched comparison is not considered to be a match:

$$p_{fn} = P(\hat{f} = 0 | e_1 = e_2) = \int_0^1 P(\hat{f} = 0 | f) P(f | e_1 = e_2) df \tag{4.5}$$

From Eq. 4.3, it follows that:

$$P(\hat{f} = 0 | f) = 1 - \theta(f - t) \tag{4.6}$$

and thus:

$$p_{fn} = P(\hat{f} = 0 | e_1 = e_2) = \int_0^t P(f | e_1 = e_2) df \tag{4.7}$$

The probability of true positives is similarly measured as:

$$p_{tp} = P(\hat{f} = 1 | e_1 = e_2) = \int_t^1 P(f | e_1 = e_2) df \tag{4.8}$$

Let us now consider the graphic illustration provided in Fig. 4.1. Assuming that f is a meaningful confidence function, the probability density function of the values of f under the condition that $e_1 = e_2$, i.e. $P(f | e_1 = e_2)$, has a higher mean and is located to the right, while $P(f|e_1 \neq e_2)$, conditioned by $e_1 \neq e_2$, is located to the left. Let also N_+ be the total number of positive pairs, i.e. matching entities, and N_- the total number of negative pairs, i.e. non matching entities, in the data. In this graphical representation, the linkage rule of Eq. 4.1 implies that the area of $P(f|e_1 = e_2)$ situated to the left of the vertical line (in yellow) corresponds to the probability of classifying a true match as a non matching pair, i.e. to the probability of producing false negatives p_{fn}. The number of false negatives FN is then given by $FN = p_{fn}N_+$. On the other hand, the area of $P(f|e_1 \neq e_2)$ situated to the right of the vertical line (in orange) corresponds to the probability of classifying a false match as a match, i.e. the probability of producing false positives p_{fp}. Similarly to the previous case, we have that $FP = N_- p_{fp}$. Finally, we also have that the grey area in the graph is the probability of true positives p_{tp}. The number of true positives is then given by: $TP = N_+ p_{tp}$. From Fig. 4.1 we can see that p_{fn}, and consequently FN, is increasing when the threshold t increases, and at the same time p_{fp}, and consequently FP, is decreasing when the threshold t increases. p_{tp} is also decreasing,

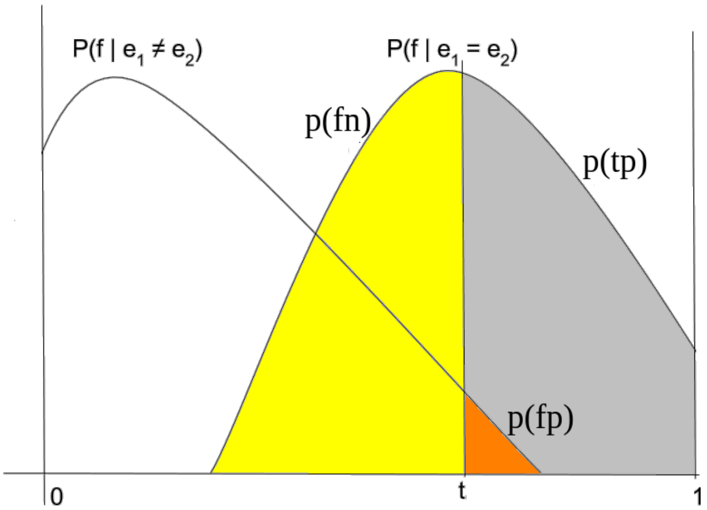


Fig. 4.1 Graphical depiction of p_{fn}, p_{fp} and p_{tp} under the linkage rule Eq. 4.1. The vertical line represents the decision threshold t. The shape of the probability distribution has illustrative purposes.

but at a slower pace. Now, if we recall the definition of precision and recall [163]:

$$p = \frac{TP}{TP+FP} \tag{4.9}$$

$$r = \frac{TP}{TP+FN} \tag{4.10}$$

we can see that, when t increases, $FP \to 0$ faster than TP, and p increases. At the same time, FN is growing and r decreases. Conversely, when t decreases FP grows and FN decreases, increasing r and decreasing p. Thus, the threshold t introduces a trade-off between the precision and the recall of the algorithm (we provide experimental evidence of this heuristic argument in Sec. 5.5.6). Note that this trade-off is not limited to Entity Matching and is well known by the Information Retrieval and Statistical Learning community, where precision-recall curves obtained through variations of the decision threshold are often used as a measure of an overall algorithm's efficiency [180, 163, 181].

4.2 The STEM approach

In this work, we show that stacking can break the trade-off by raising both precision and recall at the same time through supervised learning. Stacking [21] (also known as stacked generalization), is based on the idea of creating an ensemble of base classifiers and then combining them by means of a supervised learner, which is trained on a set of labeled examples. In this case, the base classifiers correspond to a single threshold-based classifier $\hat{f}(e_1,e_2;t)$ with a set of different decision thresholds $t_1,t_2..,t_N$. The supervised learner is a binary classifier whose features are the match decisions of the base classifiers $F:\{\hat{f}(e_1,e_2;t_1),\hat{f}(e_1,e_2;t_2)...\hat{f}(e_1,e_2;t_N)\} \to \{0,1\}$ and whose output is a binary match decision, which represents the final decision of the system. Stacking requires the creation of a gold standard G containing correctly annotated entity pairs, which are used as a training set for the supervised learning approach. More in detail, the Stacking Threshold-based Entity Matching approach (Fig. 4.2) works as follows:

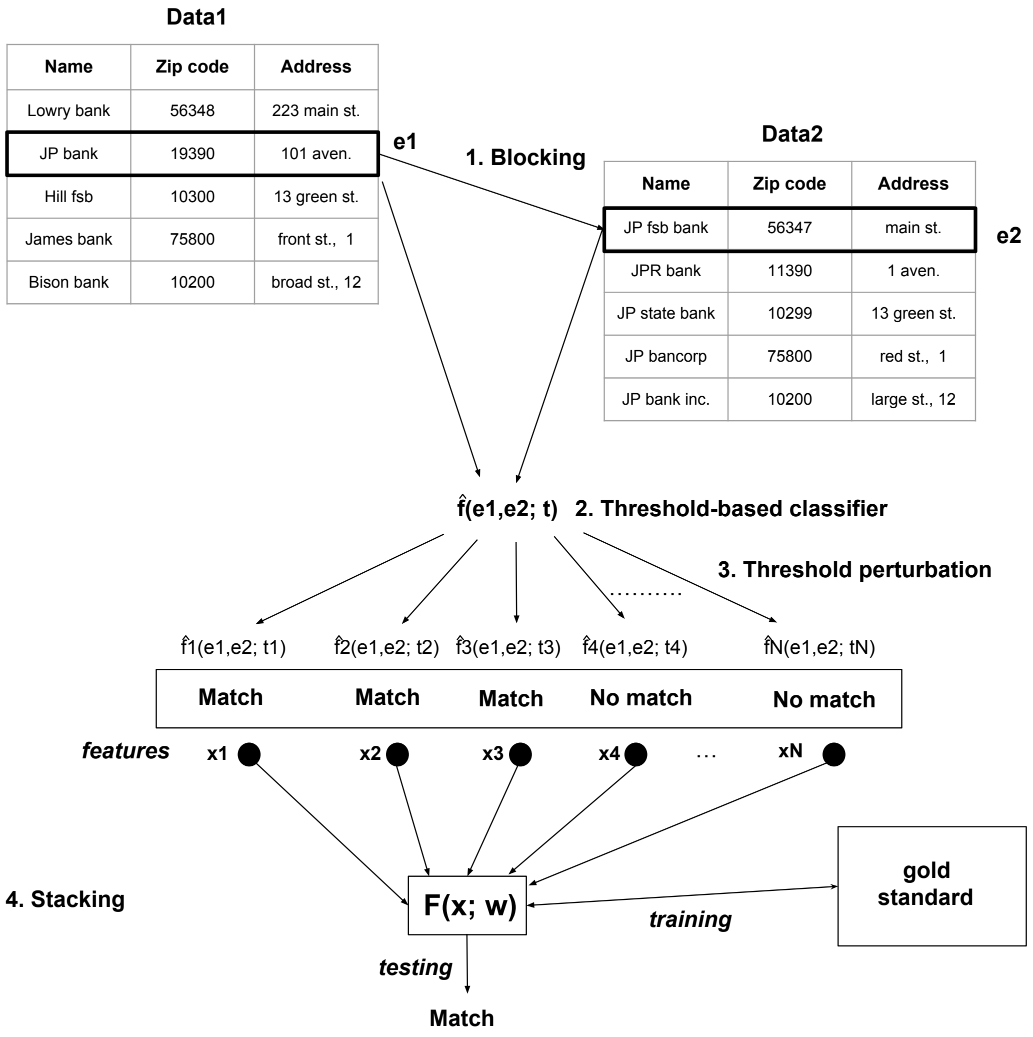
1. **Blocking** (optional): although not strictly needed, it is in practice necessary for large datasets to find good candidates e_1, e_2 using a blocking strategy, avoiding a quadratic comparison of entities (see Section 4.3.3 and Section 4.3.4).

2. **Threshold-based classifier**: start from a linkage rule based on a threshold-based classifier $\hat{f}(e_1, e_2; t)$ such as the one defined in Eq. 4.1

3. **Threshold perturbation**: generate an ensemble of N linkage rules $\hat{f}(e_1, e_2; t_i)$ where t_i are linearly spaced values in the interval $[t - \frac{a}{2}, t + \frac{a}{2}]$ and $0 < \frac{a}{2} < min\{t, 1-t\}$ is the perturbation amplitude

4. **Stacking**: combines the predictions corresponding to different thresholds using supervised learning.
   Features: use the predictions $x_i = \hat{f}(e_1, e_2; t_i)$ as features for a supervised learner $F(x; w)$ where w are parameters that are determined by the learning algorithm.
   Training: train the supervised learner $F(x; w)$ on the gold standard G, determining the parameters $\hat{w}$. This typically involves the minimization of an error function $E(x, w, G)$:
   $$\hat{w} = \min_w E(x, w, G) \tag{4.11}$$
   The shape of the error function $E(x, w, G)$ and how the optimization is solved depends on the particular supervised learner that is chosen. We use an SVM classifier and we thus rely on the SVM training algorithm [56]. Note that the training process only needs to be done once per dataset, as the learned model can be stored and loaded for testing.
   Testing: generate the final prediction $F(x; \hat{w})$.

The intuition behind this approach is that using stacking the space of features is no longer the confidence score f as for the threshold-based classifier. The supervised learner F uses as features the set of matching decisions of the base classifiers and, by combining them in a supervised way, it is no longer tied to the trade-off introduced by the threshold t that we have described in Sec. 4.1. We show experimental evidence of the effectiveness of stacking in increasing both the precision and the recall of a



Data1

Name	Zip code	Address
Lowry bank	56348	223 main st.
JP bank	19390	101 aven.
Hill fsb	10300	13 green st.
James bank	75800	front st., 1
Bison bank	10200	broad st., 12

Data2

Name	Zip code	Address
JP fsb bank	56347	main st.
JPR bank	11390	1 aven.
JP state bank	10299	13 green st.
JP bancorp	75800	red st., 1
JP bank inc.	10200	large st., 12

Fig. 4.2 Global architecture of the STEM framework.

threshold-based classifier in Sec. 4.4.

We now provide the descriptions of the two threshold-based entity matching systems that are used in this work, showing that they both constitute particular cases of Eq. 4.1.

4.2.1 Linear Classifier

One of the simplest models for the confidence function $f(e_1, e_2)$ of a pair of entities e_1 and e_2 is that obtained by the linear combination of the components of the confidence vector $c(e_1, e_2)$ introduced in Sec. 4.1. Given the set of properties $j = 1..K$ and their respective values $v_j(e_1)$ and $v_j(e_2)$ for both entities, property-wise similarities are functions that yield a vector of similarity scores $s_j = s_j(v_j(e_1), v_j(e_2))$, where typically $s_j \in [0,1]$ with $s_j = 1 \iff v_j(e_1) \equiv v_j(e_2)$. At this point, similarity scores s_i are normally turned into the property-wise confidence scores $c_i = c_i(s_i)$, which are then linearly combined through the confidence function f. This is the case of Silk[2] [62], which is a popular Link Discovery framework, specifically built to generate RDF links between data items within different Linked Data resources. More specifically, Silk works with distances d_i rather than with similarities s_i and different comparators can be selected to define the distances d_i, such as Levehnstein, Jaro-Winkler, exact comparators, Jaccard [182]. Then, distance scores $d_i > 0$ are turned into confidence scores c_i according to the rule[3] (Fig. 4.3):

$$c_i = c(d_i) = \begin{cases} -\frac{d_i}{\tau_i} + 1 & 0 \leq d_i < 2\tau_i \\ -1 & d_i \geq 2\tau_i \end{cases}$$

where τ_i are property-specific thresholds. Note that c_i is a monotone decreasing function, as it depends on distances d_i rather than on similarities s_i values. In this way, for each property used for the comparison, a confidence score $c_i \in [-1, 1]$ is obtained. Silk allows to combine these confidence scores in multiple ways, among

[2] http://silkframework.org

[3] https://bit.ly/2OWCnsR

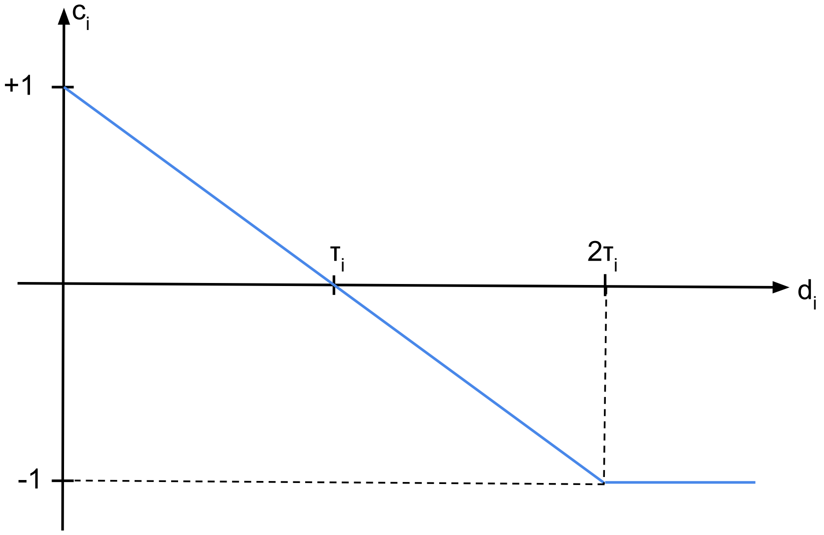


Fig. 4.3 Silk function to compute property-wise confidence scores from distance values

which the linear combination, which is the one that has been utilized in this work:

$$\hat{f} = \theta(\sum_{i=1}^{K} w_i c_i - t) \tag{4.12}$$

which corresponds to the linkage rule Eq. 4.1 with:

$$f(c(e_1, e_2)) = \sum_{i=1}^{K} w_i c_i \tag{4.13}$$

The final decision threshold t corresponds to the parameter ‘minConfidence’ in Silk configuration file. This parameter, together with all the others such as property-wise thresholds or comparators, can be manually set through a trial-and-error process or they can be learnt through an active learning algorithm that is based on the approach of letting users annotate matches that produce the utmost information gain [183].

4.2.2 Naive Bayes Classifier

Naive Bayes is a term used to describe a family of classifiers that are based on the Bayes theorem and on a particular assumption of independence among the compo-

nents of the evidence vector [184, 185]. We use the formulation of the Naive Bayes classifier that has been popularized by Paul Graham's Bayesian spam filter.[4] We now want to show that the Naive Bayes classifier can be considered as a threshold-based classifier, obeying to the decision rule of Eq. 4.1.
Given a set of classes X_i with $i = 1..N$ and a vector of observations $x = \{x_1, x_2..x_K\}$, the Naive Bayes classifier aims to estimate the probability of a class given a set of observed data $P(X_i|x)$ by applying the Bayes theorem and the conditional independence condition (from this assumption comes the adjective 'Naive'):

$$P(X_i|x) = \frac{P(x|X_i)P(X_i)}{P(x)} = \frac{P(X_i)\prod_{j=1}^{k} P(x_j|X_i)}{P(x)} \tag{4.14}$$

In our case, we have a binary classification problem, where the decisions $X_1 = 1 =$ 'Match' and $X_2 = 0 =$ 'No Match' and the observations are represented by the comparison vector s. Eq. 4.14 for $X_1 = 1$ becomes:

$$P(1|s) = \frac{P(1)\prod_{i=1}^{k} P(s_i|1)}{P(s)}$$

Since $P(s) = P(s|1)P(1) + P(s|0)P(0)$ the denominator can be rewritten as:

$$P(1|s) = \frac{P(1)\prod_{i=1}^{k} P(s_i|1)}{P(s|1)P(1) + P(s|0)P(0)} \tag{4.15}$$

and then, using again the conditional independence hypothesis, factorized as:

$$P(1|s) = \frac{P(1)\prod_{i=1}^{k} P(s_i|1)}{P(1)\prod_{i=1}^{k} P(s_i|1) + P(0)\prod_{i=1}^{k} P(s_i|0)} \tag{4.16}$$

Now, by applying Bayes theorem $P(s_i|1) = \frac{P(1|s_i)}{P(1)}P(s_i)$ and $P(s_i|0) = \frac{P(0|s_i)}{P(0)}P(s_i)$, denoting with $x = P(1)$ and $1 - x = P(0)$, we have:

$$P(1|s) = \frac{\frac{1}{x^{k-1}}\prod_{i=1}^{k} P(1|s_i)}{\frac{1}{x^{k-1}}\prod_{i=1}^{k} P(1|s_i) + \frac{1}{(1-x)^{k-1}}\prod_{i=1}^{k} P(0|s_i)} \tag{4.17}$$

[4] http://www.paulgraham.com/spam.html

Finally, assuming that, *a priori*, $P(1) = P(0)$ and thus $x = 1-x$, we can remove the coefficients and by denoting with $c_i = P(1|s_i)$ we obtain:

$$P(1|s) = \frac{c_1 c_2 .. c_k}{c_1 c_2 .. c_k + (1-c_1)(1-c_2)..(1-c_k)} \tag{4.18}$$

Details of the derivation can be found in [186]. Note that $c_i = P(1|s_i)$ exactly represents the confidence score derived from the similarity value s_i. It is also important to notice that this simplifying assumption $P(1) = P(0)$, although widespread in practice and used in the implementation of Duke, is not necessary for a Naive Bayes classifier and can be modified with any other strategy to estimate prior probabilities.

At this point, it is necessary to specify a decision rule, that is a rule to turn the probability evaluation into a decision. A common approach is the Maximum a Posteriori (MAP) Estimation [187], namely selecting the class that maximizes the posterior probability:

$$X = \arg\max_{X_i} P(X_i|x) \tag{4.19}$$

which allows to define a binary linkage rule as:

$$\hat{f} = 1 \iff P(1|s) > P(0|s) \tag{4.20}$$

which can easily be rewritten as:

$$\hat{f} = 1 \iff \frac{P(1|s)}{P(0|s)} > 1 \tag{4.21}$$

Now, by adopting a decision-theoretic notion of cost, we can turn Eq. 4.21 into [188]:

$$\hat{f} = 1 \iff \frac{P(1|s)}{P(0|s)} > \lambda \tag{4.22}$$

where λ is a value that indicates how many times false positives are more *costly* than false negatives. From Eq. 4.22, it is clear that if $\lambda > 1$, we require that $P(1|s)$ is λ times greater than $P(0|s)$ in order to consider the pair to be a match, and thus we are more keen to accept false negatives than false positives. Vice versa, if $\lambda < 1$, the algorithm will tend to have more false positives than false negatives. Finally, by considering that $P(0|s) = 1 - P(1|s)$ and by using Eq. 4.18 we obtain the decision rule:

$$\hat{f} = 1 \iff \frac{c_1 c_2..c_k}{c_1 c_2..c_k + (1-c_1)(1-c_2)..(1-c_k)} > t \tag{4.23}$$

where $t = \frac{\lambda}{1+\lambda}$. It is now easy to see that Eq. 4.23 can be rewritten as:

$$\hat{f} = \theta(\frac{c_1 c_2..c_k}{c_1 c_2..c_k + (1-c_1)(1-c_2)..(1-c_k)} - t) \tag{4.24}$$

which has the same form of Eq. 4.1, where the combination of confidence scores c_i has the role of the global 'confidence function':

$$f(c(e_1,e_2)) = \frac{c_1 c_2..c_k}{c_1 c_2..c_k + (1-c_1)(1-c_2)..(1-c_k)} \tag{4.25}$$

We have thus shown that from a Naive Bayes classifier we can obtain a threshold-based classifier abiding by Eq. 4.1. As we have argued in Sec. 4.1, the threshold t rules the trade-off between the rate of false positives and false negatives that the algorithm will accept. This is evident by its relation with λ:

$$\lambda \to \infty \Rightarrow t \to 1 \tag{4.26}$$

$$\lambda \to 0 \Rightarrow t \to 0 \tag{4.27}$$

Thus, the higher the value of t, the higher needs to be the probability that the pair is a match for the algorithm to consider it a match. Thus, we are less likely to have false positives and more likely to have false negatives.

In the past years, Naive Bayes classifiers have been utilized in a large number of fields, such as spam filtering [189], document and text classification [190], information retrieval [184], entity matching [191] and so on. Duke[5] is a popular open-source deduplication engine, which implements Naive Bayes classification. Duke is a flexible tool, which accepts different formats of input data, and is easy to configure through a simple XML file. For each field of each data source, the user can choose a number of string cleaners, such as functions that remove abbreviations or normalize lower/upper cases. For each property, Duke allows to select a comparator

[5] https://github.com/larsga/Duke

among popular string similarity measures such as Levensthein, Jaro-Winkler, exact comparators and so on [182]. The comparators thus compute, for each property, a normalized similarity score s_i. Then, in order to turn similarity scores into a confidence score c_i, Duke uses the heuristic function:

$$c_i = P(1|s_i) = \begin{cases} low_i & s_i \leq 0.5 \\ (high_i - 0.5)s_i^2 + 0.5 & s_i \geq 0.5 \end{cases}$$

where low_i and $high_i$ are parameters that the user can configure for each property. The rationale behind this formula of $P(1|s_i)$ is that $P(1|s_i = 0) = low$ and $P(1|s_i = 1) = high$, and, as Duke's users were finding the algorithm to be too strict, a quadratic instead of a linear trend has been chosen when s_i is larger than 0.5. After that c_i is computed for each property, the overall $P(1|s)$ is calculated through Eq. 4.18 and the decision is taken through Eq. 4.23. Similarly to the case of Silk, the final decision threshold t is a parameter that can be configured in a XML file. Duke also includes a genetic algorithm that automatizes the configuration process and in general represents a valid alternative to the manual configuration. Through an active learning approach, Duke asks to the user in an interactive way if a pair of entities should be a match or not, selecting the most informative pairs, i.e. the ones with utmost disagreement among the population of configurations [65].

4.2.3 Computational complexity

A crucial point for entity matching systems, which are often used to find matching entities among datasets with large numbers of instances, is their computational complexity. The stacking approach introduced by STEM adds an overhead to the runtime performance of the base classifier due to the generation of the ensemble of predictions and to the learning process. More in detail, the computational complexity of STEM in the current sequential implementation can be roughly approximated with:

$$T_{STEM} \approx N * T_{baseclassifier}(n,m) + T_{stacking}(N,g,k)$$

where N is the number of features, n is the number of instances of the first dataset, m is the number of instances of the second dataset, g is the size of the training set and k is the size of the test set. $T_{baseclassifier}(n,m)$ depends on the nature and the configuration of the base classifier, especially on the blocking strategy. Let us assume that the base classifier adopts a smart blocking strategy such as that of Silk, we can assume that $T_{baseclassifier}(n,m) = O(n+m)$ [192]. $T_{stacking}(N,g,k)$ depends on the supervised learner that is used and depends on the training and the testing time $T_{stacking}(N,g,k) = T_{train}(N,g) + T_{test}(N,k)$. The training and testing processes can easily be decoupled, for instance saving the trained model to a file and loading it subsequently for testing. However, in this section, we discuss the worst case in which we need to first train and then test the model. In this work, we use a kernel SVM[6] classifier as a supervised learner, whose complexity may vary depending on a number of practical factors depending on the specific implementation. However, as a rule of thumb, it is reasonable to assume [193, 194] that: $O(N*g^2) < T_{train}(N,g) < O(N*g^3)$ and $T_{test}(N,k) < O(N*k)$. To summarize, we can then say that:

$$T_{STEM}^{train} < N * O(n+m) + O(N*g^3) \tag{4.28}$$

$$T_{STEM}^{test} < N * O(n+m) + O(N*k) \tag{4.29}$$

In our experiments, we have that $g \leq n$, $g \leq m$ and, by using cross-validation, we have that $k \approx g$. In practice, we observe that when the number of features N grows, the time for generating the predictions quickly surpasses the time of stacking, i.e. $N * T_{baseclassifier}(n,m) > T_{stacking}(N,g,k)$ (see Section 5.5.6 for an example). Note that the time could be reduced to $T_{STEM} \approx T_{baseclassifier}(n,m) + T_{stacking}(N,g)$ by parallelizing the generation of the predictions of the N base classifiers, which we leave as a future work.

4.3 Experimental setup

As we have explained in Sec. 4.2, the STEM approach is general and can be utilized on top of any threshold-based entity matching system. In this work, we have implemented it and evaluated through two different open source frameworks, Duke and Silk, which are based respectively on a Naive Bayes and on a linear classifier.

[6] http://scikit-learn.org/stable/modules/svm.html

In Sec. 4.3.3 and in Sec. 4.3.4, we describe the configuration process of these frameworks inside STEM. The software implementation of STEM, the configuration files and the data used for the experiments are publicly available on github.[7]

4.3.1 Datasets

The first dataset utilized for the evaluation of the proposed approach is that released by the organizers of the Financial Entity Identification and Information Integration challenge of 2016 (FEIII2016).[8] The purpose of the challenge is that of creating a reference financial-entity identifier knowledge graph linking heterogeneous collections of entity identifiers. Three datasets have been released:

- FFIEC: from the Federal Financial Institution Examination Council, provides information about banks and other financial institutions that are regulated by agencies affiliated with the Council.
- LEI: contains Legal Entity Identifiers (LEI) for a wide range of institutions.
- SEC: from the Securities and Exchange Commission and contains entity information for entities registered with the SEC.

In this work, we focus on the Entity Matching of entities of the FFIEC database and the SEC database, as it proved to be the most challenging one. The gold standard, which can be seen as a benchmark for the evaluation of the systems as well as a set of annotations to create a supervised system, has been created by a panel of experts of the field. The gold standard contains 1428 entity pairs, with 496 positive and 932 negative examples. The dataset is available online.[9]

A second evaluation of the STEM approach is performed on the dataset released by the DOREMUS project[10] in the context of the instance matching track of the Ontology Alignment Evaluation Initiative 2016 (OAEI2016[11]). The Instance Matching

[7] https://github.com/enricopal/STEM
[8] https://ir.nist.gov/dsfin/index.html
[9] https://ir.nist.gov/dsfin/data/feiii-data-2016-final.zip
[10] http://www.doremus.org/
[11] http://islab.di.unimi.it/im_oaei_2016/

Track of the OAEI 2016 aims at evaluating the efficiency of matching tools when the goal is to detect the degree of similarity between pairs of items/instances expressed in the form of OWL Aboxes. The DOREMUS datasets contain real world data coming from two major French cultural institutions: the French National Library (BnF) and the Philharmonie de Paris (PP). The data is about classical music works and is described by a number of properties such as the name of the composer, the title(s) of the work, its genre, instruments and the like. We focused our evaluation on two tasks, whose description we report here:

- **DOREMUS 9-heterogeneities**: "This task consists in aligning two small datasets, BnF-1 and PP-1, containing about 40 instances each, by discovering 1:1 equivalence relations between them. There are 9 types of heterogeneities that data manifest, that have been identified by the music library experts, such as multilingualism, differences in catalogs, differences in spelling, different degrees of description."
- **DOREMUS 4-heterogeneities**: "This task consists in aligning two bigger datasets, BnF-2 and PP-2, containing about 200 instances each, by discovering 1:1 equivalence relations between the instances that they contain. There are 4 types of heterogeneities that these data manifest, that we have selected from the nine in the Nine hererogeneities task and that appear to be the most problematic: 1) Orthographical differences, 2) Multilingual titles, 3) Missing properties, 4) Missing titles."

Data is accessible online.[12] To the reference links provided by the organizers, we add 20 and 123 false links respectively for the DOREMUS 9-heterogeneities and the DOREMUS 4-heterogeneities gold standards, to enable the supervised learning approach implemented by STEM that necessitates both positive and negative entity pairs. No mapping among properties is necessary for this dataset, as the schema is already aligned.

An additional dataset used in the experimentation of the STEM approach is derived from the 3cixty Nice knowledge graph. This knowledge graph contains

[12] http://islab.di.unimi.it/im_oaei_2016/data/Doremus.zip

Dataset	Domain	Provider	Number of entities	Format
FFIEC	Finance	Federal Financial Institution Council	6652	CSV
SEC	Finance	Security and Exchange Commission	129312	CSV
BnF-1	Music	French National Library	32	RDF
PP-1	Music	Philharmonie de Paris	32	RDF
BnF-2	Music	French National Library	202	RDF
PP-2	Music	Philharmonie de Paris	202	RDF
3cixty Nice places	Culture and Tourism	3cixty Nice knowledge graph	336,900	RDF

Table 4.1 Datasets summary

Dataset 1	Dataset 2	Gold standard pairs	Challenge	Task
FFIEC	SEC	790	FEIII2016	FFIEC-SEC
BnF-1	PP-1	52	OAEI2016	DOREMUS 9-heterogeneities
BnF-2	PP-2	325	OAEI2016	DOREMUS 4-heterogeneities
3cixty Nice places	3cixty Nice places	756	-	-

Table 4.2 Matching tasks summary

Nice cultural and tourist information (such as Place-type entities) and it is created with a multi datasource collection process, where numerous entities are represented in multiple sources leading to duplicates. This creates the need of matching and the resolution of the entities. Further details of the making of the knowledge graph with the selection of the gold standard is detailed in Sec. 4.5, while in this section we report the statistics of the gold standard that drove the entity matching task.

For the FEIII and the DOREMUS datasets, we assume that the Unique Name Assumption is true, meaning that two data of the same data source with distinct references refer to distinct real world entities. For 3cixty this is clearly not the case, as we are matching the dataset with itself to detect duplicates.

A summary of the datasets statistics is reported in Tab. 4.1 and of the matching tasks in Tab. 4.2.

4.3.2 Scoring

To evaluate the efficiency of the algorithm we have used the standard precision p, recall r and f measures [163]. These measures, if not specified otherwise, have been evaluated through a 4-fold cross validation score process. Given the ambiguity of the definition of p, r and f when performing cross validation [195], we hereby specify that we have used the average of their values over the four folds. The computation of the precision score depends on the *closed/open world assumption* (Sec. 2.1), i.e. whether we assume that all correct matches are annotated in the gold standard or not. In practice, it is normal to have in the gold standard only a fraction of all the real matching entities and we thus follow, by default, the open world assumption. In this case, when a pair of entity is considered to be a match by STEM and is not present in the gold standard, we are unable to determine whether this is due to a false positive or to a missing annotation and we simply ignore it in the scoring. In the case of the experiments with DOREMUS datasets, where the organizers of the challenge claim to have annotated all the true matches, we follow the closed world assumption. In this case, every match that is not annotated in the gold standard is considered as a false positive.

4.3.3 Duke

Entity format: Duke is able to handle different formats for input data, such as .csv (comma separated value) or .nt (n-triples). In the first case, an entity is represented by a record in a table. In the second case, an entity is a node in a knowledge graph.

Blocking method: we reduce the search space for the entity matching process from the space of all possible pairs of entities *AxB* using an inverted index, in which property values are the indexes and the tuples are the documents referred by the indexes. The lookup of a tuple given a value has, therefore, a unitary cost. We reduce the search space to a small subset of the most likely matching entity pairs that satisfy a given Damerau-Levenshtein distance [196] for each value pair of the tuples, and we considered the first m candidates.[13]

[13] We empirically set the distance to 2 and the number of potentially retrievable candidates to 1,000,000 (conservative boundary).

Configuration: the first step of the implementation consists in configuring Duke. Duke is built by default on top of a Lucene Database,[14] which indexes the records through an inverted index and does full-text queries to find candidates, implementing the blocking strategy. The Lucene Database can be configured in Duke by setting a number of parameters such as the **max-search-hits**, that is the maximum number of candidate records to return or **min-relevance**, namely a threshold for Lucene's relevance ranking under which candidates are not considered. Duke then allows to select a number of properties to be taken into account to establish if a pair of entities match, such as name, address, zip code. Duke requires to specify a mapping between the fields of the data sources and those on which the comparison has to be performed, e.g. "LegalName → NAME, LegalEntityAddress → ADDRESS, LegalEntityCode → ZIPCODE". In this case, we have manually configured Duke during the participation to the FEIII2016 challenge and the choice of cleaners, comparators, as well as the detailed mapping among the properties is reported in [20].

4.3.4 Silk

Entity format: Silk is specifically built to deal with RDF formats, such as .ttl (turtle) or .nt (n-triples), where entities are represented as nodes in a Knowledge Graph. However, it allows to convert data from a variety of formats, such as .csv (comma separated values).

Blocking method: Silk implements a multidimensional blocking system, called MultiBlock [197], which is able to not lose recall performance. Differently from most blocking system that operates on one dimension, MultiBlock works by mapping entities into a multidimensional index, preserving the distances between entities.

Configuration: Silk can easily be configured through an XML file. To configure the blocking algorithm, it is sufficient to specify the number of blocks, which we have empirically set to 100. A set of properties $i = 1..K$ onto which the matching is

[14] https://lucene.apache.org

based needs to be specified and then, for each of them, the user can select among a large number of ‘transformators’ (comparable to Duke’s cleaners) to pre-process and normalize strings. The choice of transformators and comparators has been based on the result obtained with Duke in the participation to the FEIII challenge and a similar configuration file has been produced for Silk. A manual configuration to optimize the f score has been used also for the DOREMUS data in the context of the OAEI challenge [198].

4.3.5 Stacking

Differently from the previous steps, which are mainly based on low-level string similarity measures, the supervised learner can implicitly learn semantic similarities from the human annotations of the gold standard. The stacking process is implemented through a Python script that executes Duke or Silk a number N of times, editing the threshold t through uniform perturbations of amplitude a, automatically modifying Duke’s or Silk’s configuration file. Then, the script saves Duke’s or Silk’s outputs and turns them into a training set for a supervised learner with $id1, id2$ pairs on the rows and N features on the columns.
The user may choose different supervised learners for the stacking layer. What we have experimentally found to work better, given the small number of features, is an SVM with a RBF kernel [56]. In many cases, such as the default one, the learning algorithm leaves a number of parameters (so-called “hyper parameters”) to be determined. Let $F(x;\hat{w},\theta)$ be a supervised learner where θ is the vector of hyper parameters (C and γ in the case of SVM with RBF kernel). In order to optimize the efficiency of the algorithm with respect to these hyper parameters, we have trained the algorithm on an array of possible values of θ and selected $\hat{\theta}$ as the vector that optimized 4-fold cross validation score (grid search cross validation [199]).
For what concerns the number of features N, it is reasonable to expect that higher values tend to increase the efficiency of the algorithm up to a saturation point, where no further predicting power is added by an additional instance of the base classifier. Actually, we observe that increasing the number of features can also lead to efficiency decrease, as a typical overfitting problem. This saturation point

will typically depend on the amplitude a of perturbation, as with small intervals $-a/2, a/2$ we expect it to occur earlier. This will also depend on the size of the datasets and its complexity, so no one-fits-all solution has been individuated. The experiments of Sec. 4.4.1 with varying values of a and N show that $a = 0.25$ and $N = 5$ appears to be a good rule of thumb.

4.4 STEM Experimental Results

4.4.1 STEM vs threshold-based classifiers

In this section, we address the research question RQ2.1: *Does STEM improves the F-score of threshold-based classifiers in a significant and consistent way?*

We first provide evidence of the trade-off between precision and recall introduced by the decision threshold and then we show that STEM is able to increase the precision and the recall of the base classifiers at the same time. In the following, we refer to the STEM approach implemented on top of Duke as STEM-NB and to that implemented on top of Silk as STEM-LIN.

The premise of this work is that the threshold t in decision rule Eq. 4.1 introduces a trade-off between precision and recall. In Sec. 4.1 we have provided a heuristic argument of why this should be the case and now we provide experimental results. In Fig. 4.4, we report the precision and recall obtained by running Duke on the FFIEC-SEC dataset for a set of 20 equally spaced threshold values $t \in [0.05, 0.9]$. The graph clearly shows the trade-off between precision and recall of the algorithm ruled by the threshold t. The trend for both curves is non-linear, with moderate changes in the central part and sudden variations at the sides. The typical configuration process of a threshold-based classifier attempts to find a balance between the two metrics, in order to maximize the F-score of the algorithm. Then, using STEM, both metrics can be increased at the same time using stacking. In Tab. 4.3 and Tab. 4.4 we support this claim by reporting respectively the results obtained using STEM-NB and STEM-LIN on FFIEC-SEC, DOREMUS 4-heterogeneities, DOREMUS 9-heterogeneities tasks, varying the number of features N. The value of the perturbation amplitude a has

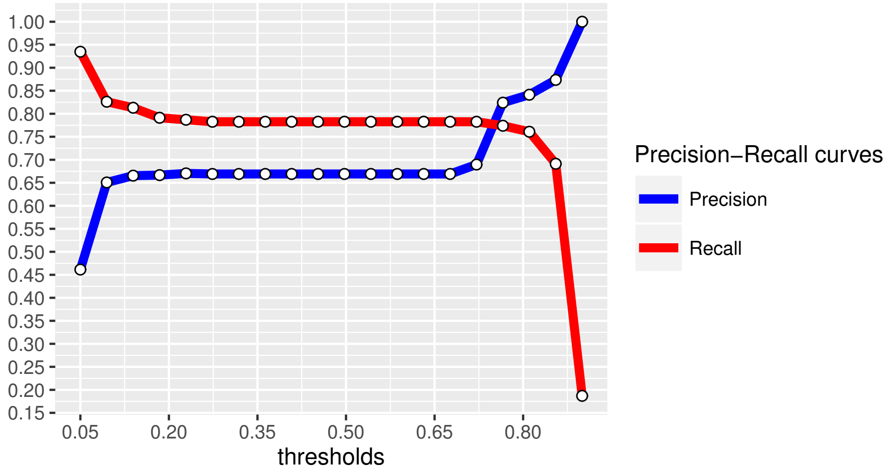


Fig. 4.4 Precision and recall curves as functions of the threshold t for Duke on the FEIII dataset. It clearly shows the trade-off between $p(t)$ and $r(t)$ introduced by the Naive Bayes classifier decision rule Eq. 4.23

been fixed to $a = 0.25$ following the analysis reported in Fig. 4.5, which shows that this value allows to reach $f = 0.95$ with only 10 configurations and limits the dependence on the value of N. The plot also shows that the saturation effect tends to occur sooner when a is small, as this corresponds to a denser and therefore less informative sampling of the interval.

In Tab. 4.3, we can observe that, for the FFIEC-SEC task, even with a small number of features $N = 5$, stacking leads to a significant increase of the F-score of the algorithm (12%), obtained by increasing both precision and recall at the same time. Increasing the number of features N tends to increase the efficiency, with a saturation effect as the number gets larger. Indeed, going from $N = 5$ to $N = 10$ only grants a 1% gain and no difference of efficiency is observed from $N = 10$ to $N = 20$. A similar behavior is observed for the DOREMUS tasks, where the big efficiency leap is given by the introduction of stacking, whereas raising the number of features N grants small improvements in the case of 4-heterogeneities and decreases the efficiency in the case of 9-heterogeneities. A similar behavior is observed for all the experiments that we have done.

To show that the increase of efficiency is not dependent on the particular threshold-based classifier, we have run the same experiments using STEM-LIN and reported the results in Tab. 4.4. In this case, we can observe that, although absolute values are

		FFIEC-SEC				DOREMUS 4-heterogeneities				DOREMUS 9-heterogeneities			
Base classifier	**N**	**p**	**r**	**f**	**δf**	**p**	**r**	**f**	**δf**	**p**	**r**	**f**	**δf**
Duke	n/a	0.88	0.77	0.82	0	0.46	0.57	0.51	0	0.48	0.66	0.55	0
STEM-NB	5	0.90	0.98	0.94	12%	0.89	0.98	0.93	42%	**0.97**	**1.0**	**0.99**	**44%**
STEM-NB	10	**0.93**	**0.97**	**0.95**	**13%**	0.89	0.98	0.93	42%	0.94	1.0	0.97	42%
STEM-NB	20	0.94	0.97	0.95	13%	**0.89**	**0.99**	**0.94**	**43%**	0.87	1.0	0.93	35%

Table 4.3 Results of STEM-NB vs Duke for $a = 0.25$ and different values of N across different datasets

		FFIEC-SEC				DOREMUS 4-heterogeneities				DOREMUS 9-heterogeneities			
Base classifier	**N**	**p**	**r**	**f**	**δf**	**p**	**r**	**f**	**δf**	**p**	**r**	**f**	**δf**
Silk	n/a	0.57	0.67	0.59	0	0.45	0.43	0.43	0	0.46	0.81	0.58	0
STEM-LIN	5	0.77	0.81	0.79	20%	**0.82**	**0.93**	**0.86**	**43%**	**0.89**	**1.0**	**0.94**	**36%**
STEM-LIN	10	0.78	0.83	0.80	21%	0.75	0.66	0.69	28%	0.89	1.0	0.94	36%
STEM-LIN	20	**0.77**	**0.84**	**0.81**	**22%**	0.75	0.60	0.64	21%	0.87	1.0	0.93	35%

Table 4.4 Results of STEM-LIN vs Silk for $a = 0.25$ and different values of N across different datasets

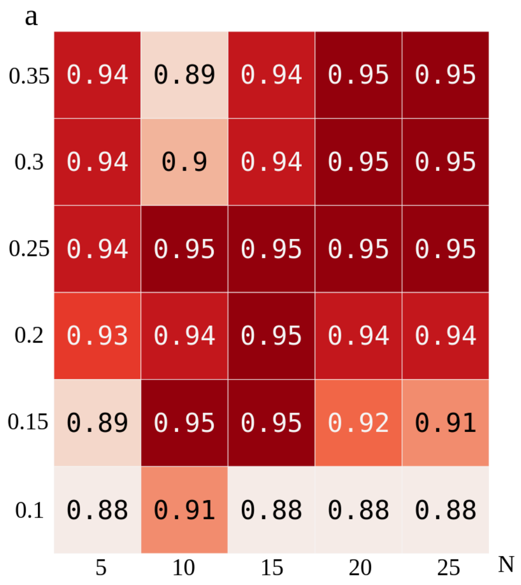


Fig. 4.5 F1 score for different combinations of a and N on the FFIEC-SEC dataset for STEM-NB

lower, the increase in efficiency given by the stacking layer is equally or even more important, achieving a +20% on the F-score with only $N = 5$ in the FFIEC-SEC task and a +43% and +36% for DOREMUS tasks. Also in this case, both precision and recall are increased at the same time and a saturation effect can be detected as N grows. In general, it seems that the saturation effect occurs earlier for DOREMUS tasks, as in three cases out of four with $N = 5$ we already reach the peak efficiency and the fourth case the efficiency only increases by 1%. This is probably due to the fact that DOREMUS datasets are smaller and thus a model with too many features tends to overfit the data.

4.4.2 STEM vs supervised learning on similarity values

In this section, we address the second research question of the chapter, namely RQ2.2: *How does STEM performs when little training data is available?*

We compare the hybrid approach of STEM with a system that performs machine learning 'from scratch'. More in detail, we have compared STEM to a number of commonly used machine learning algorithms, using similarity values s_i as features. In addition to verifying whether STEM performs better than the other systems in absolute, the intent is also to see whether it is less dependent on the amount of annotated training data. Indeed, given the quadratic nature of the entity matching problem, in most real usage scenarios, annotating a comprehensive gold standard (such as those of FEIII and DOREMUS) is an extremely time consuming endeavour and the user is able to annotate just a small fraction of all possible entity pairs. Therefore, it is interesting to see how an entity matching system performs with a small amount of annotated training pairs. To this end, we have studied how STEM performs at the variation of the amount of training data with respect to an SVM classifier with a RBF kernel, a random forest and a logistic classifier. In order to avoid possible size effects on the scores, we have split the FEIII data in two halves, according to the stratified sampling technique, i.e. keeping constant the proportion of matching and non matching pairs in the two parts. The first half is used as training data and the second half is used as test data. Then, we randomly extract a fraction z of training data from 0.1 to 0.9, train the systems and score

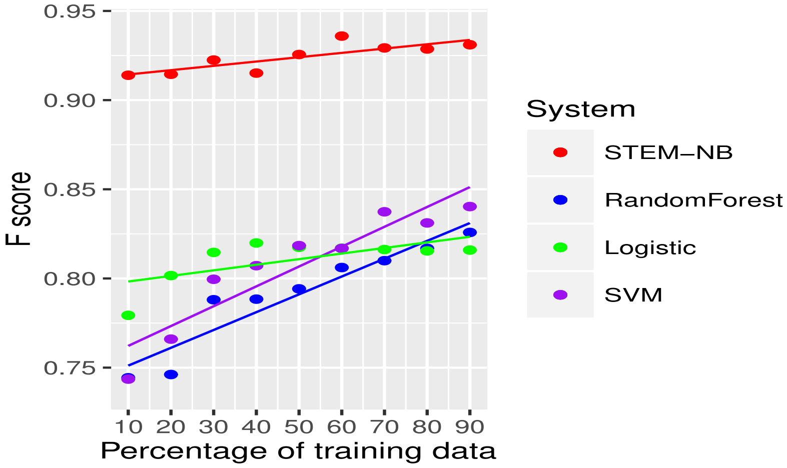


Fig. 4.6 F-score at the variation of the percentage of training data used. STEM-NB is compared to an SVM classifier, a Random Forest and a logistic classifier

them on the test set, which remains the same. For each value of z, we repeat the extraction 50 times and we compute the average value. Using the FEIII datasets and the STEM-NB implementation, values of s_i have been computed using the same comparators with the same configuration of STEM-NB. The configuration procedure of the machine learning classifiers is the same as that described in Sec. 4.3.5, namely a grid search hyper parameters optimization has been used to maximize 4-fold cross validation scores, setting C and γ for SVM, 'n_estimators' for the random forest and the regularization constant C for logistic regression.[15] The result of the experiment is depicted in Fig. 4.6. We can see that STEM-NB performs better than any other classifier in absolute terms, reaching a peak of 0.931 when 90% of the training data is used. Moreover, it shows little dependency on the amount of training data, producing 0.914 with only 10% of the training data. SVM performs better than the other pure machine learning approaches when 90% of training data is used, but decreases fast when annotated examples are reduced. In Tab. 4.5, we report, for each classifier, the quantitative estimation of the dependency of f from the fraction of training data z, obtained through the statistical estimation of the angular coefficient m of a linear fit of the points (i.e. the straight lines of Fig. 4.6). What we can observe is that more complex models such as SVM and Random Forest tend to depend more on the amount of training data, while a simple linear model such as logistic regression is

[15] http://scikit-learn.org/stable/user_guide.html

Classifier	**min**	**max**	**m**
STEM-NB	0.91	0.93	0.015 ± 0.008
SVM	0.74	0.84	0.09 ± 0.01
Random Forest	0.74	0.83	0.09 ± 0.01
Logistic	0.78	0.82	0.002 ± 0.006

Table 4.5 Dependency on the amount of training data. ‘Min’ and ‘Max’ represent respectively the minimum and maximum F-score and ‘m’ represents the angular coefficient of a straight line interpolating the points of Fig. 4.6

performing well even with a small amount of training data. The logistic model is even less dependent on the training data than a hybrid approach such as STEM, but it is not comparable in terms of absolute efficiency. STEM thus represents a model that is complex enough to achieve good efficiency in absolute terms and it is also able to maintain it with a little amount of training data.

4.4.3 Runtime performance

In Sec. 4.2.3 we have discussed the computational complexity of STEM, which can be summarized as $T_{STEM} \approx N * T_{baseclassifier}(n,m) + T_{stacking}(N,g,k)$. We have also argued that we observed that the time required to run the ensemble of base classifiers is longer than the time required for the stacking layer. In this section, we show an example of such a behavior measuring the runtime of STEM-NB on the DOREMUS 4-heterogeneities dataset with increasing number of features N. In Fig. 4.7, we can see that the time required to generate the features $T_{base} = N * T_{base\_classifier}$ quickly becomes much more significant than the time required for stacking $T_{stacking}$. A similar behavior has been observed for all the datasets under consideration, suggesting that a parallelization of the feature generation process would greatly improve the runtime performance of STEM as a whole. In general, we also observe a linear trend in both the components $T_{base}(N)$ and $T_{stacking}(N)$, which implies a linear trend for the total time T_{STEM}, consistently with what we expect from Sec. 4.2.3.

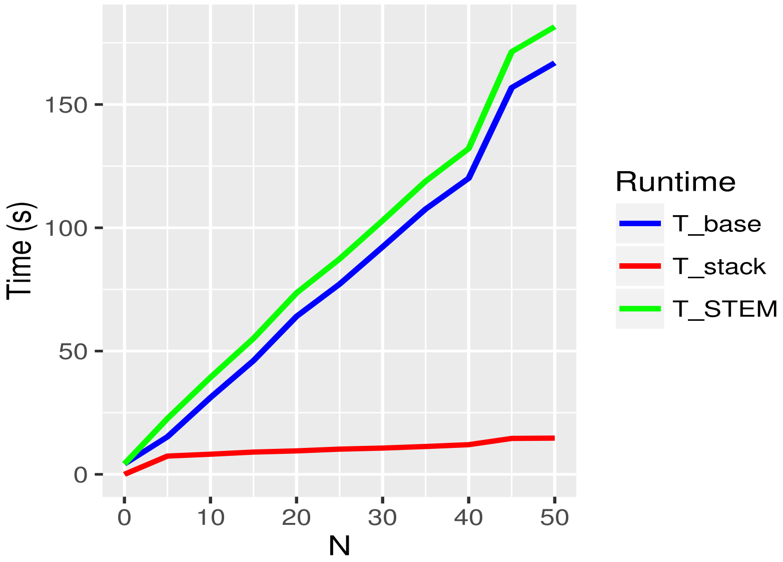


Fig. 4.7 Runtime for STEM-NB on DOREMUS 4-heterogeneities task with increasing number of features N.

4.5 Use-case: building the 3cixty Knowledge Graph

We further validate STEM by describing its implementation in a concrete use case, represented by the 3cixty European research project.[16] We address the third research question of the chapter RQ2.3: *How can STEM be applied in the process of building a knowledge graph containing Points of Interests (POI) and events for tourists?*

4.5.1 Overview

3cixty is a semantic web platform that enables to build real-world and comprehensive knowledge bases in the domain of culture and tourism for cities. The entire approach has been tested first for the occasion of the Expo Milano 2015 [200], where a specific knowledge base for the city of Milan was developed. Later, it has been refined with the development of knowledge bases for other cities, among those Nice. These knowledge bases contain descriptions of events, places (sights and businesses), transportation facilities and social activities, collected from numerous static, near- and real-time local and global data providers, including Expo Milano 2015 official

[16] https://www.3cixty.com

services in the case of Milan, and numerous social media platforms. The generation of each city-specific 3cixty knowledge base follows a strict data integration pipeline, that ranges from the definition of the data model, the selection of the primary sources used to populate the knowledge base, till the entity matching used for distilling the entities forming the final stream of cleaned data that is then presented to the users via multi-platform user interfaces. A detailed description of the technology leading to the creation of the knowledge base is reported in [29]. In the remainder of this section we introduce the creation of the gold standard for 3cixty and the experimental results obtained using STEM for the entity matching problem.

4.5.2 Gold Standard Creation

The 3cixty knowledge bases contain information about places, events, artists, transportation means, and user-generated content such as media and reviews. The knowledge bases are built using three types of data sources:

- local sources usually offered by city open data portals,
- global sources such as social media platforms,
- editorial data generated by experts of the domain.

STEM is used to remove duplicates among these heterogeneous data sources in the construction of the knowledge graph. Although ideally we might want to create a gold standard for all the cities where 3cixty is applied, this approach would not be scalable, as it would require a manual effort for each new city. Hence, we have generated a gold standard only from the 3cixty Nice KB, i.e. the knowledge base built for the Nice area. Given the generality of the stacking layer of STEM, which only uses numerical thresholds as features, and the of the data model of the different 3cixty knowledge bases, the SVM model can be trained once using the 3cixty Nice gold standard, and can then be applied to other cities.

The gold standard has gone through a process of identifying, with a random sampling, a small portion of Place-type pairs[17] to match, totaling 756 pairs. This

[17] For the sake of brevity we report the entity matching process of the Place-type entities

accounts to a tiny fraction of the entire set of possible pairs (order of 10^9 possible pairs); then, two human experts rated each as a match or as no-match. The annotation process was divided in two steps: `i)` individual annotation, i.e. each expert performed annotations separately; `ii)` adjudication phase, i.e. the two experts compared the annotations and resolved eventual conflicts.

This has prompted the creation of a gold standard that accounts 228 match and 528 no-match pairs.[18]

4.5.3 Experimental Results

Similarly to what has been done in Sec. 4.4.1, we compared STEM with Duke. In order to put Duke in the best conditions, we let it learning the best configurations using the active learning built-in function, just giving as input the instance fields to be utilized in the matching task and the gold standard created by the two experts.

The built-in active learning function works as follows: it iterates multiple times changing the configurations of the comparators aiming to minimize the matching error rate. Such a process prompts the creation of a configuration file summarizing the best Duke settings for the dataset used.

Having observed that it performs better than STEM-LIN (Sec. 5.5.6), we have then deployed STEM-NB using Duke configured as above and we conducted a 4-fold cross validation. Table 4.6 shows the results of the experiments. We can observe how STEM with five classifiers holds better results than a single run of Duke with a $\delta\mathbf{f}$ of 20%. We can also observe how the boost STEM introduces is slightly reduced with an increasing number of Duke instances N, similarly to what observed for DOREMUS data. As we mentioned earlier in the paper, this is the typical overfitting problem, where introducing additional complexity in the model does not provide better learning. As a general suggestion, $N = 5$ seems to be enough to obtain a consistent increment of efficiency with respect to the baseline without overfitting the data. The matching process with $N = 5$ took approximately 3 hours on a laptop with 4 cores and 12GB of RAM.

[18] We aim to share the Gold Standard once the paper is published to foster the reuse and experimental reproducibility.

Base classifier	**N**	**p**	**r**	**f**	**δf**
Duke	n/a	0.76	0.65	0.70	0
STEM-NB	**5**	**0.90**	**0.92**	**0.90**	**20%**
STEM-NB	10	0.76	0.81	0.78	8%
STEM-NB	20	0.79	0.81	0.79	9%

Table 4.6 Results of STEM-NB vs Duke on the 3cixty Nice dataset for $a = 0.25$ and different values of N.

4.6 Summary

In this chapter, we have described STEM: Stacked Threshold-based Entity Matching. First, we have introduced and explained the entity matching problem, introducing a set of formal definitions that allowed us to show that the use of threshold-based classifiers introduce a trade-off between precision and recall. Then, we described STEM, a machine learning layer that combines the predictions of a set of threshold-based classifiers through stacking. We have shown that stacking breaks the trade-off between precision and recall, significantly enhancing the F-score, using two different threshold-based classifiers and three different datasets. We have also shown that STEM is less dependent on the amount of training than a system that performs machine learning from 'scratch', i.e. using directly property similarity values. Finally, we have described how STEM has been applied in a real use-case: the construction of the 3cixty knowledge graph. We have described the aim of the 3cixty project and shown how STEM has been an important element of the process of knowledge graph generation, improving the data reconciliation process and leading to a higher quality of the graph.

Chapter 5

Path Recommender: Predicting Your Next Stop-over with Recurrent Neural Networks

Where should I go next? All of us have probably asked this question while visiting a city. Traditional tourist guides typically provide a set of predefined tours curated by experts, which can be useful to support a city exploration. However, predefined tours need to be manually updated, do not take into account personal preferences (e.g. art lover vs sports fan) or contextual information (e.g. sunny vs rainy weather). Thus, researchers in the RS field are trying to work on algorithms that learn to recommend tourist paths from data, providing scalability and the possibility of including personal preferences and contextual information. Often, these studies have made use of Location-based Social Networks (LBSN) data. A famous LBSN is Foursquare[1], which, through its application Swarmapp[2] allows users to check-in in a POI, sharing their location with friends. The next POI prediction problem has the goal of predicting where the user will go next, given a set of user's check-ins. Given that the user has checked-in in an *Italian Restaurant* and in a *City Park*, where is she likely to go next? Note that predicting these sequences require an implicit modeling of at least two dimensions: 1) temporal, as certain types of venues are

[1] https://it.foursquare.com/

[2] https://www.swarmapp.com/

more temporally related than others (e.g. after an *Irish Pub*, people are more likely to go to *Karaoke* than to a *History Museum*) 2) personal, as venue categories implicitly define a user profile, independently from their order (e.g. *Steakhouse* and *Vegetarian Restaurant* do not go frequently together).

In order to address this problem, it is necessary to define appropriate models that are specifically meant to deal with sequential data, to have at disposal a dataset containing sequences of check-ins to use for training, and to well define a standard and reproducible evaluation protocol that allows a fair comparison with competing approaches.

To address these challenges, in this chapter, we address the following research questions:

RQ3 *How can we create a recommender system that learns to recommend tourist paths from LSBN data, effectively leveraging the temporal correlation among tourist activities?*

To answer this research question, we introduce the Path Recommender and three sub-research questions:

RQ3.1 *How can we benchmark different SARS, improving the comparability and reproducibility of experiments?*

RQ3.2 *How do deep learning methods perform compared to other more traditional modelling approaches in the generation of tourist paths?*

RQ3.3 *How can we extend the Path Recommender to deal with the automated music playlist continuation task?*

The remainder of the chapter is structured as follows. In Sec. 5.1 we formalize the problem, providing a set of definitions generally related to sequence-aware recommender systems; in Sec. 5.2 we describe the RNN architecture of the Path Recommender; in Sec. 5.3 we describe the evaluation protocol Sequeval, which is used to evaluate the Path Recommender and compare it to a set of sequence-aware

algorithms; in Sec. 5.3.4 we describe the Semantic Trails Dataset (STD), which has been collected from Foursquare and has been used to train the Path Recommender; in Sec. 5.4, we describe the experimental results obtained from the comparison of the Path Recommender with competing systems; in Sec. 5.5 we describe the extension of the Path Recommender model devised to address the music playlist continuation task in the RecSys2018 challenge; in Sec. 5.6 we summarize the chapter.
Part of the work described in this chapter has been published in different workshops of the RecSys conference [30, 31, 33, 32].

5.1 Definitions

In a traditional recommender system, users express positive or negative preferences about a certain item. An item may be, for example, a product, a song, or a place. In contrast, we assume that when a user consumes or interacts with an item, she expresses an implicit rating about it. This assumption in literature goes under the name of *implicit feedback* (Sec. 2.2.1). Since we are also considering the temporal dimension in order to build the sequences, each rating is associated with a timestamp that represents the point in time when it was recorded.

Definition 22 *Given the space of items $\mathscr{I}$, the space of users $\mathscr{U}$, the space of timestamps $\mathscr{T}$, a rating* $\mathbf{r} \in \mathscr{R}$ *is a tuple* $\mathbf{r} = (\iota, \upsilon, \tau)$*, where $\iota \in \mathscr{I}$ is the item for which the user $\upsilon \in \mathscr{U}$ expressed a preference at the timestamp $\tau \in \mathscr{T}$.*

By relying on the set of ratings $\mathscr{R}$ available in the system, it is possible to construct the sequences that will be used to train and to evaluate the recommender. Each sequence only includes the ratings expressed by a single user. On the other hand, each user may produce several sequences.

The concept of *sequence* is similar to the concept of *session* in a traditional web interaction: if two ratings are distant in time more than an interval $\delta\tau$, then they belong to different sequences. Some ratings may be isolated and, for this reason, not part of any sequence. The most appropriate value for $\delta\tau$ depends on the domain: for example, in the point-of-interest recommendation scenario, it could be considered of a few hours as reported in [132].

Definition 23 *A sequence* $\mathbf{s} \in \mathscr{S}$ *is a temporally ordered list of ratings* $\langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n \rangle$ *created by a particular user* $\upsilon \in \mathscr{U}$*, i.e., for each i,* $\mathbf{r}_i = (\iota_i, \upsilon, \tau_i)$ *and* $\tau_i < \tau_{i+1}$.

In Algorithm 1, we list the procedure for creating the set $\mathscr{S}$, given the set of users $\mathscr{U}$, the set of ratings $\mathscr{R}$, and a time interval $\delta\tau$.

Algorithm 1 Generation of the set $\mathscr{S}$, given $\mathscr{U}$, $\mathscr{R}$, and $\delta\tau$.

```
Require: $\mathscr{U} \neq \{\emptyset\} \wedge \mathscr{R} \neq \{\emptyset\} \wedge \delta\tau \doteq \tau_i - \tau_j$
$\mathscr{S} \leftarrow \{\emptyset\}$
for all $\upsilon \in \mathscr{U}$ do
    $\mathbf{s} \leftarrow \varnothing$
    for all $\mathbf{r}_i \in \mathscr{R}_\upsilon : \tau_{i-1} < \tau_i \wedge i > 1$ do
        if $\tau_i < \tau_{i-1} + \delta\tau$ then
            if $\mathbf{s}$ is $\varnothing$ then
                $\mathbf{s} \leftarrow \langle \mathbf{r}_{i-1} \rangle$
            end if
            $\mathbf{s} \leftarrow \mathbf{s} + \langle \mathbf{r}_i \rangle$
        else
            if $|\mathscr{R}_\mathbf{s}| > 1$ then
                $\mathscr{S} \leftarrow \mathscr{S} \cup \{\mathbf{s}\}$
                $\mathbf{s} \leftarrow \varnothing$
            end if
        end if
    end for
end for
return $\mathscr{S}$
```

A sequence-based recommender is a RS capable of suggesting a personalized sequence that is built starting from a seed rating $\mathbf{r}_0$, considering the example sequences already available in the system and the specific behavior of a certain user. The seed rating is characterized by a seed item ι_0, a target user υ, and an initial timestamp τ_0. The seed item can be represented by any item that belongs to the catalog, but, more in general, it is a point in the space of items $\mathscr{I}$. For example, in the music domain, it could identify not only a particular song, but also an artist, a genre, or a mood.

The target user is the user to whom the sequence is recommended, while the initial timestamp represents the point in time in which the recommendation is created. The generated sequence is of fixed length and it contains exactly k ratings. Note that if $k = 1$, we are dealing with a sequential recommender as defined in [152].

Definition 24 *Given a seed rating* $\mathbf{r}_0 \in \mathscr{R} : \mathbf{r}_0 = (\iota_0, \upsilon, \tau_0)$*, and a length* $k \in \mathbb{N}$*, a sequence-based recommender is the function* $\text{sequence} : \mathscr{R} \times \mathbb{N} \to \mathscr{S}$*, i.e.* $\text{sequence}(\mathbf{r}_0, k) = \langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k \rangle : \mathbf{r}_i = (\iota_i, \upsilon, \tau_i)$*.*

Most sequence-based recommenders are based on probability models, and therefore they can be interpreted as a sampling function σ applied to the conditional probability $P(\langle \mathbf{r}_k, \mathbf{r}_{k-1}, \ldots, \mathbf{r}_1 \rangle | \mathbf{r}_0)$:

$$\text{sequence}(\mathbf{r}_0, k) = \sigma(P(\langle \mathbf{r}_k, \mathbf{r}_{k-1}, \ldots, \mathbf{r}_1 \rangle | \mathbf{r}_0)) \tag{5.1}$$

Using the chain rule, the sequence probability $P(\langle \mathbf{r}_k, \mathbf{r}_{k-1}, \ldots, \mathbf{r}_1 \rangle | \mathbf{r}_0)$ can be written as:

$$\begin{aligned} P(\langle \mathbf{r}_k, \mathbf{r}_{k-1}, \ldots, \mathbf{r}_1 \rangle | \mathbf{r}_0) = & P(\mathbf{r}_k | \langle \mathbf{r}_{k-1}, \ldots, \mathbf{r}_0 \rangle) \cdot \\ & P(\mathbf{r}_{k-1} | \langle \mathbf{r}_{k-2}, \ldots, \mathbf{r}_0 \rangle) \cdots P(\mathbf{r}_1 | \langle \mathbf{r}_0 \rangle) \end{aligned} \tag{5.2}$$

For example, in the case of a Markov chain, each rating depends on the previous one, i.e. $P(\mathbf{r}_k | \langle \mathbf{r}_{k-1}, \ldots, \mathbf{r}_0 \rangle) = P(\mathbf{r}_k | \mathbf{r}_{k-1})$:

$$P(\langle \mathbf{r}_k, \mathbf{r}_{k-1}, \ldots, \mathbf{r}_1 \rangle | \mathbf{r}_0) = P(\mathbf{r}_k | \mathbf{r}_{k-1}) P(\mathbf{r}_{k-1} | \mathbf{r}_{k-2}) \cdots P(\mathbf{r}_1 | \mathbf{r}_0) \tag{5.3}$$

Thus, a sequence-based recommender system typically works by learning from a set of sequences $\mathscr{S}_{training}$ the conditional probability of the next rating $\mathbf{r}_k$ to the sequence of previous ones $\langle \mathbf{r}_{k-1}, \mathbf{r}_{k-2}, \ldots, \mathbf{r}_0 \rangle$, i.e. the factors of the right-hand side of Equation 5.2. Sampling sequences directly from Equation 5.2 would require computing the probabilities of all the $|I|^k$ possible sequences, where $|I|$ is the size of the vocabulary of items and k is the length of the sequences. Since this becomes easily computationally unfeasible, we opt for a greedy approach, in which at each step we sample the next most likely item. A sampling function ρ is defined in order to select a particular next rating from the previous ones at each step:

$$\hat{\mathbf{r}}_k = \rho(P(\mathbf{r}_k | \langle \mathbf{r}_{k-1}, \mathbf{r}_{k-2}, \dots, \mathbf{r}_0 \rangle)) \tag{5.4}$$

A trivial example of ρ is the *argmax* function, which simply selects the most probable next rating. In the following, we will assume that ρ is implemented by a weighted random sampling function.

Algorithm 2 formalizes the procedure for generating a personalized sequence, given a seed rating $\mathbf{r}_0$, and a length k, i.e. it describes the sampling function σ. For k times, the next rating of the recommended sequence is generated using the function *predict*. The function $predict : \mathscr{S} \rightarrow \mathscr{R}$ implements the sampling function ρ and it returns the most probable next rating for the current input sequence. In practice, the sequence-based recommender system can estimate the probability that the next rating of the current sequence will include a particular item at a certain timestamp.

Note that the greedy procedure described in Algorithm 2 is not the only way to create samples of sequences, but only the one that is adopted in this work for his computational efficiency. Some alternatives are the brute-force approach where the probability of all possible sequences of length k is estimated, or beam search approaches where the probability of sub-sequences of length $11 < k_{sub} < k$ are estimated for a number of steps $k//k_{sub}$ of times.

Algorithm 2 Recommendation of a sequence of length k.

```
Require: $\mathbf{r}_0 \in \mathscr{R} \wedge k > 0$
$\mathbf{s} \leftarrow \langle \mathbf{r}_0 \rangle$
for $i = 1$ to $k$ do
    $\mathbf{r}_i \leftarrow \mathrm{predict}(\mathbf{s})$
    $\mathbf{s} \leftarrow \mathbf{s} + \langle \mathbf{r}_i \rangle$
end for
return $\mathbf{s} - \langle \mathbf{r}_0 \rangle$
```

In order to compute some metrics that are part of the evaluation framework, it is necessary to know the number of items that are associated with a certain sequence.

For this reason, we define the set $\mathscr{I}_{\mathbf{s}}$ as the set of items that are part of the sequence $\mathbf{s}$, and the set $\mathscr{R}_{\mathbf{s}}$ as the set of ratings that are part of the sequence $\mathbf{s}$. Therefore, $|\mathscr{I}_{\mathbf{s}}|$ is the number of distinct items available in $\mathbf{s}$, while $|\mathscr{R}_{\mathbf{s}}|$ represents the length of $\mathbf{s}$.

For instance, we can suppose that the set of ratings $\mathscr{R}$ is equal to $\{(\iota_1,\upsilon_1,\tau_1), (\iota_2,\upsilon_1,\tau_2),(\iota_3,\upsilon_2,\tau_3),(\iota_1,\upsilon_1,\tau_4),(\iota_2,\upsilon_2,\tau_5),(\iota_3,\upsilon_1,\tau_6)\}$. Then, if we assume that the only pair of timestamps that violates the $\delta\tau$ constraint is (τ_4,τ_6), we can create two sequences: $\mathbf{s}_1 = \langle(\iota_1,\upsilon_1,\tau_1),(\iota_2,\upsilon_1,\tau_2),(\iota_1,\upsilon_1,\tau_4)\rangle$, and $\mathbf{s}_2 = \langle(\iota_3,\upsilon_2,\tau_3), (\iota_2,\upsilon_2,\tau_5)\rangle$. The rating $(\iota_3,\upsilon_1,\tau_6)$ is not part of any sequence because it was created at some point in time later than $\tau_4+\delta\tau$ and we do not have any subsequent rating expressed by υ_1. We also observe that $|\mathscr{I}_{\mathbf{s}_1}| = 2$ and $|\mathscr{R}_{\mathbf{s}_1}| = 3$. We would like to recommend a sequence $\mathbf{s}$ of length 2 to user υ_1 starting from item ι_3 at timestamp $\tau_{\mathbf{s},0}$. A possible solution to this problem is to define $\mathbf{r}_{\mathbf{s},0} = (\iota_3,\upsilon_1,\tau_{\mathbf{s},0})$ and then to recommend $\mathbf{s} = \text{sequence}(\mathbf{r}_{\mathbf{s},0},2) = \langle(\iota_2,\upsilon_1,\tau_{\mathbf{s},1}),(\iota_1,\upsilon_1,\tau_{\mathbf{s},2})\rangle$, where $\tau_{\mathbf{s},1}$ and $\tau_{\mathbf{s},2}$ may be used to suggest when consuming the items.

In this work, we address the problem of next POI category prediction, i.e. we aim to learn to predict the category of the next POI that a user will visit, in order to be able to generate and recommend new paths. Thus, the role of the ratings is taken by the check-ins of the users.

Definition 25 *Given the space of POI categories C, the space of check-in ids I, the space of timestamps T, the space of users U, a check-in is a set $v = \{i,c,\tau,u\}$ where $i \in I$ is the check-in id, $c \in C$ is the category of the POI, $\tau \in T$ is the timestamp at which the check-in has been performed and $u \in U$ is the user who has performed the check-in.*

In this work, the space of possible categories C is defined by the Foursquare Taxonomy, which defines and classifies categories in a hierarchical ontology containing 920 categories[3]. We define a *path* (or trail) as a list of consecutive check-ins created by the same user within a certain amount of time:

[3] https://developer.foursquare.com/categorytree

Definition 26 *A path (or trail)* $\mathbf{s} \in \mathscr{S}$ *is a temporally ordered list of check-ins* $\langle \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n \rangle$ *created by a particular user* $\upsilon \in \mathscr{U}$*, i.e., for each i,* $\mathbf{c}_i = (\nu_i, \upsilon, \tau_i)$ *and* $\tau_i < \tau_{i+1}$.

Definition 27 *We define a category index* $\alpha \in \mathbb{N}$ *with* $\alpha = 1..|C|$ *and that uniquely identifies a category* $c^\alpha \in C$.

In order to learn to generate the next category c_{t+1} of a path, we learn a model of the conditional probability $P(c_{t+1}|c_t, c_{t-1}, c_{t-2}, ..., c_1)$ from these sequences of POI categories, in accordance to what described in Eq. 5.2, and sample from it the next POI (Eq. 5.4).

5.2 The Path Recommender model

Most of existing studies attempt to model directly sequences of POIs rather than their categories for the next POI prediction problem (see Sec. 2.2.3). In this work, we decide to focus on modeling sequences of POI categories to enhance the generality and the portability of the obtained results. This can be considered as a first step in the next POI prediction problem, as the POI category can then be turned into a specific POI by querying a database of POIs according to a variety of parameters, such as the user context (e.g. position, weather) and/or specific POI features such as popularity, average prices and the like.

We propose an approach based on Recurrent Neural Networks, which are specifically meant to deal with sequential data. The main difference of RNNs with respect to standard feed-forward neural networks is the presence of a hidden state variable h_t, whose value depends both on the input data presented at time x_t and, by means of loop connections, on the previous hidden state h_{t-1}[201]. A typical application of RNNs in neural language modeling is that of generating text recursively applying a “next word prediction" [147], and in the same spirit we address the problem of next POI category prediction. The main idea is that of using a supervised learning approach where the targets correspond to the inputs shifted in time, i.e. $X = \{(c^j{}_0, c^j{}_1, ..., c^j_{N_j-1})\}$ and $Y = \{(c^j{}_1, c^j{}_2, ..., c^j_{N_j})\}$ where $j = 1...M$ is the path index and N_j is the length of the $j-th$ path. The architecture of the neural network is illustrated in Fig. 5.1. To simplify the notation, we now drop the path index j and consider one path to illustrate the functioning of the network. A venue category c_t is fed into the network via an encoding into an input vector x_t, which is then passed to a Gated Recurrent Unit. Gated Recurrent Units (GRU) are gating mechanisms that improve the ability of the RNNs to store long sequences and that recently have been proven to be as effective as more complicated architectures such as Long Short-Term Memory (LSTM) units [143]. The update of the GRU unit hidden state, i.e. the computation of the new state h_t given the previous state h_{t-1} and the current input x_t, is described by the following equations:

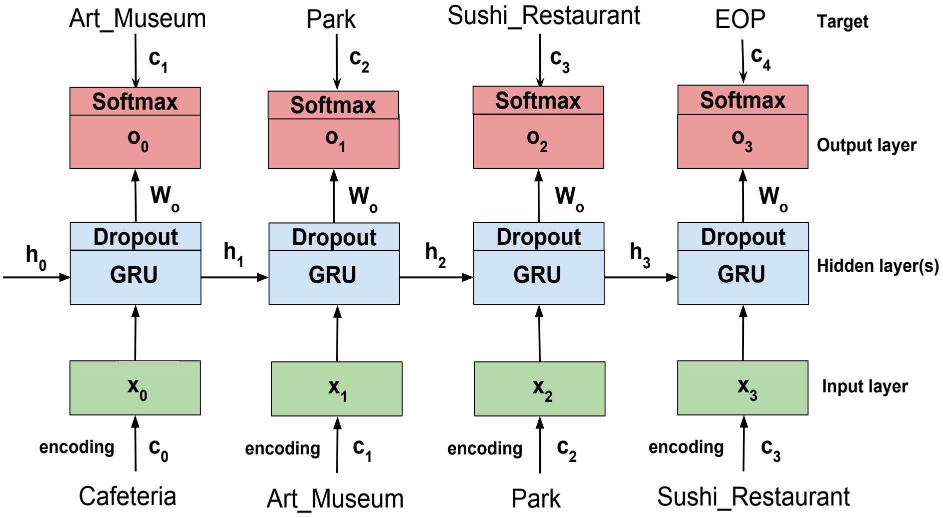


Fig. 5.1 Architecture of the RNN. ‘EOP’ is a special symbol describing the end of a path.

$$r_t = sigmoid(W_r h_{t-1} + W_r x_t + b_r) \tag{5.5}$$

$$h'_t = tanh(W_i(r_t \otimes h_{t-1}) + W_i x_t + b_i) \tag{5.6}$$

$$z_t = sigmoid(W_z h_{t-1} + W_z x_t + b_z) \tag{5.7}$$

$$h_t = z_t \otimes h' + (1 - z_t) \otimes h_{t-1} \tag{5.8}$$

where *sigmoid* and *tanh* indicate respectively the sigmoid and hyperbolic tangent activation functions and $\otimes$ represent the element-wise product of the matrices. r is called the ‘reset gate’ and it allows to forget or remember the previous state h_{t-1} when generating the candidate state h'_t. z is called the ‘update gate’ and intuitively it

controls how much the unit needs to update its state. W_i, W_r, W_z are weight matrices that are learned during the training.
The GRU computes the hidden state h_t which is stored for the next iteration and used to compute the output of the current iteration o_t. Before computing the output o_t, during training time, a Dropout layer is applied. The Dropout layer is a regularization mechanism which, at training time, randomly switches off a fraction p of neurons, called the *dropout rate*, preventing them from co-adapting and overfitting the sampled data [202]. Dropout can be modelled with a mask vector m_t, whose values can be either 1 or 0 with probability p. After the dropout layer, the output state $o_t = tanh(W_o h_t m_t)$ is computed using a fully connected layer whose weights are defined by the matrix W_o, which is learned at training time. W_o is shaped so that the dimension of the output vector is equal to the number of possible categories, i.e. $|o_t| = |C|$. Thus, we can index the components of the output vector using the category index o_t^α. Then, the Softmax layer normalizes the outputs, turning them into a probability distribution over a set of possible outcomes [203]:

$$softmax(o_t^\alpha) = \frac{e^{o_t^\alpha}}{\sum_{k=1}^{|C|} e^{o_t^k}} \tag{5.9}$$

In this way, the Softmax layer models the probability distribution of the next category:

$$softmax(o_t^\alpha) = P(c_{t+1} = c^\alpha | c_t, c_{t-1}, c_{t-2}, ..., c_1) \tag{5.10}$$

as o_t^i depends on the current category encoding x_t of c_t, but also on all the previous encodings of the sequence by means of the hidden state h_t. During the training process, we train the network to produce a probability distribution of categories that is as close as possible to that observed in the data, i.e. maximizing the probability of the observed data. Therefore, we define the loss L_t as the cross entropy:

$$L_t = -logP(c_{t+1} = c_{t+1}^\alpha | c_t, c_{t-1}, c_{t-2}, ..., c_1) = -log(softmax(o_t^\alpha)) \tag{5.11}$$

where c_{t+1}^α is the category observed in the data as $t+1$ element of the path. The loss is optimized using Adam [204], an enhanced version of the stochastic gradient descent that introduces momentum and adaptive learning rates. The gradients of the loss function are computed using back propagation on the unrolled neural network [205]. The model has a number of hyper-parameters, such as the number of neurons in the hidden state n_{hidden}, the number of hidden layers n_{layers}, the learning rate l_r and the number of epochs η. We optimize these hyper parameters using a grid search on a validation set, setting: ($n_{hidden} = 64$, $l_r = 10^-4$, $epochs = 5$, $n_{layers} = 3$).

5.3 Experimental setup: the Sequeval framework

Comparing the performance of several recommenders with an experiment that involves a live system is not always feasible or appropriate. For this reason, it is necessary to first perform a preliminary evaluation in an offline scenario [206]. However, in SARS there is a lack of standard evaluation protocols [24]. Thus, in this section, we address RQ3.1: *How can we benchmark different SARS, improving the comparability and reproducibility of experiments?*
To this end, we introduce the evaluation framework Sequeval. Sequeval is made of an evaluation protocol, presented in Section 5.3.1, a set of evaluation metrics, described in Section 5.3.2, and a software implementation that is introduced in Section 5.3.3.

5.3.1 Evaluation Protocol

One of the first problems that an evaluation framework should consider is how to split the dataset between the training set and the test set. This task is not trivial, as it will deeply influence the outcome of the experimentation [160]. Since we are dealing with sequences, we need to split the set of sequences $\mathscr{S}$ in a training and test set such that $\mathscr{S} = \mathscr{S}_{training} \cup \mathscr{S}_{test}$.

Several solutions to this problem are possible: a simple but effective one is to perform the splitting by randomly assigning sequences to these sets according to a certain ratio, typically the 80% for training and the 20% for testing. If the number of sequences available is limited, it is necessary to perform a cross-validation. Another possibility is to identify an appropriate point in time and to consider all the sequences created before it as part of the training set, and after it as part of the test set. This protocol simulates the behavior of a recommender introduced at that point in time and it avoids too optimistic results caused by the knowledge of future events [207]. The latter solution can be considered the most reliable one, but if we do not have any temporal information because the sequences have already been created, it is necessary to adopt a random protocol.

More in general, it is impossible to identify a splitting method that is appropriate for every experiment, as it depends on the domain and on the dataset available. For this reason, Sequeval does not impose the adoption of a particular splitting protocol, but the experimenter can choose the most appropriate one.

In order to compute the metrics that are part of the evaluation framework, the sequence-based recommender is trained with all the sequences $\mathbf{s} \in \mathscr{S}_{training}$. Then, for each test sequence $\mathbf{s} \in \mathscr{S}_{test} : \mathbf{s} = \langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n \rangle$, we predict a recommended sequence $\bar{\mathbf{s}}$ of length k using $\mathbf{r}_1 = (\iota_1, \upsilon, \tau_1)$ as seed rating, i.e. $\bar{\mathbf{s}} = \text{sequence}(\mathbf{r}_1, k)$. Therefore, $\bar{\mathbf{s}}$ is a sequence suggested by the recommender, for the same user and starting from the same item of $\mathbf{s}$. We also define $\mathbf{s}'$ as the reference sequence, i.e. $\mathbf{s}' = \langle \mathbf{r}_2, \mathbf{r}_3, \ldots, \mathbf{r}_n \rangle$ or $\mathbf{s}' = \mathbf{s} - \langle \mathbf{r}_1 \rangle$. The reference sequence is equal to the original sequence, but the first rating is omitted, as it was already exploited for creating the recommended sequence. This procedure is graphically illustrated in Figure 5.2.

5.3.2 Evaluation Metrics

The second component of Sequeval is a set of eight metrics that we present in the following. We include in such set not only classic metrics like coverage and precision but also less widespread ones like novelty, diversity, and serendipity. Furthermore, we introduce the metric of perplexity, as it is explicitly designed for characterizing sequences [208].

Coverage

In general, the coverage of a recommender is a measure that captures the number of items in the catalog over which the system can make suggestions [206]. For

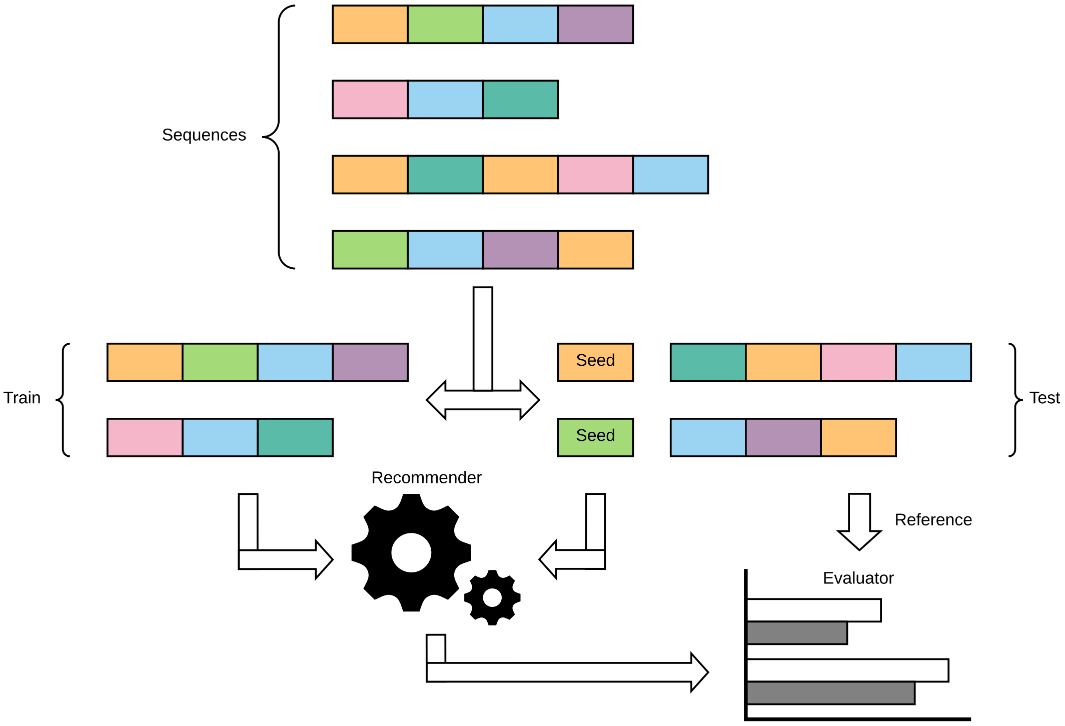


Fig. 5.2 An illustration of the evaluation procedure. First, the set of sequences is split in a training and a test set. Then, the recommender is trained with the sequences available in the training set. Finally, the recommender is asked to generate a sequence for each seed from the test set; such sequences are compared with the corresponding reference sequences.

example, in an online store scenario, it could represent the percentage of products that are recommended to users in a certain period of time. An algorithm with a higher coverage is generally considered more useful because it better helps users to explore the catalog.

We generate a set of recommended sequences considering as seed the first rating of all sequences in the test set $\mathscr{S}_{test}$ for a recommender that suggests sequences of length k. Afterwards, we compute the distinct number of items available in the sequences created and we divide the result by the cardinality of the set $\mathscr{I}$.

$$\text{coverage}(k) = \frac{|\bigcup_{\mathbf{s}\in\mathscr{S}_{test}} \mathscr{I}_{\text{sequence}(\mathbf{r}_1,k)}|}{|\mathscr{I}|} \tag{5.12}$$

This metric expresses the percentage of items that the sequence-based recommender is capable of suggesting when generating sequences similar to the ones available in the test set and it is strictly related to its cardinality. This approach is similar to the metric of prediction coverage described by Herlocker *et al.* [137].

Precision

Precision is a widespread metric in the context of information retrieval evaluation [209] and it represents the fraction of retrieved documents that are relevant. For a traditional recommender system, precision measures the fraction of recommended items that are relevant for a certain user [210]. If we consider a sequence-based recommender, it is necessary to compute this metric for each sequence $\mathbf{s} \in \mathscr{S}_{test}$, instead of each user.

$$\text{precision}(k) = \frac{1}{|\mathscr{S}_{test}|} \cdot \sum_{\mathbf{s}\in\mathscr{S}_{test}} \frac{\text{hit}(\mathbf{s}', \bar{\mathbf{s}})}{\min(|\mathscr{R}_{\mathbf{s}'}|, k)} \tag{5.13}$$

The function $hit : \mathscr{S} \times \mathscr{S} \to \mathbb{N}$ returns the number of items in $\bar{\mathbf{s}}$ that are also available in $\mathbf{s}'$. If the same item is present in $\bar{\mathbf{s}}$ multiple times, it is considered a hit only if it is repeated also in $\mathbf{s}'$. This is an extension to the traditional definition of precision that also considers the fact that an item may appear multiple times inside a sequence.

The number of relevant items is divided by the minimum number between the length of the reference sequence $|\mathscr{R}_{\mathbf{s}'}|$ and the length of the recommended sequence k. We decided to adopt this solution in order to avoid penalizing an algorithm that is evaluated considering reference sequences shorter than the recommended sequences.

nDPM

The Normalized Distance-based Performance Metric (nDPM) was originally proposed by Yao in the context of information retrieval [211]. The intuition of the author is that, in order to compare a system ranking with a reference user ranking, it is necessary to consider all the possible pairs of items available in the system ranking: they can be agreeing, contradictory, or compatible with respect to the user ranking. We decided to adopt such a metric instead of the Normalized Discounted Cumulative Gain (nDCG) [212] because, in a sequence of recommendations, it is not necessarily true that the first items are more important than the last ones.

$$\text{nDPM}(k) = \frac{1}{|\mathscr{S}_{test}|} \cdot \sum_{\mathbf{s} \in \mathscr{S}_{test}} \frac{2\,\text{pairs}^{-}(\mathbf{s}', \bar{\mathbf{s}}) + \text{pairs}^{u}(\mathbf{s}', \bar{\mathbf{s}})}{2\,\text{pairs}(\bar{\mathbf{s}})} \tag{5.14}$$

The function $pairs^{-} : \mathscr{S} \times \mathscr{S} \to \mathbb{N}$ returns the number of pairs in the sequence $\bar{\mathbf{s}}$ that are in the opposite order with respect to the reference sequence $\mathbf{s}'$. The function $pairs^{u} : \mathscr{S} \times \mathscr{S} \to \mathbb{N}$ returns the number of pairs in the sequence $\bar{\mathbf{s}}$ for which the ordering is irrelevant, i.e. when at least one of the items is not available in $\mathbf{s}'$ or when at least one of the items is available multiple times in $\mathbf{s}'$. Finally, the function $pairs : \mathscr{S} \to \mathbb{N}$ returns the number of all possible pairs available in the recommended sequence $\bar{\mathbf{s}}$. The pairs are created without considering the ordering of the items inside a pair: for example, if we have the sequence $\langle a, b, c \rangle$, the possible pairs are (a, b), $(a, c), (b, c)$.

The value of this metric will result close to 1 when the sequences generated by the recommender are contradictory, to 0 when they have the same ranking, and to 0.5 when the ordering is irrelevant because they contain different items. A low precision will imply a nDPM very close to 0.5.

Diversity

The metric of sequence diversity included in this framework is inspired by the metric of Intra-List Similarity proposed by Ziegler *et al.* [213]. The recommended sequences are considered as lists of items and the obtained value is not related to their internal ordering. The purpose of this metric is understanding if the sequences contain items that are sufficiently diverse. A higher diversity may be beneficial for the users, as they are encouraged to better explore the catalog [214].

$$\text{diversity}(k) = \frac{1}{|\mathscr{S}_{test}|} \cdot \sum_{\mathbf{s} \in \mathscr{S}_{test}} \frac{\sum_{\forall i, \forall j: 0<i<j}^{k} 1 - \text{sim}(\bar{\imath}_i, \bar{\imath}_j)}{k \times (k-1)} \tag{5.15}$$

The function *sim* : $\mathscr{I} \times \mathscr{I} \to [-1,1]$ is a generic similarity measure between two items. This measure may be taxonomy-driven or content-based: for example, a possible content-based similarity measure is the cosine similarity. The resulting value is a number in the interval $[0,2]$: higher values represent a higher diversity.

Novelty

Vargas *et al.* [170] suggested that it would be useful to be able to characterize the novelty of the recommendations. They proposed a metric that rewards algorithms capable of identifying items that have a low probability of being already known by a specific user because they belong to the long-tail of the catalog. We have included such metric in our framework in order to assess if the items available in the suggested sequences are not too obvious.

$$\text{novelty}(k) = -\frac{1}{|\mathscr{S}_{test}| \times k} \cdot \sum_{\mathbf{s} \in \mathscr{S}_{test}} \sum_{i=1}^{k} \log_2 \text{freq}(\bar{\imath}_i) \tag{5.16}$$

The function $freq : \mathscr{I} \to [0,1]$ returns the normalized frequency of a certain item $\imath \in \mathscr{I}$, i.e. the probability of observing that item in a given sequence $\mathbf{s} \in \mathscr{S}_{training}$. We can define the probability of observing the item $\imath$ as the number of ratings related to $\imath$ in the training sequences divided by the total number of ratings available. We also assume that $\log_2(0) \doteq 0$ by definition, in order to avoid considering as novel items for which we do not have any information, i.e. the items that do not appear in the training sequences.

Serendipity

Serendipity can be defined as the capability of identifying items that are both attractive and unexpected [168]. Ge *et al.* proposed to measure the serendipity of a recommender by relying on the precision of the generated lists after having discarded the items that are too obvious [169].

In order to create a list of obvious items, it is possible to exploit a *primitive* recommender, that is a recommender only capable of making obvious suggestions. For example, a primitive recommender could be implemented using the Most Popular (MP) baseline, which is defined in Section 5.3.3. It is reasonable to assume that popular items do not contribute to the serendipity of the recommendations because they are already well known by many users.

By modifying the metric of precision described in Section 5.3.2, it is possible to introduce the concept of serendipity in the evaluation of a sequence-based recommender. In this case, the primitive recommender will always create a sequence of length k that contains the items that are have been observed with the highest frequency in the training set.

$$\text{serendipity}(k) = \frac{1}{|\mathscr{S}_{test}|} \cdot \sum_{\mathbf{s} \in \mathscr{S}_{test}} \frac{\text{hit}(\mathbf{s}', \bar{\mathbf{s}} - \hat{\mathbf{s}})}{\min(|\mathscr{R}_{\mathbf{s}'}|, k)} \tag{5.17}$$

We define $\hat{\mathbf{s}}$ as the sequence generated by the primitive recommender from the same seed of $\bar{\mathbf{s}}$, i.e. $\hat{\mathbf{s}} = \text{primitive}(\mathbf{r}_1, k)$. Moreover, the sequence $\bar{\mathbf{s}} - \hat{\mathbf{s}}$ contains all the ratings related to the items available in $\bar{\mathbf{s}}$ that are not present in $\hat{\mathbf{s}}$. The resulting value will be a number in the interval $[0, 1]$, lower than or equal to precision. The difference between precision and serendipity represents the percentage of obvious items that are correctly suggested.

Confidence

The metric of confidence reflects how much the system trusts its own suggestions and it is useful for understanding how robust is the learned model [215]. It is usually computed as the average probability that the suggested items are correct. This metric expresses the point of view of the recommender, as the probability is reported by the model. Therefore, the metric is always equal to 1 with the most popular recommender, as it is certain of the predictions.

A sequence-based recommender generates the next item of the sequence by considering all the previous items. For this reason, we can interpret the conditional probability of obtaining a certain item, given the sequence of previous ones, as the confidence that the system has in that suggestion.

$$\text{confidence}(k) = \frac{1}{|\mathscr{S}_{test}| \times k} \cdot \sum_{\mathbf{s} \in \mathscr{S}_{test}} \sum_{i=1}^{k} P(\bar{\imath}_i | \bar{\imath}_{i-1}, \bar{\imath}_{i-2}, \dots) \tag{5.18}$$

We also define $\bar{\imath}_0 \doteq \imath_1$, i.e. the zero-*th* item of the recommended sequence is its seed item. Therefore, this metric is computed by also considering the probability of obtaining the first item $\bar{\imath}_1$, given the seed item of $\bar{\mathbf{s}}$.

Perplexity

Perplexity is a widespread metric in the context of neural language modeling evaluation [208], typically used to measure the quality of the generated phrases. Because there is a strong similarity between creating a sequence of natural language words and sequence of recommended items given an initial seed, perplexity can be also successfully exploited in this context.

This metric can be defined as the exponential in base 2 of the average negative log-likelihood of the model, i.e. the cross entropy of the model. For models based on the cross entropy loss function such as neural networks, the perplexity can also be seen as a measure of convergence of the learning algorithm. Differently from the metric of confidence, the conditional probability $P(\imath_{i+1}|\imath_i, \imath_{i-1}, \dots)$ is computed considering the items of the test sequence $\mathbf{s}$, and not of the recommended sequence $\bar{\mathbf{s}}$. For this reason, it does not express the point of view of the recommender.

$$\text{perplexity} = 2^{-\frac{1}{\sum_{\mathbf{s} \in \mathscr{S}_{test}} |\mathscr{R}_{\mathbf{s}}|-1} \cdot \sum_{\mathbf{s} \in \mathscr{S}_{test}} \sum_{i=0}^{|\mathscr{R}_{\mathbf{s}}|-1} \log_2 P(\iota_{i+1}|\iota_i,\iota_{i-1},\dots)} \tag{5.19}$$

Intuitively, the obtained value represents the number of items from which an equivalent random recommender should choose in order to obtain a similar sequence. The lower is the perplexity, the better is the system under evaluation. Therefore, the perplexity of a random recommender is equal to $|\mathscr{I}|$. If the performance of the recommender is worst than a random one, the perplexity will be higher than $|\mathscr{I}|$: for example, if only one conditional probability is equal to zero, then perplexity $= +\infty$.

5.3.3 Implementation

The third component of Sequeval is `sequeval` [216], a Python implementation of the evaluation framework which is publicly available.[4] This implementation is based on the evaluation protocol presented in Section 5.3.1 and it includes the metrics described in Section 5.3.2.

Furthermore, it provides four baseline recommenders, which can be interpreted as an adaptation of classic non-personalized recommendation techniques to our sequence-based scenario. In addition to these baselines, we also include a Conditional Random Field (CRF) to the comparison:

Most Popular The Most Popular (MP) recommender analyses the sequences available in the training set in order to compute the popularity of each item, i.e. the number of times an item appears in the training sequences. Then, at recommendation time, it ignores the seed rating, and it always creates a sequence that contains the most popular item as the first rating, the second most popular item as the second rating, and so on. More formally, the probability that the item ι_i will appear in the i-th rating of the sequence is $P(\iota_i) = 1$, where i also represents the position of the item in the ranking of the most popular ones.

[4] https://github.com/D2KLab/sequeval

Random The random recommender simply creates sequences composed of ratings that contain an item randomly sampled from a uniform probability distribution. The seed rating is discarded and the probability of observing the item $\imath_i$ is $P(\imath_i) = 1/|\mathscr{I}|$, where $|\mathscr{I}|$ represents the number of items available in the system.

Unigram The unigram recommender is capable of generating sequences that contain ratings with items sampled with a probability proportional to the number of times they were observed in the training sequences. In particular, the probability of observing the item $\imath_i$ is equal to the number of ratings containing $\imath_i$ divided by the total number of ratings available in the training sequences. Similarly to the previous baselines, the seed rating is ignored during the recommendation phase.

Bigram The bigram recommender estimates the 1-*st* order transition probabilities among all possible pair of items available in the training sequences. The add-one smoothing technique is exploited to avoid the attribution of a strict zero probability to the pairs that were not observed during the training phase [217]. At recommendation time, the seed rating is exploited for selecting the first item, and then each item will influence the choice of the next one. The probability of sampling item $\imath_i$ after item $\imath_{i-1}$ is equal to the number of times this transition occurred in the training sequences plus one divided by the total number of transitions available.

CRF the CRF-based recommender system is implemented using the `CRFsuite` software package.[5] Since we are interested in predicting an item given the previous one, we have considered as feature vectors the training sequences without their last rating and as corresponding output vectors the same sequences without their first rating. We have used the gradient descent algorithm with the L-BFGS method [218] as the training technique. We have chosen to generate both the state and the transition features that do not occur in the dataset and we have set the maximum number of iterations allowed for the optimization algorithm to 100.

[5] http://www.chokkan.org/software/crfsuite

5.3.4 Datasets

Different studies have been conducted by considering user-created geographical data obtained from LBSN and different datasets of check-ins collected from LBSN are already available. The NYC Restaurant Rich Dataset [219] includes check-ins of restaurant venues in New York City only, as well as tip and tag data collected from Foursquare from October 2011 to February 2012. The NYC and Tokyo Check-in Dataset [220] contains check-ins in New York City and Tokyo collected from April 2012 to February 2013, together with the timestamp, GPS coordinates and venue category of the check-in. The Global-Scale Check-in Dataset (GSCD) [221] includes long-term global-scale check-in data collected from the 415 most checked cities in the world on Foursquare. All of these datasets are publicly available on the Web.[6] However, none of these datasets is focused on temporal sequences of check-ins and we opt for collecting our own dataset.

We collected check-ins performed by the users of the Foursquare Swarm[7] mobile application and publicly shared on Twitter from the Twitter API.[8] Then, we retrieved the category of the place associated with the check-in thanks to the Foursquare API.[9] For this reason, the items of the dataset are represented by the venue categories available in the Foursquare taxonomy.[10]. In order to avoid using check-ins generated by bots, we have discarded the users that performed multiple check-ins in less than one minute. We have also pruned the check-ins associated with the venue categories that are usually not of interest for a tourist, for example the ones related to workplaces. In order to generate the sequences more efficiently, we decided to also remove the users that have performed less than 10 check-ins in total. We have set the $\delta\tau$ parameter of the evaluation framework Sequeval (Algorithm 1) to 8 hours. This means that two check-ins must not be distant in time more than eight hours belong to the same path, similarly to what has been done in [222]. Regarding the splitting protocol, we have selected the timestamp-based one, considering the timestamp associated with the first rating as the timestamp of the sequence.

[6] https://bit.ly/2DUhAQn
[7] https://www.swarmapp.com
[8] https://developer.twitter.com
[9] https://developer.foursquare.com
[10] https://developer.foursquare.com/docs/resources/categories

In order to test the RNN architecture on an additional domain, we have used an additional dataset. The dataset contains several playlists originally collected by Shuo Chen from *Yes.com* in the context of his research on metric embedding [138]. Such website provided a set of APIs[11] for programmatically retrieving songs aired by different radio stations in the United States. By crawling them in the period from December 2010 to May 2011, he managed to obtain $2,840,553$ transitions. Even if *Yes.com* is no longer active, the playlist dataset is publicly available.[12]

Yes.com does not include the timestamps, but only the playlists. Therefore, we have assumed that each playlist represents a sequence, as defined in our evaluation framework. In this case, it is not necessary to apply Algorithm 1 because the sequences are already available in the dataset in an explicit form. Since a timestamp-based splitting is not feasible, we have selected, for this dataset, a random splitting protocol. Furthermore, since we do not have any information regarding the radio stations, it is necessary to consider the playlists as if they were created by the same user. This approximation is acceptable in the context of sequence recommendation and it is allowed by the evaluation framework. In fact, differently from traditional evaluation approaches, all the metrics we propose are averaged over the sequences and not over the users. Finally, because of the computational complexity of the task, we have randomly reduced the complete dataset 10 times its original size and we have pruned the songs that appear less than 50 times.

The Yes.com and Foursquare datasets are characterized by a different distribution of their items, i.e. songs and venue categories, as it can be observed from Figure 5.3. In particular, Foursquare contains few items that are extremely popular, while Yes.com presents a plot that is more smooth. This conclusion is numerically supported by the values of entropy [223] obtained for the two distributions, which are 4.95 for Foursquare and 6.75 for Yes.com.

Table 5.1 summarizes the number of users, items, ratings, and sequences available in these datasets. For what concerns the Foursquare, these stats and the experimental results reported in Sec. 5.4 refer to the data collected from October to December 2017. The collection of data from Foursquare is ongoing and, in a successive work,

[11] https://bit.ly/2RvLjag

[12] https://www.cs.cornell.edu/~shuochen/lme/data_page.html

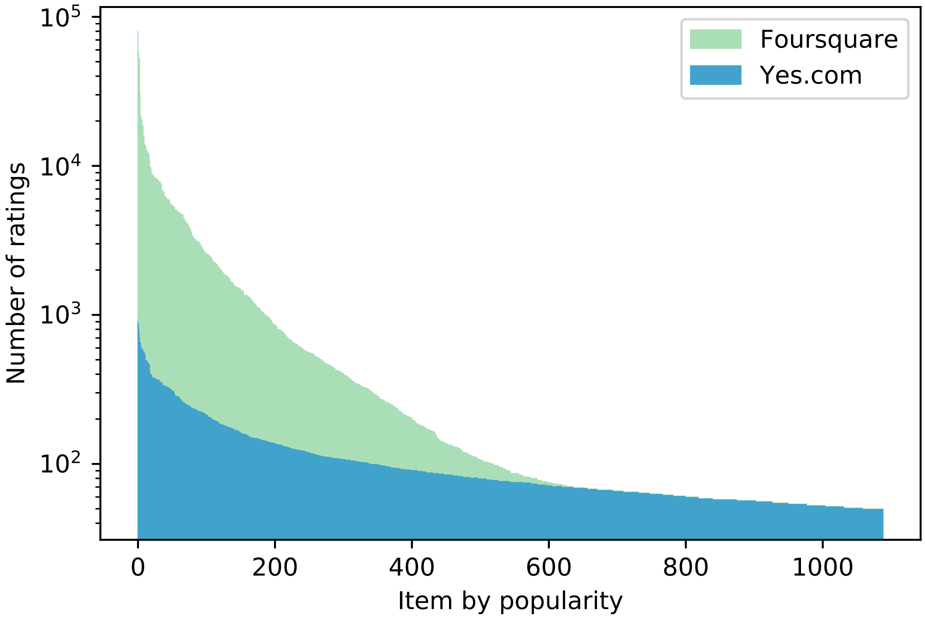


Fig. 5.3 A stacked barplot with a logarithmic scale representing the number of ratings for each item. Note the different shapes of their long-tail distributions: it is possible to observe that Foursquare has more popular items than Yes.com. Note that this also due to the fact that Foursquare has fewer items in absolute terms compared to the Yes.com dataset.

Dataset	$\lvert\mathscr{U}\rvert$	$\lvert\mathscr{I}\rvert$	$\lvert\mathscr{R}\rvert$	$\lvert\mathscr{S}\rvert$
Yes.com	1	1,089	118,022	10,551
Foursquare	44,319	651	1,047,429	400,261

Table 5.1 The number of users, items, ratings, and sequences for each dataset.

we have mapped the datasets to Schema.org categories and to Wikidata entities. This has given birth to the Semantic Trails Datasets [32].

5.4 Path Recommender Results

In this section, we address RQ3.2: *How do deep learning methods perform compared to other more traditional modelling approaches in the generation of tourist paths?* We rely on Sequeval to compare the RNN models with a set of competing systems. Table 5.2 lists the results obtained by the RNN recommender in the tourist domain. In this case, the most popular recommender system accounted for the highest precision, meaning that the top-5 items are extremely common (Fig. 5.3), but, as usual, its coverage is very limited and it achieved the lowest novelty. As expected, the random recommender scored the lowest precision, and the highest coverage and novelty. The differences among the unigram, the bigram, and the CRF recommenders are evident: the unigram accounted for higher precision because of the strong popularity bias of the dataset, while the bigram for the lowest perplexity. The RNN recommender system obtained the second best precision and perplexity, resulting in a good compromise if we are interested in optimizing both these metrics. Most importantly, RNN obtains the best serendipity, showing that it is able to generate accurate, but non-obvious, recommendations.

The value of the RNN with respect to competing systems is even more evident on the Yes.com dataset, whose popularity bias is weaker. Table 5.3 summarizes the figures of the comparison conducted with Yes.com. The Most Popular (MP) recommender achieved a fair precision, but at the price of a very low coverage, because its predictions are deterministic. Unsurprisingly, the lowest precision, and the highest novelty and diversity are associated with the random recommender. In

	MP	Random	Unigram	Bigram	CRF	**RNN**
Coverage	0.0077	1.0000	0.9616	1.0000	0.9677	0.5069
Precision	0.2259	0.0080	0.0774	0.0607	0.0754	0.0962
nDPM	0.4998	0.5000	0.4994	0.4998	0.4993	0.4991
Diversity	0.9194	0.9971	0.9616	0.9777	0.9621	0.9469
Novelty	4.6056	12.300	7.1421	9.0216	7.3710	6.8374
Serendipity	0.0000	0.0060	0.0256	0.0230	0.0252	**0.0365**
Confidence	1.0000	0.0015	0.0171	0.0140	0.0179	0.0264
Perplexity	$+\infty$	651.00	141.41	122.99	147.49	140.39

Table 5.2 Overview of the results of the baselines and both CRF and RNN with Foursquare.

	MP	Random	Unigram	Bigram	CRF	**RNN**
Coverage	0.0046	1.0000	0.9945	1.0000	0.9991	0.9458
Precision	0.0503	0.0090	0.0127	0.0103	0.0190	0.0782
nDPM	0.5007	0.5000	0.5000	0.5000	0.5000	0.4986
Diversity	0.6925	0.9900	0.9815	0.9854	0.9788	0.9052
Novelty	7.2383	10.380	9.7349	10.315	9.8449	9.5762
Serendipity	0.0000	0.0089	0.0107	0.0095	0.0179	**0.0706**
Confidence	1.0000	0.0009	0.0016	0.0011	0.0020	0.0123
Perplexity	$+\infty$	1089.0	848.96	637.53	747.33	183.49

Table 5.3 Overview of the results of the baselines and both CRF and RNN with Yes.com.

contrast, the unigram, the bigram, and the CRF recommenders obtained comparable scores of precision, but the bigram is the most appealing of these three techniques, because of its lower perplexity and higher novelty.

We can observe that the RNN recommender achieved the highest precision and the lowest perplexity, resulting to be the most promising algorithm for a future online experimentation. Its nDPM is slightly lower than 0.5, meaning that the items are usually predicted in the correct order. We can also observe that its serendipity is close to the value of precision: for this reason, it is possible to assume that the majority of the sequences are not obvious.

Given the promising results obtained by the Path Recommender also in the music domain, we extended its architecture in the participation to the RecSys2018 challenge (Sec. 5.5).

5.5 From Tourist Paths to Music Playlists: the RecSys2018 challenge

In recent years, music streaming services strongly modified the way in which people access to music content. In particular, the music experience does not foresee anymore to follow pre-defined collections of tracks (albums) edited by music stakeholders (artists and labels): the end-user can now produce her/his own playlist with potentially unlimited freedom. As a consequence, the automatic playlist generation and continuation are now crucial tasks in the recommender system field. This section describes our results for the task of automated music playlist completion obtained in the context of the RecSys Challenge 2018 [139] and addressed the RQ3.3: *How can we extend the Path Recommender to deal with the automated music playlist continuation task?*.

The model used is an extension of the Path Recommender to deal with specifical of the challenge of the challenge and the music domain.

5.5.1 The RecSys2018 Challenge

The RecSys2018 challenge has been sponsored by Spotify and dealt with the *automatic music playlist continuation* task [139]. This task consists of adding a number of tracks to a music playlist in a way that fits the original playlist. Participants had to devise algorithms that predict, for a given playlist, an ordered list of 500 recommended candidate tracks.

A specific dataset has been released by Spotify for the challenge: the Million Playlist Dataset (MPD), which includes, for each playlist, its title and the list of tracks contained in it (including album and artist names), plus some additional metadata such as Spotify URIs and the playlist's number of followers. The evaluation of the submissions was based on three metrics. The *R-precision* measures the fraction of recommended relevant items among all known relevant items, and it does not depend on the order of the tracks. The *clicks* metric is tied to a Spotify's feature that recommends 10 tracks and measures the number of refreshes that a user should do to find the first relevant track. The third metric is the Non-Discounted Cumulative Gain

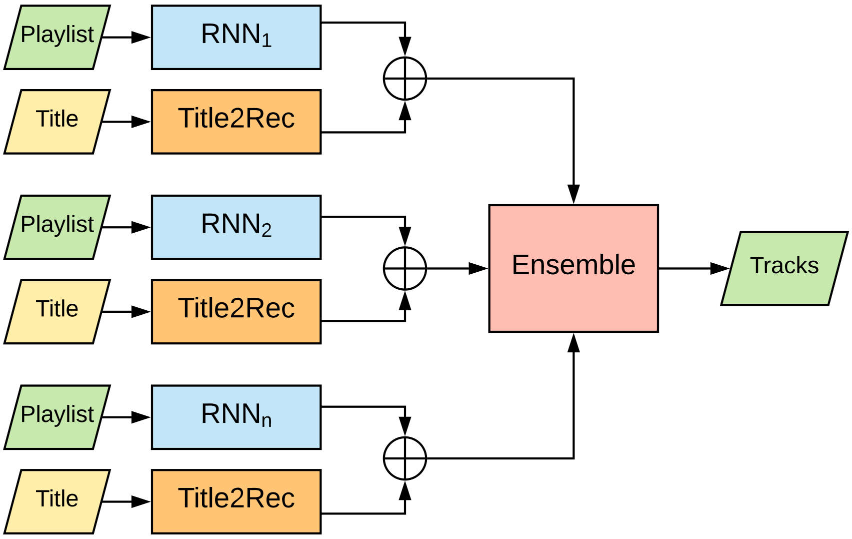


Fig. 5.4 The proposed ensemble architecture for playlist completion. The inputs are a playlist and its title.

(NDCG), a standard information retrieval metric commonly used to measure ranking quality [155].

The challenge has two tracks: in the *main track* only the MPD data can be used, in the *creative track* the MPD data can be enriched with external data sources (e.g. song lyrics).

5.5.2 Ensemble

Our approach builds upon an ensemble voting strategy of different runs of multiple Recurrent Neural Networks (RNNs) and one execution of Title2Rec (Sec. 5.5.4). The RNNs are configured differently in terms of network inputs and hyper-parameters. The RNNs are used to predict the missing tracks to be part of a playlist and thus assume to have seed(s) track(s) of the playlist to be utilized as initial elements of the network bootstrap (Section 5.5.3). However, when only the title of the playlist is available, our approach relies on a fall-back strategy that implements a K-means clustering of the playlists and a word embedding model of their titles (trained with fastText), called Title2Rec (Section 5.5.4). Figure 5.4 illustrates the overall approach.

The ensemble weighs the rankings of the different runs by giving more importance (more weights) to the top ranked tracks and less to the low ranked tracks, similarly to a Borda count election.[13] In detail, given a ranked set of predictions coming from a configuration k, corresponding to a particular configuration of the RNN jointly combined with Title2Rec, $R_k = \{T_1, T_2, \dots, T_{500}\}$, we assign to each track a score s_k that has its maximum for the first track in the ranking and minimum for the last one, i.e. $s_k(T_i) = 500 - i + 1$. Then, we sum the scores over all the configurations that we want to ensemble, obtaining a final score for each track $s(T_i) = \sum_k s_k(T_i)$ which we use to create the final ranking of the tracks. Take as an example (with 3 tracks instead of 500 in the predictions) a configuration 1 with ranking $R_1 = \{T_1, T_2, T_3\}$ and a configuration 2 with ranking $R_2 = \{T_1, T_3, T_2\}$. We would get $s_1(T_1) = 3$, $s_1(T_2) = 2$, $s_1(T_3) = 1$, $s_2(T_1) = 3$, $s_2(T_3) = 2$, $s_2(T_2) = 1$ and thus $s(T_1) = 3 + 3 = 6$, $s(T_2) = 2 + 1 = 3$, $s(T_3) = 1 + 2 = 3$, obtaining as a final ranking $R = \{T_1, T_2, T_3\}$, or equivalently $R = \{T_1, T_3, T_2\}$ as T_2 and T_3 have the same score.

5.5.3 Recurrent Neural Networks

We proceed by extending the Path Recommender approach (Sec. 5.2) to music playlist. We use RNNs, more specifically Long-Short Term Memory (LSTM) cells [142], in a similar vein to the language modeling problem, i.e. training the network to predict the next track in a playlist and sampling tracks from the learned probability model to generate predictions. In practice, rather than using only the track as input, we use a richer representation that also exploits the artist, the album, the title and, possibly, lyrics features (Figure 5.5).

In the following sections, we describe in detail the input features as well as the generation strategy.

Input Vectors

Track, Album and Artist Embeddings In order to leverage the information in the dataset concerning tracks, artists and albums, we opt for an approach based on word2vec [75] embeddings. More precisely, we train the word2vec model

[13] https://en.wikipedia.org/wiki/Borda_count

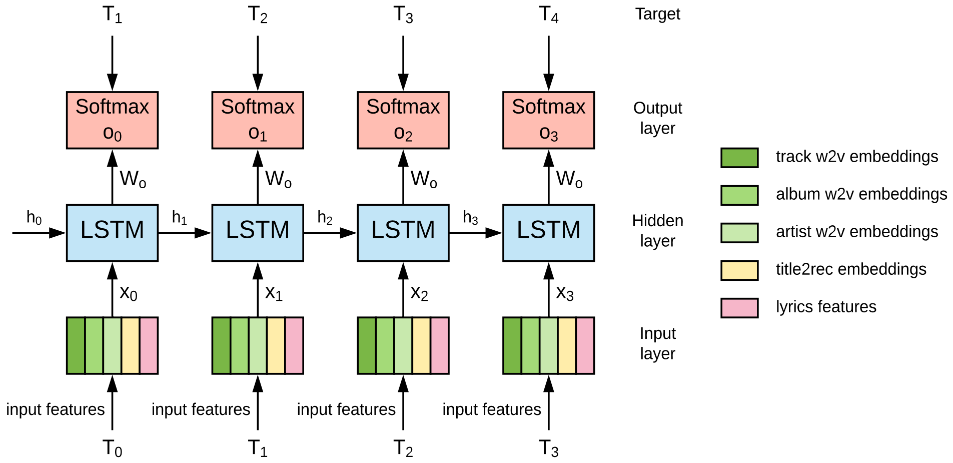


Fig. 5.5 RNN architecture for playlist completion. The input vectors include word2vec embeddings for the track, the album, and the artist, a fastText embedding for the playlist title and numerous features extracted from the lyrics.

separately on sequences of tracks, albums and artists in the order of appearance in the playlist, obtaining three separated word2vec models encoding co-occurrence patterns of tracks, albums and artists respectively. Each word2vec model is based on the Skip-gram model with negative sampling using default hyper-parameters of the Gensim implementation [224]: embedding vector dimension is $d = 100$, learning rate $\alpha = 0.025$ linearly decaying up to $min_\alpha = 0.0001$, window size $c = 5$, number of epochs is $\eta = 5$.

We concatenate the three representations of the tracks, albums and artists, obtaining an input vector x_{w2v} whose dimensionality is $|x_{w2v}| = 300$.

Titles Embeddings The title of a playlist can potentially contain interesting information about the intention and the purpose of its creator. The title can suggest that the tracks in certain playlist are intended to suit a certain goal (e.g. *party*, *workout*), a mood (*sad songs*, *relaxing*), a genre (*country*, *reggae*), or a topic (*90's*, *Christmas*). Our intuition, supported by the experiments described later in this section, is that playlists with similar titles may contain similar tracks. The title similarity could rely on pre-trained models and thesauri. However, we opted for computing a model that is specific for the playlist continuation task, using the sole data of the MPD.

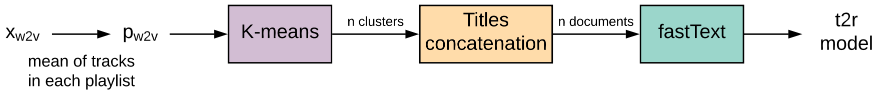


Fig. 5.6 Pipeline for generating the title embedding model used in Title2Rec. The embeddings are computed through a fastText model trained on a corpus of concatenated titles of similar playlists.

A playlist embedding p_{w2v} is computed as the mean of the embeddings of the tracks composing the playlist, already generated in Section 5.5.3. The playlist embeddings are then grouped in n clusters, applying the K-means algorithm.

We empirically observed that, apart from very general clusters, we also created clusters containing specialized playlists, obtaining as a consequence groups of titles that belong to the same semantic area. For example, a cluster contains playlists like *Christmas feels*, *December* or with titles including the emoji of Santa Claus, while another group encompasses playlists like *country* and *Alabama*.

Each cluster c expresses a composed label, which is the concatenation of the titles of all the playlist $p \in c$ separated by a blank space. These labels can be seen as a corpus of n documents (one for each cluster) that is used as input for the fastText algorithm [225]. Because this algorithm is able to represent textual information at the level of n-grams from 3 to 6 character, the Title2Rec model in output computes the embeddings of any playlist title, being this already seen in the dataset or totally unknown. Figure 5.6 illustrates the process of the Title2Rec model generation.

Lyrics Embeddings Since playlists contain tracks that share semantic properties (such as the genre) and acoustic properties (such as the mood), we hypothesize their lyrics share features as well. To this end, we extract numerous features from the lyrics for a large set of tracks used in the MPD dataset ($v \in \mathbb{R}^n$) that describe different stylistic and linguistic dimensions of a song text:

- *vocabulary* ($v \in \mathbb{R}$): as a measure of the vocabulary richness, we compute the type-token ratio of a song text.
- *style* ($v \in \mathbb{R}^{27}$): to estimate the linguistic style of a song text, we measure the line lengths (in characters and in tokens) and the frequencies of all major part-

of-speech tags. We further count rhyme occurrences and “echoisms” (sung words like “laaalala” and “yeeeeeeeaaaaaaah”).

- *semantics* ($v \in \mathbb{R}^{60}$): we build a topic model with 60 topics on the song text bag of words using Latent Dirichlet Allocation [226]. Each song text is then represented by its association to these topics.
- *orientation* ($v \in \mathbb{R}^{3}$): this dimension models how the song narrative (entities, events) is oriented with respect to the world. We encode a temporal dimension, i.e. whether the song mainly recounts past experiences or present/future ones, by representing the fraction of past tense verb forms to all verb forms as a feature.
- *emotion* ($v \in \mathbb{R}^{6}$): we model the subjectivity (subjective vs. objective) as well as the polarity (positive vs. negative) of the song text. Furthermore, the emotions conveyed are modelled in a common two-dimensional model that accounts for degrees of arousal and valence.
- *song structure* ($v \in \mathbb{R}^{4}$): as a proxy of the structure of the lyrics, we use the line lengths as well as the lengths of paragraphs in the song text.

For experimental purposes, we grouped the previous features in two additional categories:

- *deterministic* ($v \in \mathbb{R}^{23}$): it encompasses all features generated in a deterministic way such as features related to the structure, the vocabulary, and the style of the lyrics. We excluded from this group the frequencies of part-of-speech tags, as they depend on the tagger used.
- *fuzzy* ($v \in \mathbb{R}^{18}$): it includes the features generated in a non-deterministic fashion such as orientation, emotion, and the frequencies of part-of-speech tags.

All features are scaled using a custom feature scaler that combines two elements: i) account for outliers by scaling the data non-linearly based on the percentile of the feature value distribution they belong to; ii) scale the data linearly to the same $[-1,1]$ interval that non-lyrics features live in.

Retrieving lyrics for the MPD dataset is achieved by linking it to the WASABI corpus [227].[14] The WASABI corpus is an ongoing resource that contains 2.1M song texts (of 77k artists), and for each song it provides the following information: the lyrics extracted from http://lyrics.wikia.com, the synchronized lyrics (when available) from http://usdb.animux.de, DBpedia abstracts and categories the song belongs to, genre, label, writer, release date, awards, producers, artist and/or band members, the stereo audio track from Deezer (when available), the unmixed audio tracks of the song, its ISRC, BPM, and duration. In total, we linked 416k tracks in MPD (out of 2.2M unique tracks) to WASABI tracks that contain the lyrics. While the linked tracks proportion with $\sim$20% seems small, the linked tracks cover 53% of all 66M track occurrences in MPD because of the typical fat-tailed distribution, where some songs are extremely common while most titles occur only rarely in a playlist. Linking the lyrics was done in three levels of accuracy: direct Spotify URI matching gave us 155k links, exact artist and title matching provided 334k matches, and finally lower casing and deleting bracketed content (in song titles only) led to 51k matches. As the results overlap we ended up with 416k matched tracks in total. Some of our lyrics features are language-specific, so we decided to compute lyrics features exclusively on English song texts. This finally resulted in 367k English song texts we computed lyrical features on. Language detection is done with the *langdetect* package[15] and datasets of MPD and WASABI are merged along the axes of their Spotify URIs, artist names, song title names, respectively. Masking is used for all the songs for which a match has not been found and lyrics features could not be extracted.

Learning Model

As mentioned earlier, we address the problem of playlist continuation as a language modeling problem. More specifically, we train the RNN to predict the next track in a playlist, defining the targets Y to be the inputs X shifted in time, i.e. $X =$

[14] https://wasabi.i3s.unice.fr

[15] https://github.com/Mimino666/langdetect

$\{(\hat{T^j}_0, \hat{T^j}_1, \ldots, \hat{T^j}_{N_j-1})\}$ and $Y = \{(T^j{}_1, T^j{}_2, \ldots, T^j_{N_j})\}$ where $\hat{T}$ represents a track and its metadata (artist, album, playlist title, lyrics features), T represents a track id in a playlist, $j = 1, \ldots, M$ is a playlist index and N_j is the length of the j-*th* playlist. In this way, we train the model to learn a probability distribution of the next track $P(T_N | \hat{T}_{N-1}, \hat{T}_{N-2}, \ldots, \hat{T}_0)$ given the previous ones, which is parametrized by the network outputs that are converted into probabilities by the final softmax layer (Figure 5.5). The training algorithm attempts to minimize the cross-entropy loss function L, that measures the disagreement between the learned probability model and the observed probability model of the targets Y. The perplexity metric that is reported in the experiments (Section 5.5.5) corresponds to $ppl = 2^L$. In practice, rather than using probabilities, we use the 'logits' p_i where i is a track index, un-normalized scores that are proportional to the probabilities. Different optimization algorithms to minimize the loss are empirically compared (Section 5.5.5).

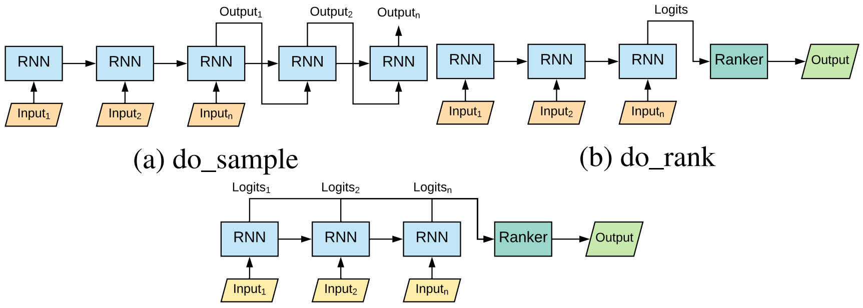
Generating predictions

We experiment three different strategies to generate track predictions from the RNN. Given an input seed and the hidden state, the trained model outputs the logits p_i, i.e. un-normalized scores that are proportional to the probability that a given track appears after the sequence of seeds s. In details, we considered the following approaches, as depicted in Figure 5.7.

do_sample It samples the track with the highest logit p_i, where $\hat{i} = arg\ max(p_i)$, given the set of seeds s. It adds the sampled track $\hat{i}$ to the seeds s, then it repeats the previous operations until 500 tracks are sampled.

do_rank It ranks the tracks according to their logit value p_i, given all the seeds s, then it selects the top-500 tracks with the highest logit.

do_summed_rank It computes the logits p_i for every seed. It averages all the logits in the sequence obtaining $\hat{p}_i$ and then it ranks the tracks according to the values of $\hat{p}_i$.



(a) do_sample

(b) do_rank

(c) do_summed_rank

Fig. 5.7 Three strategies for generating track predictions.

5.5.4 Title2Rec

In order to leverage the information contained in the title of the playlists, we propose a new approach called Title2Rec. Title2Rec recommends tracks taking as input the playlist title, following the procedure illustrated in Figure 5.8. The title is translated into a vector p_{t2r}, named title embedding, computed by applying the strategy described in Section 5.5.3 to the playlists defined in the MPD dataset.

Given a new seed playlist, we compute its title embedding in the same way. Then, we select a subset P including the top-300 most similar playlists to the given one by comparing its embeddings with p_{t2r} using the cosine similarity. Finally, the required number of tracks are selected among the ones available in P. The tracks have been ordered to ensure that the most popular ones in P are placed at the top of the list.

5.5.5 Optimization

In the following sections, we describe the empirical evaluations conducted with the purpose of optimizing the configuration of the RNN, Title2Rec, and the ensemble.

RNN Optimization

For optimizing the hyper-parameters of the RNN, we executed a grid search on a down-sampled version of the MPD dataset containing 100,000 playlists. We considered the following parameters:

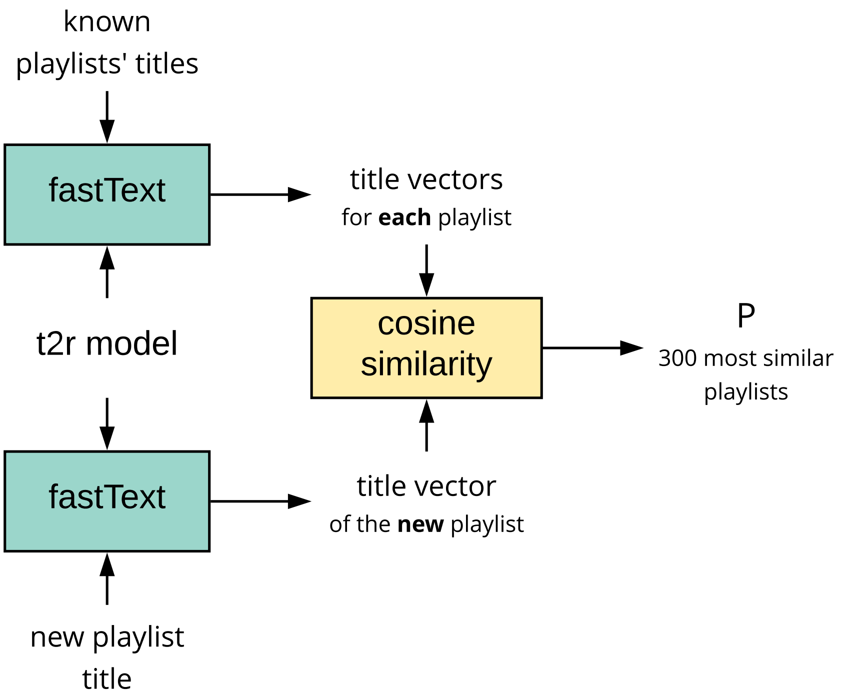


Fig. 5.8 The Title2Rec algorithm compares the fastText representation of the title of a seed playlist to the known ones using the cosine similarity.

- optimizer: $opt = \{Gradient, RMSProp, ADAM\}$
- learning rate: $lr = \{1, 0.5, 0.1, 0.01\}$
- number of steps: $ns = \{10, 20\}$
- hidden layer size: $hl = \{50, 100\}$

For each configuration (opt, lr, ns, hl), we trained the RNN model and we measured its perplexity on a validation set consisting of 1,000 playlists. Furthermore, we measured its R-Precision, NDCG, and Click metrics as defined in the challenge rules on a separate test set of the same size. The validation and test sets used for optimization purposes contain playlists with the first 5 tracks available as the initial seed, while the others are hidden.

We considered a total of 48 possible configurations: the values of perplexity of the most significant ones are reported in Table 5.4. Perplexity measures the ‘surprise’ of the probabilistic model in observing the data and it is defined as s^L where L is the

Optimizer	L.R.	Steps	Hidden	ppl	Time	R-Prec.
ADAM	1	20	100	1357.04	3:29	0.1739
ADAM	1	10	100	1482.86	3:39	0.1742
Gradient	1	10	100	1693.96	3:32	0.1566
ADAM	1	10	50	1716.92	2:30	0.1745
Gradient	1	10	50	2005.54	2:25	0.1543

Table 5.4 The results of the most significant RNN models. ‘L.R.’ stands for learning rate, ‘Steps’ for the number of time steps, ‘Hidden’ for the size of the hidden layer, ‘ppl’ stands for perplexity, ‘Time’ is the training time in hours:minutes.

cross-entropy loss function. Thus, lower values of perplexity corresponds to better models. We observe that, when the hidden size is fixed, the best performing optimizer is ADAM. Furthermore, increasing the number of steps reduces the perplexity of the RNN, but it does not have a significant effect on the R-Prec.

Finally, because of time constrains, we selected the configuration $(ADAM, 1, 10, 50)$ as the optimal one, despite its higher perplexity: in fact, we empirically observed that a smaller hidden size results in a shorter training duration.

We evaluated in a controlled setting all the strategies for generating the recommended tracks described in Section 5.5.3. We observed that, independently from other hyper-parameters, the technique called *do_summed_rank* systematically achieved better results than the other ones in all the metrics considered. For this reason, we selected this algorithm as our track generation strategy.

Finally, we analyzed the effects on the evaluation metrics of the different categories of features extracted from the lyrics as defined in Section 5.5.3, and we selected the groups emotion and fuzzy as the most performing ones.

Title2Rec Optimization

In order to improve the performances of Title2Rec, we worked on different parts of the pipeline. Each optimization has been tested by running the algorithm on a validation set of 1,000 playlists. Then, only the edits that improved the scores with respect to the non-optimized version have been kept in the final version.

We applied a pre-processing on each single title that performed a series of tasks:

- lowercasing;
- detecting and separating emoji from words;
- separating the skin code from the emoji;
- detecting and separating emoticons from words;
- transforming space-separated single letters into words (e.g. "*w o r k o u t*" becomes "*workout*";
- remove '#' from hashtags.

Other tasks that have been tested with no improvements are:

- detecting and separating punctuation from words;
- removing stop words;
- removing all spaces.

The latter point has been partially exploited because we noticed an improvement in the results by including in the corpus both versions of the title – keeping the spaces (as in "green day") and removing them ("greenday").

Another optimization step included the usage of different parameters for executing the pipeline. The clustering phase have been tested with different values of k (the number of clusters in output for the K-means algorithm). The value of 500 gives better results than smaller and bigger ones, which produce clusters that are respectively less specialized and less populated. The fastText training has been run with 5 epochs, a learning rate of 0.1 and different loss functions (ns, hs, *softmax*), window sizes (3, *5*, 10). The values in italics represent the best results.

The ordering by popularity described in Section 5.5.4 has been modified so that the impact of each playlist is proportional to the similarity of its title to the seed. In other words, a track has a higher chance to be recommended if it is included in a large number of playlists in P and if most of them are among the top ones more similar to the seed.

Finally, some improvements come from the inclusion of the playlist descriptions in the training. On the whole set of descriptions in the MPD dataset, we compute a TF-IDF model. Thanks to this, we are able to extract a set of keywords for each description by selecting the 3 words with the highest score. These keywords are added to the documents used to build the clusters. The contribution of the description is null when the playlist does not include any.

Ensemble Optimization

We studied the performance of the ensemble by applying a combination without repetition sampling of the different runs for each of the tracks, namely main and creative, and for different groups of runs. In detail, given n the total number of runs, and k the grouping factor, we devised a number of $\frac{n!}{k!(n-k!)}$, where we varied $k = 1, \ldots, n-1$. We then selected the best performing configuration for both the main and the creative tracks by optimizing the three metrics used for the final ranking. These configurations are reported in Section 5.5.6.

5.5.6 Experimental Results

In order to evaluate the effectiveness of our approach, we have divided the official MPD dataset in a training, a validation, and a test set. The validation and the test set contain 10,000 playlists each, that is the 1% of the original dataset. These playlists have been selected according to the characteristics of the MPD provided by Spotify.[16] Thus, the validation and test playlists are divided into 10 different categories: each of them defines a peculiar way of hiding some information during the testing phase, i.e. the number of seed tracks or their order.

Furthermore, we have implemented an evaluation tool that computes on our split the same metrics that are described in the challenge rules. Following this approach, it is possible to inspect the evaluation results for each category of the test set separately.

[16] https://recsys-challenge.spotify.com/challenge_readme

Approach	Optimizer	Epoch	R-Prec.	NDCG	Click
Title2Rec	-	-	0.0837	0.1260	12.007
Word2Rec	-	-	0.0963	0.1444	8.4322
RNN 300	Gradient	1	0.1417	0.1621	4.1902
RNN 300	Gradient	2	0.1500	0.1656	3.9433
RNN 300	ADAM	1	0.1557	0.1702	3.9213
RNN 300	ADAM	2	0.1457	0.1672	4.4224
RNN 400	ADAM	1	0.1572	0.1708	3.9340
RNN 400	ADAM	2	0.1520	0.1694	4.1307
RNN Emotion	ADAM	1	0.1556	0.1702	4.0101
RNN Emotion	ADAM	2	0.1500	0.1680	4.3594
RNN Fuzzy	ADAM	1	0.1555	0.1698	3.9950
RNN Fuzzy	ADAM	2	0.1503	0.1683	4.3456

Table 5.5 Results of different approaches on our test set

As expected, the category containing playlists with only their title and no tracks proved to be the most difficult one to address.

Table 5.5 contains the results obtained on our test set by Title2Rec, Word2Rec, and the RNNs trained with different optimizers and input vectors. Word2Rec corresponds to the word2vec model trained on sequences of tracks as described in Section 5.5.3 and used to generate predictions directly by looking up the 500 most similar tracks to the seeds. All the neural models, but the first two, were trained with the optimal configuration described in Section 5.5.5. These models are computationally demanding: the training phase lasted more than three days per epoch. The numbers 300 and 400 represent the dimensionality of the input vectors: the 300 models were trained without the title embeddings, while the 400 ones also exploit the fastText model described in Section 5.5.3. All the RNNs that include the features extracted from the lyrics were trained with input vectors of dimensionality higher than 400.

Table 5.6 lists the results computed on our test set for the best performing configurations in the two tracks of the challenge. The models combined in the ensemble are the following:

Track	R-Precision	NDCG	Click
Main	0.1611	0.1710	3.6349
Creative	0.1634	0.1717	3.5964

Table 5.6 Results of the ensemble on our test set

Main track RNN 300 (Gradient; Epoch 1 and 2), RNN 300 (ADAM; Epoch 1 and 2), and RNN 400 (Epoch 1 and 2).

Creative track RNN 300 (Gradient; Epoch 1 and 2), RNN 300 (ADAM; Epoch 1), RNN 400 (Epoch 1 and 2), RNN Emotion (Epoch 1 and 2), and RNN Fuzzy (Epoch 1 and 2).

We achieve the 14th position out of 33 participants in the creative track and the 36th position out of 113 participants in the main track.

5.6 Summary

In this chapter, we have introduced the Path Recommender, a RNN-based recommender systems that learns to generate tourist paths from Location-Based Social Network data, such as Foursquare check-ins. We have described and defined the problem of recommending sequences, first, in general, and then with specific reference to the tourist use-case. Then, we have described how the RNN model works, providing the details of the model's architecture. We have introduced Sequeval, an evaluation framework that includes protocols, metrics, and baselines for the evaluation of SARS, and described the process of data collection from Foursquare. Finally, we have discussed the extension of the Path Recommender model to the music playlist continuation task in the RecSys2018 challenge.

The experimental results have shown that the Path Recommender, and more in general RNNs, generates accurate and non-obvious recommendations with respect to competing algorithms. This is in line with the general trend observed in machine learning research, where neural networks are taking over as the most powerful and accurate methods to learn from data. Also, as shown in the RecSys2018 challenge experiment, their structure is sufficiently flexible to include heterogeneous features coming from different data sources (e.g. text). The analogy on which we have based the work of this chapter is a powerful one. Seeing sequence-aware recommendation as a language modelling problem leads to an easy application of all the recent breakthroughs in language modelling (e.g. transformer-based architectures such as BERT [228]) to the sequence recommendation problem.

In the evaluation process, we have observed that also for sequence-aware recommendations the popularity bias can be a strong factor that influences the results and needs to be properly addressed using novelty-aware metrics. Sequeval aims to help in this by providing a set of metrics that go beyond accuracy and that are specifically conceived for sequence-aware recommendations.

Chapter 6

Conclusions

In this thesis, we have addressed a set of research challenges in the fields of Recommender Systems, Semantics, and, more in general, Artificial Intelligence.

Two big families of approaches can be identified in AI history: a *deductive approach* where intelligence is hard-coded into machines as sophisticated rule-based algorithms that allow machines to make logical inferences and perform tasks, an *inductive approach* where intelligence arises as a pattern recognition process from a set of empirical observations (data). In recent years, the tremendous growth of data and of computing power, together with the advancements in the machine learning field (Deep Neural Networks especially) have strongly pushed the research towards inductive, data-driven approaches, which have accomplished tasks that were not possible before. Semantics, on the other hand, has historically been closer to deductive approaches, in the way it attempts to classifies and structure data according to well defined logically and semantically coherent structure.

This thesis shows that these two approaches actually *strongly benefit from each other*. Knowledge graphs are a very valuable source of data for machine learning algorithms to perform prediction tasks as, for instance, as we show in Chapter 1, they are able to enhance quality of recommendations and make them more explainable and transparent. At the same time, machine learning algorithms can be very useful in the process of generation of a knowledge graph, for instance, for the task of entity matching, as we have shown in Chapter 4. We also show that, in the end, they can serve the same purpose of supporting the decision-making process, either by providing well structured information in the form of knowledge graphs (see the 3cixty use-case of Chapter 4), or by providing personalized suggestions through recommender systems (Chapter 3 and Chapter 5).

We conclude our work by first reporting a summary of the findings of our research, in the form of answers to the RQs presented in Chap. 1, and, then, by pointing at current gaps of the work we presented, describing how future work can possibly address them.

6.1 Summary

The first chapter of the thesis deals with knowledge graph embeddings for recommender systems:

RQ1 *How can knowledge graph embeddings be used to create accurate, non-obvious and semantics-aware recommendations based on both collaborative and content-based filtering?*

The first step that is common to all approaches is the knowledge graph model. As we have discussed in Chapter 2, knowledge graphs are popular nowadays for their ability to model heterogeneous entities and relations. The key insight in the recommender system domain is that users can be included in the graph together with their appreciation for specific items, modelled by a special property, which we call 'feedback'. This enables collaborative filtering, as user liking patterns can be mined to make recommendations. Also, relations between items and other entities (e.g. starring actors of a movie) are included in the model, making possible content-based filters. Based on this model, we have introduced a number of approaches to make recommendations by creating knowledge graph embeddings.

Translational Models: translational models are popular algorithms to predict missing properties in a knowledge graph. We have shown that these models (TransE [85], TransH [86], TransR [87]) can be easily applied to make recommendations. The key ingredient is that the appreciation of an item by a particular user is modeled as the ‘feedback’ property. All the nodes and the properties of the graph are embedded in the vector space. Then, the recommendation problem is addressed as a link prediction problem of the ‘feedback’ property. Our experiments have shown that translational models are competitive with state-of-the-art collaborative filtering systems in terms of accuracy. However, neighborhood-based graph embedding methods such as node2vec and entity2rec are generally more effective.

node2vec: node2vec [78] is a state-of-the-art algorithm for graph embeddings. It has built upon existing algorithms based on neural language models (e.g. DeepWalk [77]) introducing a new definition of node neighborhood that is more flexible and adaptable to the topology of real work networks. We have shown that node2vec can be applied directly on the knowledge graph model, embedding all entities into vectors. The recommendation problem is then addressed retrieving the closest items to users in the vector space. node2vec generates very accurate recommendations, generally outperforming collaborative filtering and translational models. node2vec, however, does not preserve the semantics of the knowledge graph, as properties are ignored. To address this gap, we introduce entity2rec.

entity2rec: entity2rec generates property-specific knowledge graph embeddings for item recommendation. entity2rec can be seen as an extension of node2vec to knowledge graphs for the recommendation problem. Its property-specific approach allows to improve the quality of the recommendations and to encode the semantics of the properties in the recommendation model. We have performed several experiments to define how property-specific subgraphs should be built and how the algorithm should be configured, showing that hybrid property-specific subgraphs significantly enhance the recommendation quality and that when using hybrid property-specific subgraphs and a proper configuration of the hyper-parameters (long walks, many walks per entity, large context size), the learning to rank is no longer beneficial. Our experiments show that entity2rec generates high-quality recommendations, especially for datasets characterized by a high sparsity and a lower popularity bias. This is the typical case where the recommendation problem is challenging, since little information is known about the user and non-personalized algorithms based on item popularity are ineffective. The recommender model of entity2rec preserves the semantics of the knowledge graph, as it averages a set of property-specific relatedness scores. This linear recommendation function could be easily modified according to weights entered by a user in an interactive interface. Furthermore, property-specific relatedness scores can be leveraged to generate rich explanations of the recommendations, in terms of related past items, item properties and item content. More about this aspect is discussed in the Future Work.

Tinderbook: we have tested entity2rec in an online scenario by developing and publishing the Tinderbook application. In Tinderbook, property-specific knowledge graph embeddings built through entity2rec are used to create an item relatedness function, which is leveraged to retrieve the closest items to the one provided by the user in the onboarding phase. This setup shows that entity2rec can be used in a cold-start scenario, by using item-item similarities rather than user-item similarities as in the original formulation. The offline evaluation shows that entity2rec, within this experimental setting, outperforms baselines such as ItemKNN and MostPop, but also a content-based recommender based on KG embeddings built using RDF2Vec. Also, we have received good feedback from users in terms of usability and usefulness.

The second chapter of the thesis deals with improving the accuracy (precision and recall) of the entity matching in the process of a knowledge graph generation. Specifically, we have addressed the following research question (Chapter 4):

RQ2 *Can ensemble learning algorithms such as stacked generalization improve the performance of threshold-based classifiers in the entity matching process?*

STEM: entity matching is often based on threshold-based classifiers, i.e. binary classifiers that predicts that a pair of records is a match if their confidence is above a specific threshold. This decision threshold is typically manually configured, attempting to find a trade-off between precision and recall. To break this trade-off and improve the F-score, we introduce STEM (Stacked Threshold-based Entity

Matching), a machine learning layer that can be stacked upon any threshold-based classifier. STEM uses the predictions of different decision thresholds as features of a supervised classifier, which learns to combine them to output a final decision. We have performed experimental tests on three datasets belonging to different domains and using two different threshold-based classifiers, one based on a linear model [19] and one based on a Naive Bayes [186]. The results show that, in all cases, STEM improves the performance of the threshold-based classifier, up to 43% of F-score. We have assessed the F-score of STEM varying the amount of available training data, comparing it with that of a set of supervised classifiers directly trained on property-specific similarity values computed between pairs of records. The experiments show that STEM consistently performs better with respect to competitors and is less sensitive to the amount of training data.

3cixty: in the process of generation of the 3cixty knowledge graph for the municipality of Nice, STEM has been applied for matching places and events coming from a number of local and global data providers. In this context, it has shown to be scalable and fast enough to adapt to a real use-case and to outperform simple threshold-based classifiers.

Finally, in this thesis, we have worked on the problem of sequence-aware recommendation, through the following research question:

RQ3 *How can we create a recommender system that learns to recommend tourist paths from Location-Based Social Network data, effectively leveraging the temporal correlation among tourist activities?*

Path Recommender: often we have wondered ‘where should I go next?’ in the exploration of a city. Personalized suggestions of city tours are extremely valuable for a tourist, and lots of research has recently focused on generating sequence of tourist activities in a data-driven way. To accomplish this goal, we have collected data from Foursquare, a popular Location-based Social Network Data where users can publish information about their activities in a city. We have extracted sequences of tourist activities, so called ‘paths’ or ‘trails’, which we release in the open source Semantic Trails Dataset [32]. This dataset has been used to train the Path Recommender, a sequence-aware recommender systems based on RNNs, to predict tourist paths. We have used a simple architecture based on GRU [229] cells, which has shown to outperform competing systems such as bigram models or Conditional Random Fields (CRF) [230] for the problem of generating tourist paths. Furthermore, the Path Recommender has been used in a European project called PasTime to create an application that support tourists in exploring a city providing personalized tour recommendations.

Sequeval: in the process of benchmarking and evaluating the Path Recommender, we have realized that there was a substantial gap of metrics and evaluation protocols for sequence-aware recommender systems. Thus, we have created Sequeval, an evaluation framework for sequence-aware recommender systems. Sequeval extends a set of metrics that have originally been proposed for traditional recommender systems, considering a number of dimensions that go beyond pure accuracy, such as novelty, diversity and serendipity. Sequeval has been tested with different sequence-aware recommender systems and different datasets, is an open source project, publicly available on Github.

RecSys2018 challenge: we have extended the Path Recommender architecture to address a similar sequence prediction problem, i.e. playlist continuation in the context of the RecSys2018 challenge [139]. We have used an RNN based on LSTM cells [142], with a set of ad-hoc modifications for the playlist continuation problem. The RNN can include, together with the sequence of songs, a variety of external information extracted from song lyrics and from song titles. The RNN has achieved the 14th position out of 33 participants in the creative track and the 36th position out of 113 participants in the main track.

6.2 Future work

In this section, we discuss some of the gaps and opportunities for future work concerning what we have presented in this thesis.

entity2rec

Streaming: currently, entity2rec does not support streaming data. This means that whenever new data is added to the knowledge graph, embeddings have to be regenerated from scratch. A future work is to extend the algorithm so that the embeddings can be updated whenever new feedback comes in. Given the local nature of the random walk exploration, this could be accomplished by performing a set of random walks only in the neighborhood of the newly added nodes in the graph, similarly to what described in DeepWalk [77]. Future experiments should clarify whether this approximation performs sufficiently well and what are the most suitable configurations of the learning algorithm for this setup.

Explanations: several times we have mentioned that entity2rec can in principle generate rich explanations of recommended items, given that it uses a knowledge graph model that includes a lot of information, such as a related entities to a specific item. For instance, by comparing the values of $\rho_p(u,i)$ we might understand which specific aspect of the item i is closer to the user u. For instance, if we saw that $\rho_{director}(u,i) > \rho_p(u,i)$ for any p, we might conclude that the director is the aspect of the movie that is closer to the user u, and generate as an explanation: 'Since you like this director...'. This approach to generate explanations is hard to test in an offline setting, but it could be tested in an online experiment using a simple A/B testing protocol.

Personalized entity linking: using knowledge graph embeddings, it is easy to measure the relatedness between a user and a specific entity. Given the vector representations of u and e, a similarity function can be used to obtain $\rho(u,e)$. Given that entity relatedness is a crucial ingredient for entity linking algorithms [231], $\rho(u,e)$ could be used to personalize an entity linking algorithm, giving higher priorities to entities that are particularly relevant for a *specific user*.

Additional information about user-item interaction: we have modelled user-item interactions as simple unweighted edges in a graph, showing whether the user liked an item or not. For the Movielens and the LibraryThing dataset, we have binarized ratings data using a threshold, and for LastFM we have modelled every interaction as a positive feedback. However, the number of times a user listened to a given artist in LastFM is a valuable information and should be leveraged by the recommendation model. As a future work, it is possible to include this information as a set of weights on the 'feedback' edges of the knowledge graph. These weights are taken into account in the random walk exploration, which will then affect the embeddings and the recommendation function.

The case of explicit feedback, such as ratings, is slightly more complicated, as negative ratings are not to be considered as connections at all. A possible solution would be to only use positive ratings as an edge, attaching their value as a weight, and neglecting negative ratings. A more sophisticated approach would require a different loss function for learning the embeddings, such as a pairwise ranking loss function, where negative ratings can be considered as negative examples.

Similarity functions in entity2rec: we have run a large set of experiments for entity2rec, searching for optimal hyper-parameters, as well as comparing different aggregation functions. However, we have always used the cosine similarity function to retrieve recommended items in the embedding space. Given that many similarity functions exist, and that they could have a major impact on the set of recommended items, as a future work it is possible to experiment with new similarity functions within entity2rec.

Tinderbook

Improve data quality: DBpedia has allowed to create the knowledge graph for the recommendation algorithm, connecting books through links to common entities and complementing the collaborative information coming from LibraryThing ratings with content-based information. Furthermore, DBpedia has enabled to obtain rich book descriptions (e.g. abstract), without the cost of creating, curating and maintaining a book database. The multilinguality of DBpedia will also be a great advantage should we extend Tinderbook to other languages in the future. On the other hand, using DBpedia data has some pitfalls. The first one is the data loss during the mapping, as only 11,694 out of a total of 37,231 books (31.409%) in the LibraryThing dataset are mapped to DBpedia entities[1]. The second one is that, in some cases, the information in DBpedia resulted to be inaccurate. For example, during some preliminary tests, we have noticed that in many cases the thumbnail reported in the 'dbo:thumbnail' property is far from ideal to represent accurately the book (see Jurassic Park novel[2]), and we had to rely on Google to find better book covers. As a future work, we will experiment with different knowledge graphs such as Wikidata [40] to see whether this can limit the data loss and the overall quality of the metadata provided in the recommendations.

Diversity and novelty: users typically gave positive feedback about Tinderbook, saying that it was fun to use ("cool! it's fun!") and that recommendations were accurate ("It gave me all of my favorite books!"). Some users, though, complained that the recommendations lacked diversity (e.g. "I got all recommendations from the same author"), or novelty ("recommendations are a bit too classic"). As a short term solution, we have added a rule that prevents to have more than two books from the same author of the seed book in the recommended list. As a future work, though, we will try to improve other dimensions of the recommendation quality such as the diversity and the novelty, as most of the work so far has been done in optimizing the accuracy of the recommendations in an offline setting. This can be accomplished applying hard constraints (e.g. removing top K popular items from the recommendations) or through a soft constraint, for instance by adding a term in the loss function that encourages diverse and novel recommendations in the solution of the minimization problem.

Translational models

Multi-type user-item interactions: in this work, we have always dealt with simple user-item interactions, such as a like or a rating for a movie, which we modelled with the 'feedback' property. However, in some recommendation problems, it can be useful to model multiple types of interactions. Consider the example of ads. A user might close the ads, watch it, click on the advertised link or even purchase the product. These interactions have a different meaning and could be modelled as

[1] https://github.com/sisinflab/LODrecsys-datasets/tree/master/LibraryThing

[2] http://dbpedia.org/page/Jurassic_Park_(novel)

different properties (e.g. 'discard', 'watch', 'click', 'buy'), showing an increasing degree of interest in the item. Translational models are naturally meant to predict multiple properties, as they are conceived to perform link prediction on knowledge graphs, and could be easily applied to this type of problem.

STEM

Stacking: in STEM we have only experimented with an SVM classifier, but any classifier can in principle be used for the stacking layer. Future experimental work should clarify whether the advantage brought by stacking to threshold-based classifier can be even more significant with other machine learning algorithms (e.g. random forests, neural networks...). Furthermore, more classifiers could be stacked or ensembled together, and stacking could be applied to other parameters of the algorithms.

Parallel implementation: STEM is computationally more expensive than a single threshold-based classifier, as it involves running several instances of the threshold-based classifier and then training a supervised learner on top of their matching decisions. Among these two components of the total STEM runtime, we observe that for the datasets that we have used in this work, the first is the most expensive step. Thus, as a future work, we plan to improve the computing time of the software using a parallel and/or distributed implementation to allow the simultaneous execution of processes.

Path Recommender

Instance recommendation: the Path Recommender generates sequences of venue categories such as 'Art Museum'. Then, a logic needs to select a specific instance (e.g. 'The Museum of Modern Art') of that category that is suitable for the user. As a future work, we plan to implement a logic that selects instances from venue categories based on the user context (geographic position, time of the day, weather, etc...) and the personal user history.

Knowledge and sequence-aware recommendations: in this manuscript, we have considered recommender systems based on knowledge graphs and on sequence-aware algorithms, but we have not discussed the intersection between these two. For the datasets used for the Path Recommender evaluation, no knowledge graph was available to enhance recommendations using techniques discussed in Chapter 3 and thus we have relied only on collaborative information, i.e. user checkins. However, The use of knowledge graph embeddings in sequence-aware recommendations is, to the best of our knowledge, a highly promising and highly unexplored research area. Hence, in the future, we plan to explore the effect of features extracted using knowledge graph embeddings in a sequence-aware recommender system.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[2] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[3] Enrico Palumbo, Giuseppe Rizzo, and Raphaël Troncy. Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 32–36. ACM, 2017.

[4] Barry Schwartz. The paradox of choice: Why more is less. Ecco New York, 2004.

[5] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13, 2016.

[6] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[7] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, pages 76–80, 2003.

[8] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[9] John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. A proposal for the dartmouth summer research project on artificial intelligence (1955). *Reprinted online at http://www-formal. stanford. edu/jmc/history/-dartmouth/dartmouth. html*, 1996.

[10] Marvin Lee Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.

[11] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[12] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

[13] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[14] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[15] Rose Catherine, Kathryn Mazaitis, Maxine Eskenazi, and William Cohen. Explainable entity-based recommendations with knowledge graphs. *arXiv preprint arXiv:1707.05254*, 2017.

[16] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[17] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2013.

[18] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[19] Robert Isele, Anja Jentzsch, and Christian Bizer. Silk server-adding missing links while consuming linked data. In *Proceedings of the First International Conference on Consuming Linked Data-Volume 665*, pages 85–96. CEUR-WS. org, 2010.

[20] Enrico Palumbo, Giuseppe Rizzo, and Raphael Tröncy. An ensemble approach to financial entity matching for the feiii 2016 challenge. In *DSMM'16: Proceedings of the Second International Workshop on Data Science for Macro-Modeling*. ACM, 2016.

[21] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[22] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[23] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[24] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Computing Surveys*, 51(4):1–36, jul 2018.

[25] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. An empirical comparison of knowledge graph embeddings for item recommendation. In *DL4KGS@ ESWC*, pages 14–20, 2018.

[26] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. Translational models for item recommendation. In *European Semantic Web Conference*, pages 478–490. Springer, 2018.

[27] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. Knowledge graph embeddings with node2vec for item recommendation. In *European Semantic Web Conference*, pages 117–120. Springer, 2018.

[28] Enrico Palumbo, Giuseppe Rizzo, and Raphaël Troncy. Stem: Stacked threshold-based entity matching for knowledge base generation. *Semantic Web*, 10:117–137, 2018.

[29] Raphael Troncy, Giuseppe Rizzo, Anthony Jameson, Oscar Corcho, Julien Plu, Enrico Palumbo, Juan Carlos Ballesteros Hermida, Adrian Spirescu, Kai-Dominik Kuhn, Catalin Barbu, et al. 3cixty: Building comprehensive knowledge bases for city exploration. *Web Semantics: Science, Services and Agents on the World Wide Web*, 46:2–13, 2017.

[30] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, and Elena Baralis. Predicting your next stop-over from location-based social network data with recurrent neural networks, 2017.

[31] Diego Monti, Enrico Palumbo, Giuseppe Rizzo, and Maurizio Morisio. Sequeval: An offline evaluation framework for sequence-based recommender systems. *Information*, 10(5):174, 2019.

[32] Diego Monti, Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, and Maurizio Morisio. Semantic trails of city explorations: How do we live a city. *arXiv preprint arXiv:1812.04367*, 2018.

[33] Diego Monti, Enrico Palumbo, Giuseppe Rizzo, Pasquale Lisena, Raphaël Troncy, Michael Fell, Elena Cabrio, and Maurizio Morisio. An ensemble approach of recurrent neural networks using pre-trained embeddings for playlist completion. In *Proceedings of the ACM Recommender Systems Challenge 2018*, page 13. ACM, 2018.

[34] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[35] John F Sowa. Semantic networks. *Encyclopedia of Cognitive Science*, 2006.

[36] Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5, 2012.

[37] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.

[38] Ora Lassila and Ralph R Swick. Resource description framework (rdf) model and syntax specification. 1999.

[39] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[40] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[41] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[42] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 85–92. ACM, 2013.

[43] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[44] William W Cohen, Henry Kautz, and David McAllester. Hardening soft information sources. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 255–259. ACM, 2000.

[45] Howard B Newcombe and James M Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.

[46] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal—The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[47] Andreas Thor and Erhard Rahm. Moma-a mapping-based object matching system. In *CIDR*, pages 247–258, 2007.

[48] Luís Leitão, Pável Calado, and Melanie Weis. Structure-based inference of xml similarity for fuzzy duplicate detection. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 293–302. ACM, 2007.

[49] Mikhail Bilenko and Raymond J Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[50] Sheila Tejada, Craig A Knoblock, and Steven Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.

[51] Mohamed G Elfeky, Vassilios S Verykios, and Ahmed K Elmagarmid. Tailor: A record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 17–28. IEEE, 2002.

[52] Hanna Köpcke and Erhard Rahm. Training selection for tuning entity matching. In *QDB/MUD*, pages 3–12, 2008.

[53] Huimin Zhao and Sudha Ram. Entity identification for heterogeneous database integration—a multiple classifier system approach and empirical evaluation. *Information Systems*, 30(2):119–132, 2005.

[54] Sheila Tejada, Craig A Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 350–359. ACM, 2002.

[55] Tommaso Soru and Axel-Cyrille Ngonga Ngomo. A comparison of supervised learning classifiers for link discovery. In *10th International Conference on Semantic Systems*, pages 41–44, 2014.

[56] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[57] Zhaoqi Chen, Dmitri V Kalashnikov, and Sharad Mehrotra. Exploiting context analysis for combining multiple entity resolution systems. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 207–218. ACM, 2009.

[58] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *International semantic web conference*, pages 519–534. Springer, 2014.

[59] Giuseppe Rizzo, Marieke van Erp, and Raphaël Troncy. Benchmarking the extraction and disambiguation of named entities on the semantic web. In *LREC*, pages 4593–4600, 2014.

[60] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. Adoption of the linked data best practices in different topical domains. In *International Semantic Web Conference*, pages 245–260. Springer, 2014.

[61] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web*, (Preprint):1–18, 2015.

[62] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *International Semantic Web Conference*, pages 650–665. Springer, 2009.

[63] Robert Isele and Christian Bizer. Learning linkage rules using genetic programming. In *Proceedings of the 6th International Conference on Ontology Matching-Volume 814*, pages 13–24. CEUR-WS. org, 2011.

[64] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes-a time-efficient approach for large-scale link discovery on the web of data. *integration*, 15:3, 2011.

[65] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *Extended Semantic Web Conference*, pages 149–163. Springer, 2012.

[66] Mohamed Ahmed Sherif, Axel-Cyrille Ngonga Ngomo, and Jens Lehmann. Wombat–a generalization approach for automatic link discovery. In *European Semantic Web Conference*, pages 103–119. Springer, 2017.

[67] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock. Unsupervised entity resolution on multi-type graphs. In *International Semantic Web Conference*, pages 649–667. Springer, 2016.

[68] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In *Advances in neural information processing systems*, pages 1401–1408, 2002.

[69] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer, and Hermann De Meer. idmesh: graph-based disambiguation of linked data. In *Proceedings of the 18th international conference on World wide web*, pages 591–600. ACM, 2009.

[70] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96. ACM, 2005.

[71] Mustafa Al-Bakri, Manuel Atencia, Jérôme David, Steffen Lalande, and Marie-Christine Rousset. Uncertainty-sensitive reasoning for inferring sameas facts in linked data. In *22nd european conference on artificial intelligence (ECAI)*, pages 698–706. IOS press, 2016.

[72] Fatiha Saıs, Nathalie Pernelle, and Marie-Christine Rousset. L2r: A logical method for reference reconciliation. In *Proc. AAAI*, pages 329–334, 2007.

[73] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[74] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[75] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[76] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[77] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[78] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[79] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[80] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[81] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080, 2016.

[82] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013.

[83] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347, 2017.

[84] Maximilian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. *arXiv preprint arXiv:1806.03417*, 2018.

[85] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[86] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, volume 14, pages 1112–1119, 2014.

[87] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187, 2015.

[88] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. Rdf2vec: Rdf graph embeddings and their applications. *Semantic Web*, (Preprint):1–32, 2018.

[89] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[90] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

[91] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

[92] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[93] Cataldo Musto, Tiziano Franza, Giovanni Semeraro, Marco de Gemmis, and Pasquale Lops. Deep content-based recommender systems exploiting recurrent neural networks and linked open data. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, UMAP '18, pages 239–244, New York, NY, USA, 2018. ACM.

[94] Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. Linked open data to support content-based recommender systems. In *Proceedings of the 8th International Conference on Semantic Systems*, pages 1–8. ACM, 2012.

[95] Jessica Rosati, Petar Ristoski, Tommaso Di Noia, Renato de Leone, and Heiko Paulheim. Rdf graph embeddings for content-based recommender systems. In *CEUR workshop proceedings*, volume 1673, pages 23–30. RWTH, 2016.

[96] Cataldo Musto, Giovanni Semeraro, Marco de Gemmis, and Pasquale Lops. Learning word embeddings from wikipedia for content-based recommender systems. In Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane

Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, and Gianmaria Silvello, editors, *Advances in Information Retrieval*, pages 729–734, Cham, 2016. Springer International Publishing.

[97] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[98] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[99] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[100] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[101] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015.

[102] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[103] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 497–506. IEEE, 2011.

[104] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[105] Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review*, 13(5-6):393–408, 1999.

[106] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. *Aaai/iaai*, 23:187–192, 2002.

[107] Ian Soboroff and Charles Nicholas. Combining content and collaboration in text filtering. In *Proceedings of the IJCAI*, volume 99, pages 86–91. sn, 1999.

[108] Asela Gunawardana and Christopher Meek. A unified approach to building hybrid recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 117–124. ACM, 2009.

[109] Asim Ansari, Skander Essegaier, and Rajeev Kohli. Internet recommendation systems, 2000.

[110] Steffen Rendle. *Context-aware ranking with factorization models*. Springer, 2011.

[111] Xia Ning and George Karypis. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 155–162, 2012.

[112] Cristhian Figueroa, Iacopo Vagliano, Oscar Rodríguez Rocha, and Maurizio Morisio. A systematic literature review of linked data-based recommender systems. *Concurrency and Computation: Practice and Experience*, 27(17):4659–4684, 2015.

[113] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[114] Houda Khrouf and Raphaël Troncy. Hybrid event recommendation using linked data and user diversity. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 185–192. ACM, 2013.

[115] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 283–292. ACM, 2014.

[116] Rose Catherine and William Cohen. Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 325–332. ACM, 2016.

[117] Tommaso Di Noia, Vito Claudio Ostuni, Paolo Tomeo, and Eugenio Di Sciascio. Sprank: Semantic path-based ranking for top-n recommendations using linked open data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):9, 2016.

[118] Cataldo Musto, Fedelucio Narducci, Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Explod: A framework for explaining recommendations based on the linked open data cloud. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 151–154, New York, NY, USA, 2016. ACM.

[119] Vito Bellini, Angelo Schiavone, Tommaso Di Noia, Azzurra Ragone, and Eugenio Di Sciascio. Knowledge-aware autoencoders for explainable recommender systems. In *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, DLRS 2018, pages 24–31, New York, NY, USA, 2018. ACM.

[120] Weizhi Ma, Min Zhang, Yue Cao, Woojeong Jin, Chenyang Wang, Yiqun Liu, Shaoping Ma, and Xiang Ren. Jointly learning explainable rules for

recommendation with knowledge graph. In *The World Wide Web Conference*, pages 1210–1221. ACM, 2019.

[121] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 353–362. ACM, 2016.

[122] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 297–305, New York, NY, USA, 2018. ACM.

[123] Cataldo Musto, Pierpaolo Basile, and Giovanni Semeraro. Embedding knowledge graphs for semantics-aware recommendations based on dbpedia. In *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization*, UMAP'19 Adjunct, pages 27–31, New York, NY, USA, 2019. ACM.

[124] Chuan Shi, Binbin Hu, Xin Zhao, and Philip Yu. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[125] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 312–321. ACM, 2008.

[126] Yanhua Li, Moritz Steiner, Limin Wang, Zhi-Li Zhang, and Jie Bao. Exploring venue popularity in foursquare. In *INFOCOM, 2013 Proceedings IEEE*, pages 3357–3362. IEEE, 2013.

[127] Giuseppe Rizzo, Rosa Meo, Ruggero G Pensa, Giacomo Falcone, and Raphaël Troncy. Shaping city neighborhoods leveraging crowd sensors. *Information Systems*, 64:368–378, 2017.

[128] Anastasios Noulas, Salvatore Scellato, Cecilia Mascolo, and Massimiliano Pontil. Exploiting semantic annotations for clustering geographic areas and users in location-based social networks. *The Social Mobile Web*, 11(2), 2011.

[129] Daniel Preoţiuc-Pietro, Justin Cranshaw, and Tae Yano. Exploring venue-based city-to-city similarity measures. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing*, page 16. ACM, 2013.

[130] Bin Liu and Hui Xiong. Point-of-interest recommendation in location based social networks with topic and location awareness. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 396–404. SIAM, 2013.

[131] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. Where you like to go next: Successive point-of-interest recommendation. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2605–2611. AAAI, 2013.

[132] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. Personalized ranking metric embedding for next new poi recommendation. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 2069–2075. AAAI Press, 2015.

[133] Jihang Ye, Zhe Zhu, and Hong Cheng. What's your next move: User activity prediction in location-based social networks. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 171–179. SIAM, 2013.

[134] Jing He, Xin Li, Lejian Liao, Dandan Song, and William K Cheung. Inferring a personalized next point-of-interest recommendation model with latent behavior patterns. In *AAAI*, pages 137–143, 2016.

[135] Munmun De Choudhury, Moran Feldman, Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel, and Cong Yu. Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, pages 35–44. ACM, 2010.

[136] S Kotiloglu, T Lappas, K Pelechrinis, and PP Repoussis. Personalized multi-period tour recommendations. *Tourism Management*, 62:76–88, 2017.

[137] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[138] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 714–722. ACM Press, 2012.

[139] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys challenge 2018: Automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 527–528. ACM, 2018.

[140] Hamed Zamani, Markus Schedl, Paul Lamere, and Ching-Wei Chen. An analysis of approaches taken in the acm recsys challenge 2018 for automatic music playlist continuation. *arXiv preprint arXiv:1810.01520*, 2018.

[141] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[142] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[143] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[144] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[145] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, pages 1422–1432, 2015.

[146] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[147] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[148] B. Zhou, S.C. Hui, and K. Chang. An intelligent recommender system using sequential web access patterns. In *IEEE Conference on Cybernetics and Intelligent Systems*, pages 393–398. IEEE, 2004.

[149] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[150] Charu C. Aggarwal. Mining discrete sequences. In *Data Mining*, chapter 15, pages 493–529. Springer International Publishing, 2015.

[151] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 811–820. ACM Press, 2010.

[152] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for next basket recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 403–412. ACM Press, 2015.

[153] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[154] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[155] Gobinda G Chowdhury. *Introduction to modern information retrieval*. Facet publishing, 2010.

[156] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 333–336. ACM, 2011.

[157] Harald Steck. Evaluation of recommendations: rating-prediction and ranking. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 213–220. ACM, 2013.

[158] Pablo Sánchez and Alejandro Bellogín. Attribute-based evaluation for recommender systems: incorporating user and item attributes in evaluation metrics. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 378–382. ACM, 2019.

[159] Dietmar Jannach, Lukas Lerche, Iman Kamehkhosh, and Michael Jugovac. What recommenders recommend: an analysis of recommendation biases and possible countermeasures. *User Modeling and User-Adapted Interaction*, 25(5):427–491, 2015.

[160] Alan Said and Alejandro Bellogín. Comparative recommender system evaluation. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 129–136. ACM Press, 2014.

[161] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[162] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[163] David Martin Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2011.

[164] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[165] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[166] Tommaso Di Noia. Recommender systems meet linked open data. In *International Conference on Web Engineering*, pages 620–623. Springer, 2016.

[167] Daniel Billsus and Michael J Pazzani. Learning collaborative information filters. In *Icml*, volume 98, pages 46–54, 1998.

[168] Marco de Gemmis, Pasquale Lops, Giovanni Semeraro, and Cataldo Musto. An investigation on the serendipity problem in recommender systems. *Information Processing & Management*, 51(5):695–717, 2015.

[169] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender Systems*, pages 257–260. ACM Press, 2010.

[170] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender Systems*, pages 109–116. ACM Press, 2011.

[171] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. MyMediaLite: A free recommender system library. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)*, 2011.

[172] Alan Said and Alejandro Bellogín. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 129–136. ACM, 2014.

[173] Gaby David and Carolina Cambre. Screened intimacies: Tinder and the swipe logic. *Social media+ society*, 2(2):2056305116641976, 2016.

[174] N Babich. Designing card-based user interfaces, 2016.

[175] Bernard L Welch. The generalization of student's problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.

[176] Haifa Alharthi, Diana Inkpen, and Stan Szpakowicz. A survey of book recommender systems. *Journal of Intelligent Information Systems*, 51(1):139–160, 2018.

[177] T Cai. The tinder effect: Swipe to kiss (keep it simple, stupid!), 2018.

[178] C Cousins. The complete guide to an effective card-style interface design, 2015.

[179] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[180] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[181] Vijay Raghavan, Peter Bollmann, and Gwang S Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems (TOIS)*, 7(3):205–229, 1989.

[182] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *KDD Workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[183] Robert Isele and Christian Bizer. Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23:2–15, 2013.

[184] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, pages 4–15. Springer, 1998.

[185] Irina Rish. An empirical study of the naive Bayes classifier. In *IJCAI Workshop on Empirical Methods in Artificial Intelligence*, pages 41–46, 2001.

[186] Ernest Croot. Bayesian spam filter. http://people.math.gatech.edu/~ecroot/bayesian_filtering.pdf - Last visited: 28 April 2016.

[187] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[188] Ion Androutsopoulos, John Koutsias, Konstantinos V Chandrinos, and Constantine D Spyropoulos. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *23rd International Conference on Research and development in Information Retrieval (SIGIR)*, pages 160–167, 2000.

[189] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes-which naive bayes? In *CEAS*, pages 27–28, 2006.

[190] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI Workshop on Learning for Text Categorization*, volume 752, pages 41–48, 1998.

[191] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–278. ACM, 2002.

[192] J Volz, C Bizer, M Gaedke, and G Silk Kobilarov. A link discovery framework for the web of data. In *2nd Workshop about Linked Data on the Web, Madrid, Spain*, 2009.

[193] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[194] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. *Large scale kernel machines*, 3(1):301–320, 2007.

[195] George Forman and Martin Scholz. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, 12(1):49–57, 2010.

[196] Gregory V. Bard. Spelling-error Tolerant, Order-independent Pass-phrases via the Damerau-levenshtein String-edit Distance Metric. In *5th Australasian Symposium on ACSW Frontiers*, 2007.

[197] Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *WebDB*, 2011.

[198] Manel Achichi, Michelle Cheatham, Zlatan Dragisic, Jérôme Euzenat, Daniel Faria, Alfio Ferrara, Giorgos Flouris, Irini Fundulaki, Ian Harrow, Valentina Ivanova, et al. Results of the ontology alignment evaluation initiative 2016. In *11th ISWC workshop on ontology matching (OM)*, pages 73–129. No commercial editor., 2016.

[199] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf - Last visited: 28 April 2016, 2003.

[200] G. Rizzo, R. Troncy, O. Corcho, A. Jameson, J. Plu, J.C. Ballesteros Hermida, A. Assaf, C. Barbu, A. Spirescu, K. Kuhn, I. Celino, R. Agarwal, C.K. Nguyen, A. Pathak, C. Scanu, M. Valla, T. Haaker, E.S. Verga, M. Rossi, and J.L. Redondo Garcia. 3cixty@Expo Milano 2015: Enabling Visitors to Explore a Smart City. In *14^{th} International Semantic Web Conference (ISWC), Semantic Web Challenge*, 2015.

[201] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[202] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[203] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[204] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[205] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[206] Asela Gunawardana and Guy Shani. Evaluating recommender systems. In *Recommender Systems Handbook*, chapter 8, pages 265–308. Springer US, 2 edition, 2015.

[207] Dimitris Paraschakis, Bengt J. Nilsson, and John Hollander. Comparative evaluation of top-n recommenders in e-commerce: An industrial perspective. In *IEEE 14th International Conference on Machine Learning and Applications*, pages 1024–1031. IEEE, 2015.

[208] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[209] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

[210] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 158–167. ACM Press, 2000.

[211] Y.Y. Yao. Measuring retrieval effectiveness based on user preference of documents. *Journal of the American Society for Information Science*, 46(2):133–145, 1995.

[212] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[213] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, pages 22–32. ACM Press, 2005.

[214] Tommaso Di Noia, Vito Claudio Ostuni, Jessica Rosati, Paolo Tomeo, and Eugenio Di Sciascio. An analysis of users' propensity toward diversity in recommendations. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 285–288. ACM Press, 2014.

[215] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, pages 241–250. ACM Press, 2000.

[216] Diego Monti, Enrico Palumbo, Giuseppe Rizzo, and Maurizio Morisio. Sequeval: A framework to assess and benchmark sequence-based recommender systems. In *Proceedings of the Workshop on Offline Evaluation for Recommender Systems at the 12th ACM Conference on Recommender Systems*, 2018.

[217] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[218] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[219] Dingqi Yang, Daqing Zhang, Zhiyong Yu, and Zhiwen Yu. Fine-grained preference-aware location search leveraging crowdsourced digital footprints from LBSNs. In *Proceedings of the 2013 ACM international joint conference on pervasive and ubiquitous computing*. ACM, 2013.

[220] Dingqi Yang, Daqing Zhang, Vincent W. Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142, 2015.

[221] Dingqi Yang, Daqing Zhang, and Bingqing Qu. Participatory cultural mapping based on collective behavior data in location-based social networks. *ACM Transactions on Intelligent Systems and Technology*, 7(3):1–23, 2016.

[222] Munmun De Choudhury, Moran Feldman, Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel, and Cong Yu. Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM conference on hypertext and hypermedia*. ACM, 2010.

[223] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[224] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[225] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[226] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[227] Gabriel Meseguer-Brocal, Geoffroy Peeters, Guillaume Pellerin, Michel Buffa, Elena Cabrio, Catherine Faron Zucker, Alain Giboin, Isabelle Mirbel, Romain Hennequin, Manuel Moussallam, Francesco Piccoli, and Thomas Fillon. WASABI: a Two Million Song Database Project with Audio and Cultural Metadata plus WebAudio enhanced Client Applications. In *Web Audio Conference 2017 – Collaborative Audio #WAC2017*, London, United Kingdom, August 2017. Queen Mary University of London.

[228] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[229] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[230] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[231] Diego Ceccarelli, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. Learning relatedness measures for entity linking. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 139–148. ACM, 2013.

[232] Tommaso Di Noia, Corrado Magarelli, Andrea Maurino, Matteo Palmonari, and Anisa Rula. Using ontology-based data summarization to develop semantics-aware recommender systems. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 128–144, Cham, 2018. Springer International Publishing.

[233] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. Personalized ranking for non-uniformly sampled items. In *Proceedings of KDD Cup 2011*, pages 231–247, 2012.

Appendix A

Item Modeling

In order to create the knowledge graph, we need an item model, i.e. we need to define the properties that describe an item type for the datasets used in this work. To this end, we start from the DBpedia Ontology, that defines properties for different item types[1]. However, a property selection strategy purely based on the schema of the data does not provide any guarantee on how frequently used are those properties in the data. Thus, we opt for an empirical approach where we count what are the DBpedia properties (“dbo:") most frequently used in DBpedia data to describe the items in the datasets. More specifically, for each item i in the dataset, we retrieve all triples (i, p, o) in DBpedia and we the frequency of occurrence of the properties p. Then, we sort the properties according to their frequency of occurrence and we select the first N so that the frequency of the N+1-th property is less than 50% of the previous one. In this way, we avoid to select a fixed number of properties and we rely on the actual frequency of occurrence to determine the cut-off. Finally, we add “dct:subject" to the set of properties, as it provides an extremely rich categorization of items, as done in previous work [3, 117]. We obtain: [“dbo:director", “dbo:starring", “dbo:distributor", “dbo:writer",“dbo:musicComposer", “dbo:producer", “dbo:cinematography", “dbo:editing", “dct:subject"] for Movielens 1M, [“dbo:genre”, “dbo:recordLabel”, “dbo:hometown”, “dbo:associatedBand”, “dbo:associatedMusicalArtist”, “dbo:birthPlace”, “dbo:bandMember”, “dbo:formerBandMember”, “dbo:occupation”,

[1] http://mappings.dbpedia.org/server/ontology/classes/

“dbo:instrument”, “dct:subject"] for LastFM and [“dbo:author", “dbo:publisher", “dbo:literaryGenre", “dbo:mediaType", “dbo:subsequentWork", “dbo:previousWork", “dbo:series", “dbo:country", “dbo:language", “dbo:coverArtist", “dct:subject"]. Recently, some works such as ABSTAT [232] have dealt with the problem of finding a selection of properties to create a knowledge graph that optimizes recommender systems accuracy. For example, for the datasets under analysis, ABSTAT property selection in the configuration: $k = 10$, *norep.AbsOccAvgS* is: [“dbo:director”, “dbo:starring”, “dbo:distributor”, “dbo:writer”, “dbo:musicComposer”, “dbo:producer”, “dbo:cinematography”, “dbo:music”, “dbo:language”, “dct:subject”] for Movielens 1M, [“dbo:genre”, “dbo:recordLabel”, “dbo:hometown”,“dbo:birthPlace”,“dbp:placeOfBirth”, “dbo:deathPlace”, “dbo:field”, “dbo:nationality”, “dbp:placeOfDeath”, “dct:subject”] for LastFM and [“dct:subject”, “dbo:author”, “dbo:publisher”, “dbo:literaryGenre”, “dbo:mediaType”, “dbo:country”, “dbo:language”, “dbo:series”, “dbo:nonFictionSubject”, “dbo:coverArtist”]. We have run an experiment comparing entity2rec on a KG built using our property selection and the ABSTAT selection for the LastFM dataset, which is the one where the properties are differing more, but the scores did not show a consistent improvement of the recommendation quality (Tab. A.1).

Property selection	System	P@5	R@5	SER@5	NOV@5
dbo + frequency (C3)	$entity2rec_{lambda}$	0.1852	0.1066	0.1512	10.101
dbo + frequency (C3)	$entity2rec_{avg}$	0.2062	0.1191	0.1682	10.379
dbo + frequency (C3)	$entity2rec_{min}$	0.2055	0.1191	0.1664	9.807
dbo + frequency (C3)	$entity2rec_{max}$	0.1693	0.0986	0.1423	10.243
ABSTAT (C3)	$entity2rec_{lambda}$	0.1799	0.1039	0.1419	10.343
ABSTAT (C3)	$entity2rec_{avg}$	0.1915	0.1102	0.1528	10.681
ABSTAT (C3)	$entity2rec_{min}$	0.2010	0.1162	0.1604	9.830
ABSTAT (C3)	$entity2rec_{max}$	0.1731	0.1003	0.1422	10.280
dbo + frequency (C1)	$entity2rec_{lambda}$	0.1745	0.1009	0.1405	11.227
dbo + frequency (C1)	$entity2rec_{avg}$	0.1505	0.0870	0.1182	12.267
dbo + frequency (C1)	$entity2rec_{min}$	0.1699	0.0981	0.1343	11.331
dbo + frequency (C1)	$entity2rec_{max}$	0.1295	0.0753	0.1037	10.537
ABSTAT (C1)	$entity2rec_{lambda}$	0.1750	0.1011	0.1433	11.251
ABSTAT (C1)	$entity2rec_{avg}$	0.1390	0.0808	0.1043	12.039
ABSTAT (C1)	$entity2rec_{min}$	0.1604	0.0926	0.1264	11.320
ABSTAT (C1)	$entity2rec_{max}$	0.1382	0.0800	0.1110	10.968

Table A.1 ABSTAT property selection and the heuristics used in this paper (“dbo + frequency") are compared on the LastFM dataset. The heuristics work best for $C3 = \{p : 4, q : 4, d : 200, l : 100, c : 60, n : 100\}$, ABSTAT selection works best for $C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

Appendix B

entity2rec scores

In this section, we report the extended results of the comparison of entity2rec with the state-of-the-art in tabular form. For completeness, we include also:

- SoftMarginRankingMF: a matrix factorization model for item prediction optimized for a soft margin (hinge) ranking loss [110].
- WeightedBPRMF: a weighted version of BPRMF with frequency adjusted sampling [233].

All scores can be considered as without error up to the digit reported in the tables, as the standard deviation is negligible.

System	P@5	R@5	SER@5	NOV@5
$entity2rec_{lambda}$ (C2)	0.2125	0.0967	0.1913	9.654
$entity2rec_{avg}$ (C2)	0.2372	**0.1045**	**0.2125**	9.577
$entity2rec_{min}$ (C2)	0.2198	0.0976	0.1946	9.466
$entity2rec_{max}$ (C2)	0.2206	0.0951	0.2038	10.046
node2vec (C2)	0.2313	0.0994	0.2119	9.675
TransE	0.2014	0.0791	0.1951	9.882
TransH	0.2001	0.0772	0.1920	9.768
TransR	0.1864	0.0731	0.1822	9.960
BPRMF	0.2150	0.0809	0.1829	9.303
BPRSLIM	0.2252	0.0961	0.1969	9.438
ItemKNN	0.2004	0.0758	0.1920	10.546
LeastSquareSLIM	0.2162	0.0832	0.1656	8.811
MostPopular	0.1446	0.0492	0.0647	8.429
SoftMarginRankingMF	0.1222	0.0405	0.1206	9.721
WeightedBPRMF	0.0945	0.0364	0.0900	**11.334**
WRMF	**0.2550**	0.0991	0.2117	8.910
RankingFM	0.2202	0.0737	0.1782	8.996

Table B.1 Results for the Movielens 1M dataset. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

System	**P@5**	**R@5**	**SER@5**	**NOV@5**
$entity2rec_{lambda}$ (C3)	0.1852	0.1066	0.1512	10.101
$entity2rec_{avg}$ (C3)	**0.2062**	**0.1191**	**0.1682**	10.379
$entity2rec_{min}$ (C3)	0.2055	0.1191	0.1664	9.807
$entity2rec_{max}$ (C3)	0.1693	0.0986	0.1423	10.243
node2vec (C3)	0.1994	0.1161	0.1665	10.337
TransE	0.1628	0.0935	0.1393	9.662
TransH	0.1549	0.0892	0.1347	9.791
TransR	0.1417	0.0812	0.1276	10.643
BPRMF	0.1254	0.0720	0.0798	7.976
BPRSLIM	0.1921	0.1106	0.1526	8.860
ItemKNN	0.1144	0.0652	0.0956	**13.116**
LeastSquareSLIM	0.1502	0.0868	0.1050	8.307
MostPopular	0.0764	0.0444	0.0231	7.307
SoftMarginRankingMF	0.0426	0.0244	0.0387	9.876
WeightedBPRMF	0.1067	0.0611	0.0958	10.452
WRMF	0.2003	0.1152	0.1598	8.305
RankingFM	0.1678	0.0963	0.1324	8.985

Table B.2 Results for the LastFM dataset. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

System	**P@5**	**R@5**	**SER@5**	**NOV@5**
$entity2rec_{lambda}$ (C4)	0.1271	0.0803	0.1229	12.469
$entity2rec_{avg}$ (C4)	0.1800	0.1072	0.1736	12.089
$entity2rec_{min}$ (C4)	**0.1831**	**0.1084**	**0.1757**	11.709
$entity2rec_{max}$ (C4)	0.1634	0.0984	0.1591	12.783
node2vec (C4)	0.1749	0.1046	0.1706	12.495
TransE	0.0972	0.0598	0.0944	11.370
TransH	0.1041	0.0634	0.1016	11.397
TransR	0.0774	0.0459	0.0748	11.474
BPRMF	0.0377	0.0262	0.0240	9.487
BPRSLIM	0.1136	0.1013	0.0982	9.853
ItemKNN	0.1225	0.1009	0.1177	**13.114**
LeastSquareSLIM	0.0722	0.0522	0.0565	10.131
MostPopular	0.0343	0.0256	0.0070	8.453
SoftMarginRankingMF	0.0204	0.0141	0.0183	10.313
WeightedBPRMF	0.0381	0.0326	0.0375	11.830
WRMF	0.0802	0.0620	0.0658	9.101
RankingFM	0.0840	0.0464	0.0778	10.021

Table B.3 Results for the LibraryThing dataset. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

Appendix C

List of publications

Journals

- **Palumbo, E.**, Monti, D., Rizzo, G., Troncy, R. and Baralis, E., 2020. entity2rec: Property-specific Knowledge Graph Embeddings for Item Recommendation. Expert Systems with Applications, p.113235.

- **Palumbo, E.**, Rizzo, G. and Troncy, R., 2019. STEM: Stacked threshold-based entity matching for knowledge base generation. Semantic Web, pp.1-21.

- Troncy, R., Rizzo, G., Jameson, A., Corcho, O., Plu, J., **Palumbo, E.**, Hermida, J.C.B., Spirescu, A., Kuhn, K.D., Barbu, C. and Rossi, M., 2017. 3cixty: Building comprehensive knowledge bases for city exploration. Journal of Web Semantics, 46, pp.2-13.

- Monti, D., **Palumbo, E.**, Rizzo, G. and Morisio, M., 2019. Sequeval: An Offline Evaluation Framework for Sequence-Based Recommender Systems. Information, 10(5), p.174.

Conference proceedings

- **Palumbo, E.**, Rizzo, G. and Troncy, R., 2017, August. Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems (pp. 32-36). ACM.

- **Palumbo, E.**, Buzio, A., Gaiardo, A., Rizzo, G., Troncy, R. and Baralis, E., 2019, June. Tinderbook: Fall in Love with Culture. In European Semantic Web Conference (pp. 590-605). Springer, Cham.
- **Palumbo, E.**, Rizzo, G., Troncy, R., Baralis, E., Osella, M. and Ferro, E., 2018, June. Translational Models for Item Recommendation. In European Semantic Web Conference (pp. 478-490) Satellite Events. Springer, Cham.

Workshop proceedings

- **Palumbo, E.**, Rizzo, G., Troncy, R. and Baralis, E., 2017. Predicting Your Next Stop-over from Location-based Social Network Data with Recurrent Neural Networks. In RecTour@ RecSys (pp. 1-8).
- **Palumbo, E.**, Rizzo, G. and Troncy, R., 2016, June. An ensemble approach to financial entity matching for the FEIII 2016 challenge. In Proceedings of the Second International Workshop on Data Science for Macro-Modeling (p. 14). ACM.
- **Palumbo, E.**, Rizzo, G., Troncy, R., Baralis, E., Osella, M. and Ferro, E., 2018, June. An Empirical Comparison of Knowledge Graph Embeddings for Item Recommendation. In DL4KGS@ ESWC (pp. 14-20).
- Monti, D., **Palumbo, E.**, Rizzo, G., Lisena, P., Troncy, R., Fell, M., Cabrio, E. and Morisio, M., 2018, October. An Ensemble Approach of Recurrent Neural Networks using Pre-Trained Embeddings for Playlist Completion. In Proceedings of the ACM Recommender Systems Challenge 2018 (p. 13). ACM.

Poster paper

- **Palumbo, E.**, Rizzo, G., Troncy, R., Baralis, E., Osella, M. and Ferro, E., 2018, June. Knowledge graph embeddings with node2vec for item recommendation. In European Semantic Web Conference (pp. 117-120). Springer, Cham. **(best poster paper award)**