POLITECNICO DI TORINO
Repository ISTITUZIONALE

Exact memory–constrained UPGMA for large scale speaker clustering

Original
Exact memory–constrained UPGMA for large scale speaker clustering / Cumani, S., Laface, P.. - In: PATTERN RECOGNITION. - ISSN 0031-3203. - ELETTRONICO. - 95:(2019), pp. 235-246. [10.1016/j.patcog.2019.06.018]

Availability:
This version is available at: 11583/2760412 since: 2019-11-26T15:33:50Z

Publisher:
Elsevier

Published
DOI:10.1016/j.patcog.2019.06.018

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2019. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.patcog.2019.06.018

(Article begins on next page)

28 November 2026

Exact memory–constrained UPGMA for large scale speaker clustering

Sandro Cumani*, Pietro Laface

Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy

Abstract

This work focuses on clustering large sets of utterances collected from an unknown number of speakers. Since the number of speakers is unknown, we focus on exact hierarchical agglomerative clustering, followed by automatic selection of the number of clusters. Exact hierarchical clustering of a large number of vectors, however, is a challenging task due to memory constraints, which make it ineffective or unfeasible for large datasets. We propose an exact memory–constrained and parallel implementation of average linkage clustering for large scale datasets, showing that its computational complexity is approximately $\mathcal{O}(N^2)$, but is much faster (up to 40 times in our experiments), than the Reciprocal Nearest Neighbor chain algorithm, which has $\mathcal{O}(N^2)$ complexity. We also propose a very fast silhouette computation procedure that, in linear time, determines the set of clusters. The computational efficiency of our approach is demonstrated on datasets including up to 4 million speaker vectors.

Keywords: Clustering, UPGMA, similarity measures, Reciprocal Nearest Neighbor, PLDA, PSVM, silhouette, cluster quality measures.

1. Introduction

Clustering is a widely used unsupervised machine learning technique for exploratory data analysis, which allows discovering groups of data items that are similar [1, 2]. Clustering has a large scope of application in speaker

*Corresponding author

Email addresses: sandro.cumani@polito.it (Sandro Cumani), pietro.laface@polito.it (Pietro Laface)

recognition. Diarization of meetings [3], segmentation of multi–speaker conversations [4, 5, 6], and adaptation of a speaker recognition system to a new domain [7] are successful examples of its usage. In these applications, the number of different speakers and turns is relatively small. Clustering of large number of speakers and speech segments is required, instead, by other applications. Examples of the latter are semi–supervised training of speaker models, and speaker indexing in broadcast news. Clustering can also be useful for the detection of serial fraudsters in speaker verification applications. This type of fraudsters challenge the system performing many calls with different usernames just using their own voice, rather than using more difficult techniques such as replying or synthesizing the target speaker voice.

In this work we focus on unsupervised clustering of a large set of utterances collected from an unknown and large number of speakers. Among the most widely used techniques for clustering, K–means [8] and spectral clustering [9, 10] require the number of clusters to be predefined, whereas mean-shift [11] relies on the selection of a bandwidth parameter, which cannot be easily selected, or which is computationally expensive to estimate for large datasets. Thus in this work we focused on Hierarchical Agglomerative Clustering (HAC), a popular clustering technique, which discovers hierarchical relations between sets of clusters. HAC has been used since long time in many different fields and applications, for example for vector quantization [12], for document clustering [13], or for creating phylogenetic trees of biological data [14].
We aim at obtaining an exact hierarchical agglomerative solution, rather than relying on approximate Nearest Neighbor (NN) clustering techniques, or on other solutions that take local clustering decisions, based on predefined parameters, without considering all current items and clusters. This allows us deferring the estimation of the number of clusters at the end of the procedure, exploiting an internal validity measure applied to each cluster set that can be extracted from the hierarchy.

In particular, we focus on a computational efficient and memory–constrained implementation of the so called Unweighted Pair Group Method with Arithmetic Mean (UPGMA), which progressively merges clusters according to their minimum average distance producing a dendrogram [15]. UPGMA, or average linkage clustering, has been preferred to the single and complete linkage because it gives better performance in terms of cluster and speaker purity [16]. This has been confirmed by our preliminary experiments not reported here due to space limitations.

Comparison with other clustering approaches, requiring predefined parameters, such as the number of clusters, or any kind of thresholds, is out of the scope of this paper.

In the following, to avoid the distinction between similarity scores and distances, we will mostly refer to similarity scores or simply to scores. The considerations that apply to similarity scores are, of course, also valid for distances or dissimilarity scores.

Exact UPGMA of a large number of vectors is a challenging task due to memory and computational constraints. The standard UPGMA algorithm receives as input a score matrix $\mathbf{S}$ over a set of N items, thus it requires $\mathcal{O}(N^2)$ memory, which is simply not available for large datasets. Its running time is dominated by the time required for the selection of the cluster–pairs. By keeping the entries of each row of $\mathbf{S}$ in a heap, UPGMA time complexity reduces to $\mathcal{O}(N^2 \log(N))$ [17, 18, 13]. It can be further reduced to $\mathcal{O}(N^2)$ [18] by using the Reciprocal Nearest Neighbor chain (RNN) algorithm [19, 20, 21], which, given a score matrix $\mathbf{S}$, iteratively builds a chain of nearest neighbor clusters until it finds the nearest neighbor pair.
However, if N is large, we can only provide to RNN the items vectors, rather than the score matrix that would require $\mathcal{O}(N^2)$ memory. This solves the memory problem, but since the search for the chain of the nearest neighbor clusters is sequential, RNN cannot exploit massive parallel computation. We will further comment on this, and propose an efficient implementation of RNN, after introducing our fast and memory–constrained UPGMA algorithm.

In this work we propose an exact and parallel memory–constrained UPGMA implementation for large scale speaker clustering. Our approach, which will be referred to in the following as k–best UPGMA or K–UPGMA, performs clustering in multiple iterations starting from the singleton clusters. At each iteration, we precompute all the pairwise scores between the current set of clusters. Precomputing blocks of scores in parallel by multiple threads using vectorized dot-products, is much more efficient than searching sequentially the nearest neighbor of a given cluster. We keep the k–best list of cluster scores, obtained by means of a quick–select algorithm that has linear time complexity even on the worst case [22]. An exact dendrogram is grown by merging a subset of clusters up to a given level implicitly imposed by the size of the k–best list of scores kept in memory. Thus, we trade, to some extent, computations for memory by recomputing at every iteration the subset of scores that did not contribute to the growth of the dendrogram

in the previous iteration. Since this subset rapidly shrinks at every iteration, we experimentally show that, for a large enough size of the k–best list, the overall complexity remains almost quadratic in the number of vectors in the dataset.
K–UPGMA is similar to the approach proposed in [14] because the clustering process is performed in multiple rounds, but in [14] the matrix of dissimilarities are computed and kept sorted on disk. This is reasonable only if the number of items is not large, and if the complexity of the dissimilarity computation is high, as it is possibly the case for clustering protein sequences. This does not fit our case study.
Hierarchical clustering, up to a predefined level of the dendrogram, is also proposed in [23], with the assumption that higher levels of the dendrogram do not convey interesting information. In this approach, clustering is not performed in multiple rounds because it is supposed that, given a predefined level of the dendrogram, it is possible to store the matrix of precomputed dissimilarities on the memory of different nodes of a cluster of computers. In our case, however, the higher levels of the dendrogram convey interesting information because, even if we cluster millions of utterances, the number of speakers might be limited.

All these approaches, including ours, have at least quadratic complexity. This is unavoidable because they use a global rather than a local clustering decisions strategy, i.e., they compute all the pairwise scores between the current clusters. However, thanks to efficient data structures and algorithms, and exploiting the possibility of parallel computation of the similarity scores between clusters, we show that K–UPGMA has approximately $\mathcal{O}(N^2)$ computational complexity, but is 40 times faster than RNN. Considering a real use-case, K–UPGMA is able to cluster 900K 400–dimensional speaker vectors in approximately 15 minutes running on a single machine.

Crucial for the clustering efficiency is the possibility of computing the average score between two clusters without evaluating the pairwise distances between all the items of the two clusters. This has been done for cosine or Euclidean scoring functions [24, 25] by associating to each cluster the so called "Clustering Features", i.e., the number of vectors in the cluster, their mean and variance. We extend the Clustering Features concept to a class of scoring functions commonly used in speaker recognition and in statistical classification. This is obtained by formalizing the scoring functions in terms of dot–products.

Finally, by definition, UPGMA associates to each cluster, represented by

a dendrogram entry, the average dissimilarity between its merged clusters. We show that this information can be used for obtaining approximate, but accurate, silhouette values [26, 27]. Using the Silhouette Width Criterion (SWC) [27] we can determine with good accuracy the set of clusters in the dataset, and evaluate their quality.

Summarizing, our novel contributions are:

- A fast, parallel, and exact implementation of UPGMA for large scale vector clustering.
- An efficient similarity computation method, based on an extension of the "Clustering Features" concept, for scoring functions commonly used in speaker recognition and statistical classification.
- A fast approximate Silhouette Width Criterion for the selection of the number of clusters.

The outline of the paper is as follows: Section 2 outlines our memory–constrained K–UPGMA. The details of the main data structures and the algorithm that we use are given in Section 3 together with the vector–valued functions that allow speeding up pairwise similarity score computation. Section 4 presents a detailed description of K–UPGMA. In Section 5 we give several examples of commonly used scoring functions. Section 6 introduces our fast technique for automatic selection of the number of clusters from the dendrogram produced by UPGMA. Our experimental results are illustrated in Section 7, in terms of running time of the algorithm, and of internal and external cluster purity measures. Conclusions are given in Section 8.

2. K–UPGMA: outline

The outline of K–UPGMA, ignoring for the moment the details that make it effective, is given in Algorithm 1. In order to fulfill memory constraints, K–UPGMA receives as input a set of N vectors, and computes all their pairwise similarity scores, but keeps in memory only the k–best scoring pairs. It then loops selecting the current best scoring cluster pair (C_i, C_j), merging their elements to cluster C_m, appending them to the dendrogram, and updating the k–best list until it is empty. The update is performed by eliminating from the k–best list the pairs that include either C_i or C_j, and inserting only the pairs (C_m, C_k) whose score $S(C_m, C_k)$ is better than the worst score S_w in the k-best list. Fast selection of the pairs, and efficient computation of these scores is detailed in Section 3.1. Once the k-best list is empty, these

Algorithm 1 K–UPGMA: outline

Input: A set of vectors $\mathcal{V}$, and the number k of scores that can been kept in memory.
Output: A cluster hierarchy $\mathcal{H}$ over $\mathcal{V}$.
Initialization: Initialize the cluster set C by defining a singleton cluster $C_i = \{i\}$ for every vector $i \in \mathcal{V}$.
while $|C| > 1$ **do**
  if k–best list is empty **then**
    Refill k–best list: Compute the pairwise scores for each cluster pair (C_i, C_j), with $C_i, C_j \in C$.
    Keep only the k–best scores. Keep also the current worst score S_w among the k–best.
  end if
  Cluster–pair selection: Get the best scoring cluster pair (C_i, C_j) from the k–best list.
  Cluster–pair merging: Merge its elements into cluster C_m.
  Reduce the k–best list by discarding every cluster pair that includes an element C_i or C_j.
  Set $C = (C \setminus \{C_i, C_j\}) \cup \{C_m\}$.
  Append the pair (C_i, C_j) and its score to hierarchy $\mathcal{H}$.
  Update: For all $C_k \subseteq C$, compute score $s_k = S(C_m, C_k)$ (see Section 3.1)
  if $s_k > S_w$ **then**
    Append it, and the cluster identity, to the k–best list.
  end if
end while

steps are iterated by refilling the list with the scores of the remaining cluster pairs.

3. Data structures and scoring functions

Before presenting the details of the K–UPGMA algorithm, let us introduce the main data structures and algorithms that we use. K–UPGMA computes the pairwise similarity scores between N input vectors, corresponding to the elements above the main diagonal of the score matrix. The computational complexity of this step is quadratic. This complexity is amortized by dividing the score matrix into square blocks of size B. A set of T threads, or processes, computes the pairwise scores for these blocks in parallel. Each thread computes one block of the score matrix, and selects the k–best scores

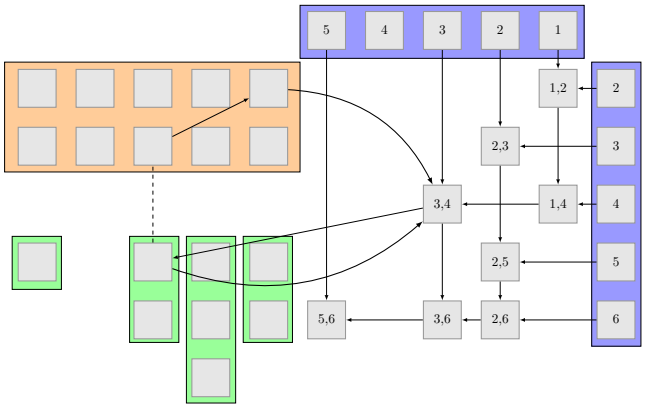

Figure 1: Heap arrays (green), BestHeap (orange), and Sparse matrix data structure.

of that block. After T blocks have been processed, the partial k–best scores are merged, and the resulting k–best scores are retained. This procedure continues until all blocks have been scored, and the overall k–best scores selected. For the sake of efficiency, each k–best list of cluster scores is obtained without sorting the scores computed by each thread. We use, instead, a quick–select algorithm, based on the median of medians selection (introselect), which has linear time complexity even on the worst case [22]. Furthermore, by keeping trace of the worst score in the k–best lists, most of the scores computed by the threads can be eliminated before performing the selection.
The amount of memory that is required by this software architecture is $\mathcal{O}(BT)$ for the scores, and $\mathcal{O}(kT)$ for storing the provisional k-best scores computed by each thread.

Fig. 1 illustrates the data structures that we use to perform the steps of Algorithm 1 without requiring expensive search operations. Once the k–best list of scores has been filled, we create for each entry in the list the corresponding entry in the sparse matrix, in the heap arrays, and in BestHeap array structures shown in figure as green and orange rectangles, respectively. In particular, since we consider only cluster pairs (C_i, C_j) with $j > i$, we devote a heap array for each cluster identifier C_i.
The BestHeap array stores the top element of each heap vector. In the following, we will refer collectively to these structures simply as "heap".
A heap array entry consists of a pointer, shown as a thick arrow, to an element of the sparse matrix.
A sparse matrix entry includes the cluster pair identifiers, C_i and C_j, their similarity score $S(C_i, C_j)$, and an index to the corresponding heap vector, shown as a thin arrow in Fig. 1. This index allows eliminating a single entry of a heap vector without search. The sparse matrix is implemented as two arrays of asymmetric double linked lists. The entries in each row (or column) are not sorted, they appear ordered in the figure just for the sake of clarity. Using this data structure it is easy discarding an entry, or an entire row or column of the sparse matrix, and inserting new cluster pairs both in the matrix and in the heap. Furthermore, when a row or column of the sparse matrix is discarded, it is also immediate to release the corresponding heap vector.
An entry of the BestHeap array consists of a structure including the pointer to an entry in the sparse matrix $[C_i, C_j, S(C_i, C_j)]$, and the index in the BestHeap array where the top value of the corresponding heap array is stored. This information is necessary to update the BestHeap array if an entry is removed from, or inserted at, the top of a heap array. Indeed, the new value at the top of a heap array must replace the old one in the BestHeap, possibly requiring reordering the BestHeap array.
In Fig. 1, cluster C_{34} is the current best scoring cluster because it is the top element of the BestHeap array, and of the third heap vector. The index for the current BestHeap value and the corresponding heap array are represented by the thin and dashed line, respectively.

Please notice that using a single or multiple heaps is a natural solution for speeding–up nearest neighbor search [17, 18, 12, 13, 28]. In our approach, their usage is tightly integrated with the k–best list management.

3.1. Heap management

In the "Update" step of the algorithm, it is necessary to compute the similarity of the new cluster with respect to all current clusters. Since this operation is essentially sequential, K–UPGMA would not perform better than the standard RNN approach without a strategy that is able to greatly reduce the number of the computed scores.

This section details the subset of the scores that are computed in the "Update" step of K–UPGMA Algorithm 1, how these scores are computed, and in which case they are inserted in the heap.

Let $\mathbf{x}_i$ be the i–th vector of a set $\mathcal{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. Let $S(\mathbf{x}_i, \mathbf{x}_j)$ be a scoring function for the pair of vectors $(\mathbf{x}_i, \mathbf{x}_j)$. Denoting by C_k the set of vectors in cluster k, the average pairwise score of all vectors in C_a and in C_b is:

$$S(C_a, C_b) = \frac{1}{N_a N_b} \sum_{i \in C_a} \sum_{j \in C_b} S(\mathbf{x}_i, \mathbf{x}_j) \ , \tag{1}$$

where N_a and N_b is the number of vectors in the corresponding cluster.
After merging clusters C_a and C_b into a new cluster C_m, K–UPGMA must remove all clusters (C_a, C_x) and (C_b, C_y) currently in the heap, and has to compute the average score between cluster C_m and every other cluster C_k in the heap. This score is the weighted-by-cardinality average of the scores of the merged clusters:

$$S(C_m, C_k) = \frac{1}{N_a + N_b} \left[N_a S(C_a, C_k) + N_b S(C_b, C_k) \right] \ . \tag{2}$$

According to the reducibility property [29], only a subset of the clusters in the heap must be considered. Let us illustrate the three possible scenarios induced by the set of clusters currently in the heap.

3.1.1. $S(C_a, C_k)$ and $S(C_b, C_k)$ are not in the heap

If both scores $S(C_a, C_k)$ and $S(C_b, C_k)$ are less than the worst score in the heap S_w, the score of the new cluster would be:

$$\begin{aligned} S(C_m, C_k) &= \frac{1}{N_a + N_b} \left[N_a S(C_a, C_k) + N_b S(C_b, C_k) \right] \\ &\leq \frac{N_a}{N_a + N_b} S_w + \frac{N_b}{N_a + N_b} S_w \leq S_w \ . \end{aligned} \tag{3}$$

Since $S(C_m, C_k)$ would not be better than S_w, we can avoid computing it. Thus, the clusters in the heap that require the computation of a new cluster

score, and possibly its insertion in the heap, is limited to the cases, illustrated in the next two subsections, where either $S(C_a, C_k)$ or $S(C_b, C_k)$, or both, appears in the heap.

3.1.2. $S(C_a, C_k)$ and $S(C_b, C_k)$ are both in the heap

The computation of the new cluster score $S(C_m, C_k)$ is straightforward. Since $S(C_a, C_k)$ and $S(C_b, C_k)$ are both available, average score $s = S(C_m, C_k)$ can be easily computed by means of equation (2), and inserted in the heap because score s is at least equal to S_w.

3.1.3. Only $S(C_a, C_k)$ is in the heap

If one of the scores, say $S(C_a, C_k)$, is in the heap but the other is not, the average score $S(C_m, C_k)$ cannot simply obtained by using equation (2).
For a generic score function $S(.,.)$ it would be necessary to compute all pairwise scores between each vector in C_b and each vector in C_k. Since the number of these pairs is potentially huge, the proposed approach would be not effective, or even unfeasible. For this reason, we rely on a class of scoring functions that allow obtaining $S(C_m, C_k)$ as a function of the average of two representation vectors, each associated to the corresponding cluster. This leads to a much more effective computation of $S(C_m, C_k)$. Only if $S(C_m, C_k) \geq S_w$, the new score is inserted in the heap. Our scoring functions generalize what was proposed in [24, 25] for Euclidean distance scoring.

3.2. Scoring functions

Extending the Clustering Feature [24] concept, we consider the class of scoring functions defined by the product of two vector–valued functions:

$$S(\mathbf{x}_i, \mathbf{x}_j) = \overline{\mathbf{f}}(\mathbf{x}_i)^T \overline{\mathbf{g}}(\mathbf{x}_j) \ . \tag{4}$$

Since a scoring function must be symmetric, it is also necessary that $\overline{\mathbf{f}}(\mathbf{x}_i)^T \overline{\mathbf{g}}(\mathbf{x}_j) = \overline{\mathbf{g}}(\mathbf{x}_i)^T \overline{\mathbf{f}}(\mathbf{x}_j)$.

From (1), the average merging score of two clusters C_a and C_b, in terms of the vectors associated to each cluster, $\overline{\mathbf{f}}(\mathbf{x}_i)$ and $\overline{\mathbf{g}}(\mathbf{x}_j)$, respectively, is:

$$S(C_a, C_b) = \frac{1}{N_a N_b} \sum_{i \in C_a} \sum_{j \in C_b} \overline{\mathbf{f}}(\mathbf{x}_i)^T \overline{\mathbf{g}}(\mathbf{x}_j) = \left[\frac{1}{N_a} \sum_{i \in C_a} \overline{\mathbf{f}}(\mathbf{x}_i) \right]^T \left[\frac{1}{N_b} \sum_{j \in C_b} \overline{\mathbf{g}}(\mathbf{x}_j) \right] \ . \tag{5}$$

This approach allows computing the score of the new cluster by means of the dot–product of the average of two transformed vectors. Thus, we associate to each cluster two average transformed vectors, one for each function:

$$\mathbf{x}_{f,C_k} = \frac{1}{N_k} \sum_{i \in C_k} \overline{\mathbf{f}}(\mathbf{x}_i) \ , \quad \mathbf{x}_{g,C_k} = \frac{1}{N_k} \sum_{i \in C_k} \overline{\mathbf{g}}(\mathbf{x}_i) \ , \tag{6}$$

where N_k is the number of vectors in cluster C_k. If two clusters C_a and C_b merge into cluster C_m, the new transformed vector $\mathbf{x}_{f,C_m}$ can be easily obtained from the vectors $\mathbf{x}_{f,C_a}$ and $\mathbf{x}_{f,C_b}$ as their weighted average:

$$\begin{aligned} \mathbf{x}_{f,C_m} &= \frac{1}{N_a + N_b} \sum_{i \in C_m} \overline{\mathbf{f}}(\mathbf{x}_i) = \frac{1}{N_a + N_b} \left[\sum_{i \in C_a} \overline{\mathbf{f}}(\mathbf{x}_i) + \sum_{i \in C_b} \overline{\mathbf{f}}(\mathbf{x}_i) \right] \\ &= \frac{[N_a \mathbf{x}_{f,C_a} + N_b \mathbf{x}_{f,C_b}]}{N_a + N_b} \ . \end{aligned} \tag{7}$$

The new transformed vector $\mathbf{x}_{g,C_m}$ can be similarly computed.
The average score of the new cluster C_m with respect to another cluster C_k can then be computed as the dot–product of the two vectors:

$$S(C_m, C_k) = \mathbf{x}_{f,C_m}^T \ \mathbf{x}_{g,C_k} \ . \tag{8}$$

Thus, the average of all pairwise scores between two clusters can be effectively computed by means of a single dot–product, rather than by averaging all the inter–cluster pairwise scores.

We will show in Section 5 that, for a set of commonly used scoring functions, the dimensions of $\mathbf{x}_{f,C_k}$ and $\mathbf{x}_{g,C_k}$ are similar to the ones of the original item vectors.

4. K–UPGMA detailed steps

The detailed steps of K–UPGMA are summarized in Algorithm 2. Each singleton cluster is associated with a representation vector initialized according to Section 5. T threads (or processes) are created in a single node, or in multiple computation nodes, to compute in concurrency all the pairwise scores of the vector set. Each thread computes the scores of a block B of vectors, and quick–selects the k–best scores for that block. The T individual k–best lists are then merged (using again quick–select) also keeping the

Algorithm 2 K–UPGMA: detailed

Input: A set of vectors $\mathcal{V}$, and the number k of scores that can been kept in memory.
Output: A cluster hierarchy $\mathcal{H}$ over $\mathcal{V}$.
Initialization: Initialize the cluster set C by defining a singleton cluster $C_i = \{i\}$ for every vector $i \in \mathcal{V}$.
Initialize the appropriate representation vector for each cluster C_i according to Section 3.2.
Creates a fixed number T of threads devoted to the computation of the pairwise scores
while $|C| > 1$ **do**
  if BestHeap is empty **then**
    Refill the k–best list: The pairwise scores for the set of clusters C is divided into blocks of maximum dimension B.
    repeat
      Process T blocks in parallel.
      Each thread selects the k–best clusters for its block.
      The T k–best lists are then merged, keeping the worst score S_w.
    until All pairwise scores computed
    Create for each entry in k–best list the corresponding entry in the heap vector, sparse matrix, and BestHeap structures illustrated in Fig. 1
  end if
  Cluster–pair selection: Get the best scoring cluster pair (C_i, C_j) from the BestHeap.
  Cluster–pair merging: Merge its elements into cluster C_m.
  Compute the representation vectors associated to cluster C_m (7).
  Discard every entry referring to clusters C_i or C_j from the sparse matrix, and from the corresponding heap vector, possibly updating BestHeap.
  Set $C = (C \setminus \{C_i, C_j\}) \cup \{C_m\}$.
  Append the pair (C_i, C_j) and the corresponding score to hierarchy $\mathcal{H}$.
  Update:
  for $C_k \in C$ **do**
    Compute the score $S(C_m, C_k)$ according to the rules given in Sections 3.1 and 3.2
    if $S(C_m, C_k) > S_w$ **then**
      Insert this information in the sparse matrix, and in the corresponding heap vector, possibly updating BestHeap.
    end if
  end for
end while

current worst score S_w. Keeping the worst score allows reducing the computation burden of the quick–select. When the final k–best list of clusters has been collected, its entries are used for creating the heap vectors, sparse matrix, and BestHeap structures. The procedure then loops selecting the best cluster pair (C_i, C_j), found at the top of the BestHeap, until it remains empty.
The pair (C_i, C_j), and the corresponding merging score is appended to the dendrogram. The representation vector associated to this new cluster is computed. All the entries in the heap referring to cluster C_i and C_j are discarded, and the heap is updated inserting new entries according to the rules given in Sections 3.1, with associated representation vector computed according to (7).
Since the number of elements in the heap decreases as the dendrogram grows, the heap shrinks until it remains empty. The dimension of the k–best list, thus, determines an implicit score threshold that defines the maximum height of the hierarchy tree for the current iteration. Having the possibility of allocating a huge k–best list in memory would allow the dendrogram to be completed in a single iteration, but for a large dataset and reasonable k–best list size, the algorithm must iterate until the dendrogram is completed. When the k-best list is empty, it is refilled with the k–best scores of the current cluster pairs, which can be effectively computed from their representation vectors.

Let us consider, for comparison, the RNN approach, which iteratively builds nearest neighbor chains. Each chain consists of an initial randomly selected vector, followed by its NN, which is followed by its NN among the remaining clusters, and so on. The distances between adjacent vectors in the NN chain are monotonically decreasing, and the last pair of nodes are Reciprocal Nearest Neighbors. Since searching the NN is sequential, RNN cannot exploit massive parallel computation. Indeed, each RNN iteration must compute the score of a single cluster with respect to all other clusters. This operation can be performed by means of a vector–matrix product. Our approach, instead, relies on much more effective matrix–matrix product computations, and it is worth noting that the computation of the scores and of the partial k–best lists can be distributed among different computer nodes. A comparison of the running times for the two approaches is given in Table 1 of Section 7.

RNN can exploit massive parallel computation only if it precomputes a set of distances, i.e., if it uses the same approach that we are proposing.

In particular, our heap structure can be used for a fast implementation of the RNN algorithm that builds the dendrogram up to the implicit threshold defined by the heap size. The complete dendrogram can be obtained by iteratively refilling the heap with an approach similar to the one described for K–UPGMA.

5. Examples of scoring functions

An equivalent formulation of the scoring function (4) is:

$$S(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{f}(\mathbf{x}_i)^T \mathbf{g}(\mathbf{x}_j) + h(\mathbf{x}_i) + h(\mathbf{x}_j) \tag{9}$$

where h is a scalar–valued function. By setting:

$$\mathbf{f} = \overline{\mathbf{f}} \ , \quad \mathbf{g} = \overline{\mathbf{g}} \ , \quad h(\mathbf{x}) = 0 \ , \tag{10}$$

(9) becomes (4), and by setting:

$$\overline{\mathbf{f}}(\mathbf{x}_i) = [\mathbf{f}(\mathbf{x}_i) \ , h(\mathbf{x}_i) \ , 1]^T \quad , \quad \overline{\mathbf{g}}(\mathbf{x}_j) = [\mathbf{g}(\mathbf{x}_j) \ , 1 \ , h(\mathbf{x}_j)]^T \tag{11}$$

their product (4) is equivalent to (9).

We here show that scoring functions commonly used for clustering can be represented according to (9), and that the size of the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ is equal or close to the original vector size.

We first formulate as in (9) the most popular scoring functions in clustering: cosine similarity scoring and Squared Euclidean distance. We then extend the formulation to the similarity scores produced by Probabilistic Linear Discriminant Analysis (PLDA) [30, 31], by its non–linear extension [32], and by the Pairwise Support Vector Machine (PSVM) [33, 34]. Finally, we show that we can also deal with calibrated and normalized scores, and with sets of scores of different classifiers that are linearly combined to improve system accuracy.

5.1. Cosine similarity scoring

Since the definition of the cosine similarity score is:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\| \, \|\mathbf{x}_2\|} \ , \tag{12}$$

the corresponding functions $\mathbf{f}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, and $h(\mathbf{x})$ of (9) can be set as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|} \ , \quad h(\mathbf{x}) = 0 \ . \tag{13}$$

Clustering, thus, can be performed using pre–normalized vectors.

5.2. Squared Euclidean distance

The scoring function for the squared Euclidean distance is:

$$S(\mathbf{x}_1, \mathbf{x}_2) = -\frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = \mathbf{x}_1^T\mathbf{x}_2 - \frac{1}{2}\mathbf{x}_1^T\mathbf{x}_1 - \frac{1}{2}\mathbf{x}_2^T\mathbf{x}_2 \ . \tag{14}$$

This leads to defining:

$$\mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{x}) = \mathbf{x} \ , \quad h(\mathbf{x}) = -\frac{1}{2}\|\mathbf{x}\|^2 \ . \tag{15}$$

Score computation requires, thus, that for each item vector $\mathbf{x}$, also $h(\mathbf{x})$ is pre–computed and stored in memory. The increase of memory and computation costs, however, is minimal because $h(\mathbf{x})$ is a scalar.

It is worth noting that keeping the mean and variance of the clusters to speed up the computation of the similarities between clusters was proposed in [24, 25] for clustering based on the Euclidean distance. We here show that Squared Euclidean distance is just a particular case of the more general class of score functions which allow efficient computation of cluster average scores.

Cosine and Euclidean scoring do not rely on prior information about the distribution of the item vectors. We can expect, however, better clustering performance if this information is available in terms of a statistical model. In the next subsections we recall the PLDA model and its fast scoring formulation that allows obtaining the pairwise similarity between two representation vectors. A similar formulation can be used for PSVM scoring.

5.3. PLDA scoring

Probabilistic Linear Discriminant Analysis is one of the most effective models for speaker recognition. It can be used in combination with different types of compact representations of a speech segment, such as i–vectors [35], e–vectors [36], or speaker embeddings [37].
Given a pair of vectors, a PLDA classifier allows computing the log–likelihood ratio between the hypotheses that they belong to the same speaker or to different speakers. The Gaussian PLDA model is defined as:

$$\mathbf{x} = \mathbf{m} + \mathbf{U}\mathbf{y} + \boldsymbol{\varepsilon} \tag{16}$$

where $\mathbf{m}$ is the mean vector of the vectors estimated in training from the utterances of a large number of speakers, $\mathbf{y}$ represents the speaker factors corresponding to a given utterance, which are assumed to obey a Normal prior

distribution $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{U}$ represents the trained speaker factor loading matrix, and $\boldsymbol{\varepsilon}$ is a residual term, with prior distribution $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}^{-1})$.

Given these assumptions, the PLDA log–likelihood ratio between the "same speaker" hypothesis $\mathcal{H}_S$, and the "different speaker" hypothesis $\mathcal{H}_D$, is given by:

$$\begin{aligned} S(\mathbf{x}_1, \mathbf{x}_2) &= \log \frac{P(\mathbf{x}_1, \mathbf{x}_2|\mathcal{H}_S)}{P(\mathbf{x}_1, \mathbf{x}_2|\mathcal{H}_D)} \\ &= \log \frac{P(\mathbf{x}_1, \mathbf{x}_2|\mathbf{y})P(\mathbf{y})}{P(\mathbf{y}|\mathbf{x}_1, \mathbf{x}_2)} \frac{P(\mathbf{y}|\mathbf{x}_1)}{P(\mathbf{x}_1|\mathbf{y})P(\mathbf{y})} \frac{P(\mathbf{y}|\mathbf{x}_2)}{P(\mathbf{x}_2|\mathbf{y})P(\mathbf{y})} \\ &= \log \frac{P(\mathbf{y}|\mathbf{x}_1)P(\mathbf{y}|\mathbf{x}_2)}{P(\mathbf{y}|\mathbf{x}_1, \mathbf{x}_2)P(\mathbf{y})} \end{aligned} \tag{17}$$

In [33] it is shown that this speaker verification score can be computed as the quadratic function of the speaker vector pair:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{A} \mathbf{x}_1 + \mathbf{x}_2^T \mathbf{A} \mathbf{x}_2 + \mathbf{x}_1^T \mathbf{B} \mathbf{x}_2 + \mathbf{x}_1^T \mathbf{c} + \mathbf{x}_2^T \mathbf{c} + k_1 \ , \tag{18}$$

where the square symmetric matrices $\mathbf{A}$ and $\mathbf{B}$, vector $\mathbf{c}$, and the constant term k_1, are functions of the parameters of the PLDA model (16). UPGMA clustering can be performed, using PLDA scores, by defining the functions $\mathbf{f}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, and $h(\mathbf{x})$ of equation (9) as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{x} \ , \quad \mathbf{g}(\mathbf{x}) = \mathbf{B}\mathbf{x} \quad , \quad h(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{c} + \frac{k_1}{2} \ . \tag{19}$$

Score computation requires that an array, $\mathbf{Bx}$, and a scalar $h(\mathbf{x})$ are pre–computed and stored in memory in addition to the item vector. Again, this has low impact on memory and computation costs. Furthermore, since matrix $\mathbf{B}$ is positive definite, the additional storage can be reduced by using the equivalent set of functions:

$$\mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{x}) = \mathbf{B}^{\frac{1}{2}}\mathbf{x} \quad , \quad h(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{c} + \frac{k_1}{2} \ , \tag{20}$$

i.e., just storing a single transformed vector $\mathbf{B}^{\frac{1}{2}}\mathbf{x}$.

5.4. Non–Linear PLDA scoring

Non–linear PLDA (NLPLDA) [38, 32] introduces a non–linear transformation of vectors aimed at better fitting the Gaussian assumptions of PLDA. The model can be described as:

$$\mathbf{F}(\mathbf{x}) = \mathbf{m} + \mathbf{U}\mathbf{y} + \boldsymbol{\varepsilon} \ , \tag{21}$$

where $\mathbf{F}$ is a parametric invertible function. In [38, 32] it was shown that, once the model has been trained, NLPLDA scoring can be obtained as the standard PLDA scoring just using the transformed vectors $\mathbf{F}(\mathbf{x})$.

5.5. PSVM scoring

A successful alternative to the generative PLDA model has been presented in [33, 34]. where a single pairwise SVM (PSVM) is trained to classify a pair of vectors as belonging to the "same speaker", or to the " different speaker" class. The PSVM model is trained to estimate the parameters of a symmetric quadratic function approximating a log–likelihood ratio score [33]. Thus, the scoring function can be computed similarly to (18), of course using different parameters $\mathbf{A}$, $\mathbf{B}$, $\mathbf{c}$, and k_1. However, since it cannot be assumed that matrix $\mathbf{B}$ is positive definite, for PSVM scores we cannot rely on (20), but we must store not only vector $\mathbf{x}$, but also the array $\mathbf{Bx}$.

5.6. Score calibration

The scores obtained by a classifier cannot always be interpreted as the likelihood-ratio between the "same speaker" and "different speaker" hypotheses. However, the parameters of a linear score mapping can be estimated on a held–out development dataset, and applied to map the original scores to new scores that can be used as calibrated log–likelihood ratios [39]. Linear score calibration can be embedded, without additional costs, in our scoring functions.
Let $S_u(\mathbf{x}_1, \mathbf{x}_2)$ be an uncalibrated score, the corresponding linearly calibrated score is given by:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \alpha S_u(\mathbf{x}_1, \mathbf{x}_2) + \beta \ , \tag{22}$$

where α and β are scalars estimated on the development dataset. According to (9) the calibrated scores are given by:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \alpha \mathbf{f}_u(\mathbf{x}_1)^T \mathbf{g}_u(\mathbf{x}_2) + \alpha h_u(\mathbf{x}_1) + \alpha h_u(\mathbf{x}_2) \ . \tag{23}$$

Thus, UPGMA clustering can be performed using the calibrated scores by setting in (9) :

$$\mathbf{f}(\mathbf{x}) = \alpha \mathbf{f}_u(\mathbf{x}) \ , \quad \mathbf{g}(\mathbf{x}) = \mathbf{g}_u(\mathbf{x}) \quad , \quad h(\mathbf{x}) = \alpha h_u(\mathbf{x}) + \frac{\beta}{2} \ . \tag{24}$$

Since parameter α is typically positive, $\mathbf{f}$ and $\mathbf{g}$ can be replaced by:

$$\mathbf{f}(\mathbf{x}) = \alpha^{\frac{1}{2}} \mathbf{f}_u(\mathbf{x}) \ , \quad \mathbf{g}(\mathbf{x}) = \alpha^{\frac{1}{2}} \mathbf{g}_u(\mathbf{x}) \ . \tag{25}$$

If $\mathbf{f}_u = \mathbf{g}_u$, this allows keeping in memory just one array.

All the scoring functions presented so far keep the dimensions of the original item vectors, plus one for the scalar h.

5.7. Score normalization

In addition to score calibration, score normalization allows the designer of a classifier to select a constant decision threshold in spite of possible mismatches between the development and test conditions.
Among the techniques proposed for score normalization, the so called symmetric normalization, or S–norm, is one of the most effective both in term of performance and computation costs [31, 40].

Given an unnormalized score $S_u(\mathbf{x}_1, \mathbf{x}_2)$, S–norm score is defined as:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \frac{S_u(\mathbf{x}_1, \mathbf{x}_2) - \mu(\mathbf{x}_1)}{2\sigma(\mathbf{x}_1)} + \frac{S_u(\mathbf{x}_1, \mathbf{x}_2) - \mu(\mathbf{x}_2)}{2\sigma(\mathbf{x}_2)} \ , \tag{26}$$

where $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ denote the mean and standard deviation of the scores computed comparing a vector $\mathbf{x}$ with respect to an held–out set of different speaker vectors.
Rewriting (26) in terms of the functions $\mathbf{f}_u$, $\mathbf{g}_u$ and h_u, we obtain:

$$\begin{aligned} S(\mathbf{x}_1, \mathbf{x}_2) = \quad & \frac{\mathbf{f}_u(\mathbf{x}_1)^T \mathbf{g}_u(\mathbf{x}_2) + h_u(\mathbf{x}_1) + h_u(\mathbf{x}_2) - \mu(\mathbf{x}_1)}{2\sigma(\mathbf{x}_1)} \\ & + \frac{\mathbf{f}_u(\mathbf{x}_1)^T \mathbf{g}_u(\mathbf{x}_2) + h_u(\mathbf{x}_1) + h_u(\mathbf{x}_2) - \mu(\mathbf{x}_2)}{2\sigma(\mathbf{x}_2)} \ . \end{aligned} \tag{27}$$

The S–norm score $S(\mathbf{x}_1, \mathbf{x}_2)$ can, thus, be computed by setting:

$$\begin{aligned} & \mathbf{f}(\mathbf{x}) = \left[\frac{\mathbf{f}_u(\mathbf{x})}{2\sigma(\mathbf{x})}, \frac{1}{2\sigma(\mathbf{x})}, \mathbf{f}_u(\mathbf{x}), h_u(\mathbf{x}) \right]^T \ , \ \mathbf{g}(\mathbf{x}) = \left[\mathbf{g}_u(\mathbf{x}) \ , h_u(\mathbf{x}), \frac{\mathbf{g}_u(\mathbf{x})}{2\sigma(\mathbf{x})}, \frac{1}{2\sigma(\mathbf{x})} \right]^T \\ & h(\mathbf{x}) = \frac{h_u(\mathbf{x}) - \mu(\mathbf{x})}{2\sigma(\mathbf{x})} \ . \end{aligned} \tag{28}$$

Clustering using S–normalized scores requires keeping in memory, for each item vector, two precomputed arrays $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$, and a scalar $h(x)$, the former having double dimension with respect to the item vector.

5.8. Score combination

It is well known that the combination of the scores of a set of classifiers that use different models or features increases the performance of a speaker recognition system. The linear combination of the scores of K different classifiers is given by:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \sum_{k=1}^{K} \alpha_k S_u^k(\mathbf{x}_1^k, \mathbf{x}_2^k) + \beta \tag{29}$$

where S_u^k is the score produced by the k–th classifier for the k-th pair of vectors $\mathbf{x}_1^k, \mathbf{x}_2^k$. As for calibration, equation (29) can be rewritten as:

$$S(\mathbf{x}_1, \mathbf{x}_2) = \sum_{k=1}^{K} S^k(\mathbf{x}_1^k, \mathbf{x}_2^k) \ , \tag{30}$$

where

$$S^k(\mathbf{x}_1, \mathbf{x}_2) = \alpha_k S_u^k(\mathbf{x}_1^k, \mathbf{x}_2^k) + \frac{\beta}{K} \ . \tag{31}$$

The scoring functions $\mathbf{f}^k$, $\mathbf{g}^k$ and h^k corresponding to each score S^k can be computed from the corresponding functions $\mathbf{f}_u^k$,, $\mathbf{g}_u^k$ and h_u^k, as shown in Section 5.6.

Since the combination score is obtained as the sum of the scores S^k, it can be computed by setting, for each item vector:

$$\mathbf{f}(\mathbf{x}) = \left[\mathbf{f}^1(\mathbf{x}), \cdots, \mathbf{f}^K(\mathbf{x})\right]^T \ , \ \mathbf{g}(\mathbf{x}) = \left[\mathbf{g}^1(\mathbf{x}), \cdots, \mathbf{g}^K(\mathbf{x})\right]^T \ , \ h(\mathbf{x}) = \sum_{k=1}^{K} h^k(\mathbf{x}) \tag{32}$$

The dimension of the arrays $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ is the sum of the dimensions of the representation vectors of each classifier, thus, the score computation time in clustering increases as a function of the number of classifiers scores that are combined.

6. Silhouette Width Criterion

A further advantage of using UPGMA is that it is possible to devise a fast technique for computing an approximate Silhouette Width Criterion (SWC) curve [27], which allows estimating the number of speakers in large datasets, and evaluating the quality of the corresponding clusters.

SWC was selected based on the results of the comparison of 40 internal clustering validity measures performed in [27], which shows that SWC is not only one of the few best techniques in relative terms, but it is also the most robust when used in different scenarios. Silhouette was also among the best cluster validity measure compared in [41].

The silhouette value is a measure of how similar a vector $\mathbf{x}_i$ is to its own cluster (tightness) compared to other clusters (separation) [26], formally:

$$s_{\mathbf{x}_i} = \frac{b_{\mathbf{x}_i} - a_{\mathbf{x}_i}}{\max(a_{\mathbf{x}_i}, b_{\mathbf{x}_i})} \ , \tag{33}$$

where a_{x_i} is the average distance between vector $\mathbf{x}_i$ and the other vectors within the same cluster, and $b_{\mathbf{x}_i}$ is the average distance between $\mathbf{x}_i$ and the vectors in its closest cluster. The denominator is just a normalization term, and $s_{\mathbf{x}_i} = 0$ if $\mathbf{x}_i$ belongs to a singleton cluster.

The Silhouette Width Criterion (SWC) [27] is defined as the average silhouette over all the item vectors:

$$sw = \frac{1}{N} \sum_{1}^{N} s_{x_i}. \tag{34}$$

The best clustering structure, characterized, at the same time, by maximum compactness and separation, is expected to be found for the set of clusters that lead to the maximum value of sw.

Unfortunately, the computation of the sw value for a single set of clusters has complexity $\mathcal{O}(N^2)$ [27]. The complexity becomes cubic if we need to find the maximum of the SWC curve, i.e., the average silhouette value for the set of clusters obtained by cutting the dendrogram at decreasing levels of score similarity. Cubic time complexity makes automatic selection of the number of clusters unfeasible even for small size dataset.

However, since UPGMA merges the clusters according to their average similarity, an approximate but extremely fast computation of the sw values can be obtained exploiting the value b_p associated to each dendrogram entry p, consisting of the triple $\{C_i, C_j, b_p\}$, where the subscript p refer to the parent entry merging clusters C_i and C_j. By definition of UPGMA, b_p is the average dissimilarity between these clusters. Thus, given a vector $\mathbf{x}_i$ assigned to cluster C_i, we can take b_p as an approximation of $b_{\mathbf{x}_i}$ in (33).
The average dissimilarity w_i between each vector x_i and all the other vectors

Algorithm 3 Fast Silhouette Width Criterion

Input: A dendrogram array structure H of size $N-1$. Each entry of H, representing a cluster, includes the left and right clusters that are merged, the average dissimilarity between these clusters, the number of leaves stemming from it, and its parent index.
Output: An array of sw values as a function of the number of clusters.

```
i = 1
while i < N do
  if has_more_than_two_leaves(i) then
    Compute w[i] according to (35).
    s[i] = (b[p[i]] - w[i]) / max (b[p[i]], w[i])
  else
    w[i] = b[i]
    s[i] = 0
  end if
  sw[i+1] = sw[i] - s[left_child[i]] - s[right_child[i]] + s[i]
  i = i + 1
end while
```

within cluster C_i, can be computed as:

$$\begin{aligned} w_i =& \frac{b_i l_{i1} l_{i2} + \frac{w_{i1} l_{i1}(l_{i1}-1)}{2} + \frac{w_{i2} l_{i2}(l_{i2}-1)}{2}}{l_{i1} l_{i2} + \frac{l_{i1}(l_{i1}-1)}{2} + \frac{l_{i2}(li_2-1)}{2}} \\ =& \frac{2b_i l_{i1} l_{i2} + w_{i1} l_{i1}(l_{i1}-1) + w_{i2} l_{i2}(l_{i2}-1)}{(l_{i1}+l_{i2})(l_{i1}+l_{i2}-1)} \ , \end{aligned} \tag{35}$$

where l_{i1} and l_{i2} are the number of leaves of the two children of cluster C_i in the dendrogram, w_{i1} and w_{i2} are their within cluster average dissimilarity, and $w_i = b_i$ for clusters including only two vectors.
Considering all vectors in cluster C_i, their approximate silhouette value can be obtained as:

$$s_i = l_i \cdot \frac{b_p - w_i}{\max(b_p, w_i)} \ . \tag{36}$$

To avoid division by 0, $s_i = 0$ if $b_p = 0$, i.e., if C_p merges clusters that include identical vectors only.

Our fast procedure for obtaining the UPGMA approximate sw is summarized in Algorithm 3. The values of w_i and of sw_i are computed iteratively as

a function of the number of clusters obtained by cutting the dendrogram at decreasing similarity levels. Starting from the leaves of the hierarchy, which have null silhouette value, we proceed towards the root of the tree by computing the average within cluster dissimilarity w_i of the current dendrogram entry i, the silhouette value s_i, and the silhouette width value sw_i for the corresponding set of clusters. The cluster structure changes, at each iteration, only because two clusters are merged, thus, the sw value can be simply updated by subtracting the silhouette value of the clusters that are merged, and by adding the silhouette value of the merged cluster.

Since the dendrogram b_p values produced by UPGMA clustering using the PLDA or PSVM are approximate average log–likelihood ratio scores, we transform these values to the dissimilarity scores that are needed for computing the Silhouette Width Criterion curve. The transformation is performed according to:

$$\hat{b} = \exp(-b/b^*) \ , \tag{37}$$

where $b^* = 3 \cdot \sigma$, and σ is the standard deviation of the Gaussian fitting the distribution of the b_p values in the dendrogram. This transformation normalizes and compresses the range of the b values in the dendrogram so that these similarity scores are projected to a dissimilarity "linear" scale.

7. Experiments

This section presents the results of a set of experiments performed on Nuance servers, on a set of pre-extracted e–vectors [36] coming from an anonymized text-dependent dataset of passphrases from phone calls that last 5 seconds on average. All speakers pronounce the same passphrase. This dataset includes 900K length–normalized e–vectors of dimension $d = 400$.
A subset of these vectors, consisting of 350K vectors from 83123 speakers, are labeled. The average number of utterance per speaker for the 350K dataset is 4.2, with standard deviation 5.2. Speakers with more than 50 vectors could be potential fraudsters.
For this subset we can compute some external measures to validate the quality of the clusters produced by K–UPGMA, either considering a priori known the number of speakers, or automatically detecting their number by means of the Silhouette Width Criterion.
The complete set of 900K vectors has been used, instead, for testing the scalability of the K–UPGMA approach, using a fixed size for the k–best list.

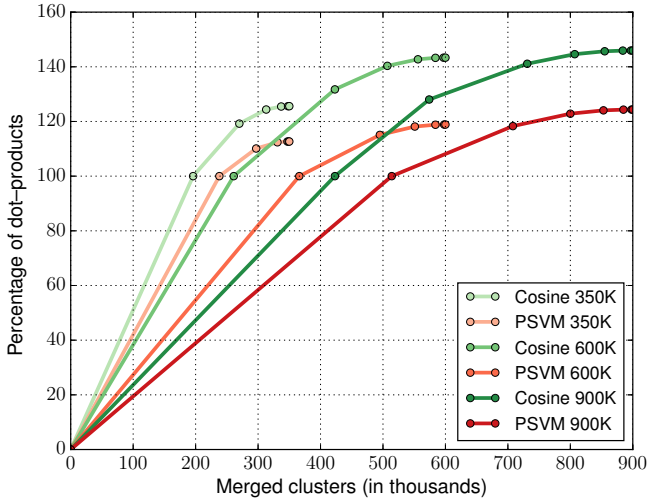

Figure 2: Percentage of scores computed by K–UPGMA using cosine and PSVM scores, as a function of the number of merged clusters for three sets of item vectors. 100% corresponds to $\frac{N*(N-1)}{2}$ scores. The abscissa of a filled circle in the graph represents the number of clusters merged at each refill of the k–best scores.

A held–out set of 59862 e–vectors from 6000 speakers, not used in testing, has been used for PLDA and PSVM training.
All the experiments have been performed on a single server equipped with a 24 core Intel(R) Xeon(R) E5-2680 v3 2.50GHz, and 128GB of main memory.

7.1. K–UPGMA computational costs

Fig. 2 shows the percentage of scores computed by K–UPGMA, in addition to the standard $\frac{N*(N-1)}{2}$ scores, using either cosine or PSVM scores, as a function of the number of merged vectors for three different dataset size: 350K, 600K, and 900K, respectively. The abscissa of a filled circle in the graph represents the total number of vectors merged after each K–UPGMA iteration. These results were obtained keeping fixed the k–best list size, equal to 2 millions elements. Examining these plots, it can be observed that more than a half of the clusters are merged after the first iteration, i.e., after 100% of the scores have been computed.

In the worst case, clustering 900K items using cosine scoring requires approximately 40% additional score computations. This percentage of ad-

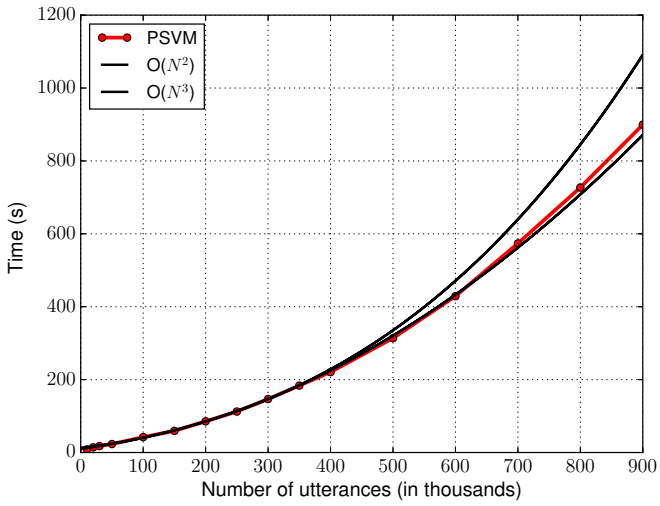

Figure 3: Elapsed time of K–UPGMA using the PSVM similarity scores as a function of the number of dataset utterances, and the corresponding quadratic and cubic polynomial fitting functions.

ditional computations decreases for smaller datasets. It is also interesting noting that the percentage of additional computations is always smaller for PSVM scores than for cosine scores. Indeed, using the prior information provided by the model makes the clustering task easier because it improves the discrimination among same speaker and different speakers vectors. This helps keeping in the heap the clusters that contribute to growing the dendrogram, discarding the ones scoring worse than the worsts score S_w. Since successive iterations recompute the scores of a smaller and smaller subset of the original item set, the overall time complexity of K-UPGMA remains almost quadratic in the number of item vectors. This is shown in Fig. 3, which compares the elapsed time of K–UPGMA using the PSVM similarity scores as a function of the number of clustered vectors, with respect to the quadratic and cubic polynomial fitting functions. These functions were estimated using the elapsed times for an increasing number of item vectors in the interval $[50K - 300K]$. The elapsed time curve of K–UPGMA is much closer to the quadratic than to the cubic fitting function.

The same trend can be observed for different similarity scores, as shown in Fig. 4. PSVM scores allow obtaining the fastest clustering time, whereas the elapsed time increases using cosine similarity scores due to their inferior

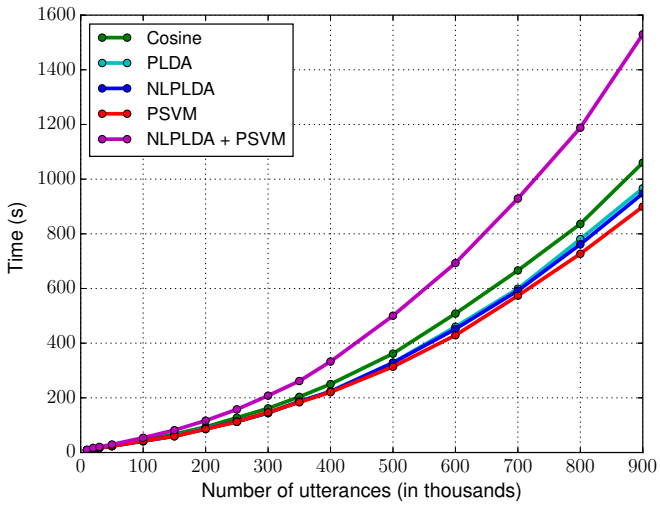

Figure 4: Elapsed time of K–UPGMA using different similarity scores, as a function of the number of dataset utterances.

discrimination capability. Clustering with the combination of the scores of the PSVM and NLPLDA system (label NLPLDA+PSVM in figure) is much slower because, although the system is more accurate, the representation vectors double their dimensions.

To further assess the scaling capabilities of K–UPGMA, we performed a set of experiments artificially generating new vectors. In particular, the new vectors were sampled according to the PLDA model (16), and added to our labeled dataset consisting of 350K vectors. We considered augmented datasets including 1, 2, and 4 million vectors. The latter is the maximum number supported by our current implementation without exceeding our system memory. The results, using PSVM scores, are shown in the first part of Table 1, for different sizes of the k–best list, in terms of running time, and of percentage of score computations with respect to $\frac{N*(N-1)}{2}$.

As expected, increasing the number of vectors in the dataset, but keeping fixed the size of the k–best list, adversely affects the running time of K–UPGMA. This happens because the size of the k–best list becomes insufficient to accommodate most of the same–speaker scores, which are by definition the best ones, thus, more refill operation are required. However, we get similar percentage of score computations by doubling the dimension

Table 1: Top: Run time, in [hh:]mm:ss, and percentage of score computations with respect to N^2 for datasets including 350K, 1M, 2M, and 4M vectors, respectively, using an increasing size of the k–best list. Bottom: Corresponding run time for standard RNN.

k–best list size	Number of vectors in the dataset							
	350K		1M		2M		4M	
2M	3:03	**112.7%**	18:24	124.8%	1:42:24	152.0%	7:04:03	199.9%
4M	3:39	109.0%	18:23	**113.3%**	1:11:00	122.1%	5:24:04	149.1%
8M	4:34	106.8%	20:57	109.7%	1:14:37	**113.6%**	4:47:52	122.5%
16M	6:44	105.3%	24:05	107.7%	1:16:13	110.2%	4:50:39	**115.0%**
RNN	1:36:00		12:40:00		50:52:00		204:20:00	

of the dataset, but also doubling the k–best list size. Using linearly–growing k–best list and dataset size allows keeping approximately quadratic time complexity, as shown by the bold percentage values in the table. It is worth noting that scaling up the dimension of the k–best list is not a problem because it memory costs are order of magnitudes smaller with respect to the ones necessary for storing the 400–dimensional representation vectors. It is possible to compare the running time of standard RNN for clustering the same datasets at the bottom of Table 1. K–UPGMA is from 30 to 40 times faster than RNN.[1] As stated in Section 4, our heap structure can be used for a fast implementation of the RNN algorithm. This implementation of RNN requires multiple heaps to be managed, as in K–UPGMA. However, the vast majority of the computational costs for both algorithms remains the evaluation and selection of the similarity scores, whereas the search and merge steps are not expensive (less than 10% and 4% for our method with the 1M and 4M items datasets, respectively), thus, the running times of the two algorithms would be very close.

[1] Another memory efficient algorithm, proposed in [12], is fast Pairwise Nearest Neighbor (fPNN), which has $\mathcal{O}(\tau N^2)$ computational complexity, because after each merge it finds the nearest neighbor of τ clusters on average. However, RNN has a per–merge complexity that is less or equal to $3N$, thus for $\tau > 3$ it is more efficient than fPNN.



Figure 5: True and approximate Silhouette Width Criterion value computed on the dendrogram produced by UPGMA clustering 1000 e–vectors of 190 speakers, using cosine similarity.

7.2. Cluster validation by means of SWC

We first performed an experiment to assess the quality of the approximate Silhouette Width Criterion. In particular, we compared the true and the approximate silhouette *sw* values obtained by clustering 1000 e–vectors belonging to 190 speakers, using cosine similarity. The number of vectors was limited to 1000 because obtaining the SWC values for larger datasets is too expensive having cubic complexity.[2] The two curves, shown in in Fig. 5, are similar and attain their maximum value approximately for the same number of clusters. As shown in Table 2, SWC underestimates the number of clusters both using the true and the approximate SWC curve. Both values, however, are not far from the true number of speakers. Furthermore, three standard external cluster evaluation measures confirm that the performance of the two methods is similar. The first measure is the Adjusted Rand Index (ARI) [42, 43, 27], which represents the number of pairs of vectors that are

[2]The elapsed time, in hh:mm:ss, for producing the silhouettes of all clusters of 100, 200, 400, 1000 vectors, using the standard Python `silhouette_score` function, is 00:00:4, 00:00:34, 00:05:18, and 1:34:57, respectively. The computation of our approximate SWC is, instead, extremely fast because Algorithm 4 has linear complexity.

Table 2: Cluster purity measures for the number of speaker estimated by using the true and approximate SWC, on 1000 e–vectors of 190 speakers.

SWC	True	Approximate	Known number
Estim. number of clusters	165	171	190
ARI	0.97	0.97	0.98
C_{imp}	2.6%	2.5%	3.2%
S_{imp}	5.0%	3.8%	2.8%

Table 3: The SWC from the dendrograms produced by K–UPGMA using different similarity measures.

Number of	Number of	Estimated numer of clusters			
vectors	speakers	Cosine	PLDA	NLPLDA	PSVM
10000	2233	1794	1829	1963	1856
30000	6481	5549	5110	5349	5276
50000	10793	9706	9213	8824	8796
100000	21350	21108	18864	18469	18722
200000	44537	51790	44722	46007	44336
350000	83123	97612	88321	86543	85411

either in the same cluster or in different clusters in the reference and in the obtained partition, divided by the total number of pairs of vectors. The other two measures are the Cluster and Speaker impurity [44, 45], labeled as C_{imp} and S_{imp} in the table. C_{imp} gives the percentage of vectors from different speakers in a cluster, whereas S_{imp} corresponds to the percentage of same speakers vectors that are distributed among the clusters.

Table 3 compares the true number of speakers and the number of clusters estimated by means of the Silhouette Width Criterion from the dendrograms produced by K–UPGMA using different similarity measures. The location of maximum of the SWC curve is a good predictor of the number of true speakers in the dataset, even if it underestimate approximately by 20% the number of true speakers for small datasets. For large datasets, instead, the number of speakers is slightly overestimated.

Table 4: Cluster performance measures of K–UPGMA using different similarity scores on 350000 e–vectors of 82123 speakers. Columns 3–6 refer to different scoring models (see Section VI). Last column refers to the combination of the scores of the last two models.

Similarity	Number of Speakers	Cosine	PLDA	NPLDA	PSVM	NLPLDA+ PSVM
ARI		0.43	0.80	0.86	0.88	0.89
C_{imp}	Known	16.2%	10.7%	9.1%	8.7%	8.3%
S_{imp}	Known	16.3%	10.4%	8.6%	8.5%	7.8%
C_{imp}	Est.	18.9%	11.7%	9.7%	9.1%	8.7%
S_{imp}	Est.	11.2%	8.6%	7.4%	7.5%	7.0%

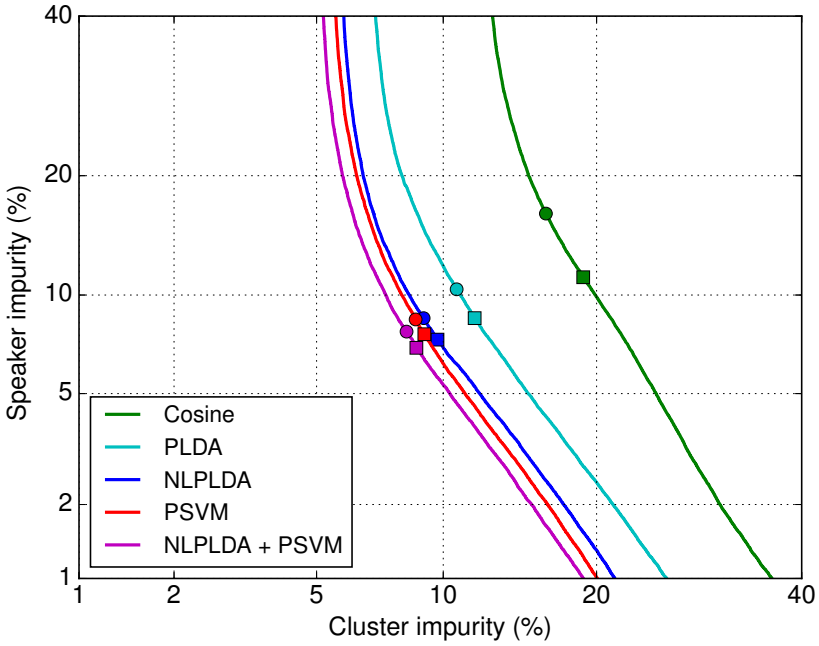

Figure 6: Cluster and speaker impurity trade–off plot obtained by clustering 350000 item vectors using cosine scoring or the scores produced by different models. The filled circle in each curve marks the performance that can be obtained knowing the number of speakers in the dataset, whereas the squares indicate the performance obtained by using SWC for automatic selection of the number of clusters.

Table 4 summarizes the evaluation measures applied to clusters obtained by K–UPGMA on 350000 e–vectors of 82123 speakers, using cosine scoring or the scores produced by a set of classifiers. The results for the classifiers illustrated in Section 5, PLDA, PSVM, NLPLDA, are given in columns 3–6. Last column refers to the combination of the scores of the last two models. Since we obtain similar C_{imp} and S_{imp} values for known and estimated number of speakers, we conclude that approximate SWC scales well, also considering that speaker discrimination becomes more difficult increasing the number of speakers.
Fig. 6 shows the cluster and speaker impurity trade–off plot for the same dataset. The filled circles and the squares in each curve mark the performance that can be obtained by cutting the dendrogram generated by K–UPGMA knowing the number of speakers or according to the approximate Silhouette Width Criterion, respectively. It is interesting noting that, looking at the plots from the right to the left, the more accurate the similarity measure is, the better is clustering performance, and the accuracy of the estimated number of clusters, as also shown in the last row of Table 3.

8. Conclusions

We introduced a fast and exact implementation of UPGMA for large scale vector clustering, and a fast approximate Silhouette Width Criterion, which allows us estimating with good accuracy the number of clusters in a large dataset.
We also described a class of scoring functions that allow obtaining the similarity score of two clusters by means of a single dot–product of the representation vectors associated to the clusters.

A large set of experiments has been performed to evaluate the scalability of our K–UPGMA implementation, showing that the elapsed time of our approach remains almost quadratic with the size of the dataset. Thanks to efficient data structures and algorithms, and exploiting the possibility of parallel computation of the similarity scores between clusters, we have shown that we can perform exact UPGMA clustering of millions of vectors more than 40 times faster than RNN. Our framework can also be used for an efficient implementation of the UPGMA algorithm based on the RNN chain algorithm.

We also assessed the accuracy of the fast approximate Silhouette Width Criterion using an increasing number of vectors. The external cluster vali-

dation measures obtained with the estimated number of cluster on a 350K vectors dataset are in good accordance with the ones obtained by a priori knowing the true number of speakers in the dataset.

For huge datasets only approximate clustering solutions are appropriate, not only because the quadratic complexity of the algorithms considered in this work makes them too slow, but also because it is impossible to store in memory the dataset vectors.

We plan to investigate approximate solutions in a future work.

References

[1] A. K. Jain, Data clustering: 50 years beyond k-means, Pattern Recognition Letters 31 (8) (2010), pp. 651–666.

[2] D. Pandove, S. Goel, R. Rani, Systematic review of clustering high-dimensional and large datasets, ACM on Transactions Knowledge Discovery from Data 12 (2) (2018), pp. 16:1–16:68.

[3] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland, O. Vinyals, Speaker diarization: A review of recent research, IEEE Transactions on Audio, Speech, and Language Processing 20 (2) (2012), pp. 356–370.

[4] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, C. Vair, Stream-based speaker segmentation using speaker factors and eigenvoices, in: Proceedings of ICASSP 2008, 2008, pp. 4133–4136.

[5] S. Shum, N. Dehak, R. Dehak, J. Glass, Unsupervised methods for speaker diarization: An integrated and iterative approach, IEEE Transactions on Audio, Speech, and Language Processing 21 (10) (2013), pp. 2015–2028.

[6] E. Khoury, L. E. Shafey, M. Ferras, S. Marcel, Hierarchical speaker clustering methods for the NIST i-vector challenge, in: Odyssey: The Speaker and Language Recognition Workshop, 2014, pp. 254–259.

[7] D. Garcia-Romero, A. McCree, S. Shum, N. Brümmer, C. Vaquero, Unsupervised domain adaptation for i-vector speaker recognition, in: Proc. of Odyssey 2014, The Speaker and Language Recognition Workshop, 2014, pp. 260–264.

[8] J. Hartigan, M. Wong, A k-means clustering algorithm, Journal of the Royal Statistical Society. Series C: Applied Statistics 28 (1) (1979), pp. 100–108.

[9] A. Ng, M. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: Proc. of Neural Information Processing Systems: Natural and Synthetic, NIPS'01, 2001, pp. 849–856.

[10] U. Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (4) (2007), pp. 395–416.

[11] D. Comaniciu, P. Meer, Mean shift: A robust approach toward feature space analysis, IEEE Trans. on Pattern Analysis and Machine Intelligence 24 (5) (2002), pp. 603–619.

[12] P. Franti, T. Kaukoranta, D. Shen, K. Chang, Fast and memory efficient implementation of the exact PNN, IEEE Transactions on Image Processing 9 (5) (2000), pp. 773–777.

[13] Y. Zhao, G. Karypis, Evaluation of hierarchical clustering algorithms for document datasets, in: Proceedings of the Eleventh International Conference on Information and Knowledge Management, 2002, pp. 515–524.

[14] Y. Loewenstein, E. Portugaly, M. Fromer, M. Linial, Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space, Bioinformatics 24/ISBM (13) (2008), pp. 141–149.

[15] B. Everitt, S. Landau, M. Leese, D. Stahl, Cluster Analysis, Wiley & Sons, 2011.

[16] G. Sell, D. Garcia-Romero, Speaker diarization with PLDA i-vector scoring and unsupervised calibration, in: 2014 IEEE Spoken Language Technology Workshop (SLT), 2014, pp. 413–417.

[17] W. Day, H. Edelsbrunner, Efficient algorithms for agglomerative hierarchical clustering methods, Journal of Classification 1 (1) (1984), pp. 7–24.

[18] I. Gronau, S. Moran, Optimal implementations of UPGMA and other common clustering algorithms, Information Processing Letters 104 (6) (2007), pp. 205–210.

[19] J. Benzécri, Construction d'une classification ascendante hieŕarchique par la recherche en chaîne des voisins réciproques, Les Cahiers de l'Analyse des Données 12 (7) (1982), pp. 209–217.

[20] P. Juan, Programme de classification hieŕarchique par l'algorithme de la recherche en chaîne des voisins réciproques, Les Cahiers de l'Analyse des Données 12 (7) (1982), pp. 219–225.

[21] F. Murtagh, A survey of recent advances in hierarchical clustering algorithms, Computer Journal 26 (4) (1983), pp. 354–359.

[22] M. Blum, R. Floyd, V. Pratt, R. Rivest, R. Tarjan, Time bounds for selection, Journal of Computer and System Sciences 7 (4) (1973), pp. 448–461.

[23] J. F. Matias Rodrigues, C. von Mering, HPC-CLUST: distributed hierarchical clustering for large sets of nucleotide sequences, Bioinformatics 30 (2) (2014), pp. 287–288.

[24] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: An efficient data clustering method for very large databases, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1996, pp. 103–114.

[25] B. Leibe, K. Mikolajczyk, B. Schiele, Efficient clustering and matching for object class recognition, in: Proc. of the British Machine Vision Conference (BMVC), 2006, pp. 81.1–81.10.

[26] P. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, Journal of Computational and Applied Mathematics 20 (1) (1987), pp. 53–65.

[27] L. Vendramin, R. Campello, E. Hruschka, Relative clustering validity criteria: A comparative overview, Statistical Analysis and Data Mining 3 (4) (2010), pp. 209–235.

[28] P. Franti, O. Virmajoki, V. Hautamaki, Fast agglomerative clustering using a k-Nearest Neighbor graph, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (11) (2006), pp. 1875–1881.

[29] M. Bruynooghe, Méthodes nouvelles en classification automatique de données taxinomiques nombreuses, Statistique et analyse des données 2 (3) (1977), pp. 24–42.

[30] S. Ioffe, Probabilistic Linear Discriminant Analysis, in: Proceedings of the 9th European Conference on Computer Vision - Volume Part IV, ECCV'06, 2006, pp. 531–542.

[31] P. Kenny, Bayesian speaker verification with Heavy–Tailed Priors, in: Keynote presentation, Odyssey 2010, The Speaker and Language Recognition Workshop, 2010, available at `http://www.crim.ca/perso/patrick.kenny/kenny_Odyssey2010.pdf`.

[32] S. Cumani, P. Laface, Joint estimation of PLDA and non–linear transformations of speaker vectors, IEEE/ACM Transactions on Audio, Speech, and Language Processing 25 (10) (2017), pp. 1890–1900.

[33] S. Cumani, N. Brümmer, L. Burget, P. Laface, O. Plchot, V. Vasilakakis, Pairwise discriminative speaker verification in the i-vector space, IEEE/ACM Transactions on Audio, Speech, and Language Processing 21 (6) (2013), pp. 1217–1227.

[34] S. Cumani, P. Laface, Large scale training of Pairwise Support Vector Machines for speaker recognition, IEEE/ACM Transactions on Audio, Speech, and Language Processing 22 (11) (2014), pp. 1590–1600.

[35] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, P. Ouellet, Front–end factor analysis for speaker verification, IEEE Transactions on Audio, Speech, and Language Processing 19 (4) (2011), pp. 788–798.

[36] S. Cumani, P. Laface, Speaker recognition using e–vectors, IEEE/ACM Transactions on Audio, Speech, and Language Processing 26 (4) (2018), pp. 736–748.

[37] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, S. Khudanpur, X-vectors: Robust DNN embeddings for speaker recognition, in: Proceedings of ICASSP 2018, 2018, pp. 5329–5333.

[38] S. Cumani, P. Laface, Non–linear i–vector transformations for PLDA based speaker recognition, IEEE/ACM Transactions on Audio, Speech, and Language Processing 25 (4) (2017), pp. 908–919.

[39] N. Brümmer, J. A. du Preez, Application-independent evaluation of speaker detection, Computer Speech & Language 20 (2-3) (2006), pp. 230–275.

[40] S. Shum, N. Dehak, R. Dehak, J. R. Glass, Unsupervised speaker adaptation based on the cosine similarity for text–independent speaker verification, in: Proceedings of Odyssey 2010, 2010, pp. 76–82.

[41] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. PéRez, I. Perona, An extensive comparative study of cluster validity indices, Pattern Recognition 46 (1) (2013), pp. 243–256.

[42] W. Rand, Objective criteria for the evaluation of clustering methods, Journal of the American Statistical Association 66 (1971), pp. 846–850.

[43] L. Hubert, P. Arabie, Comparing partitions, Journal of Classification 2 (1) (1985), pp. 193–218.

[44] H. Kim, H. Park, Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis, Bioinformatics 23 (12) (2007), pp. 1495–1502.

[45] D. A. van Leeuwen, Speaker linking in large data sets, in: Proceedings of Odyssey 2010, 2010, pp. 202–208.

SANDRO CUMANI received the Ph.D. in computer and system engineering from Politecnico di Torino, Italy, in 2011. He is a Research Fellow in the Department of Control and Computer Engineering, Politecnico di Torino. His current research interests include machine learning, speech processing and biometrics, in particular speaker and language recognition.

PIETRO LAFACE is full Professor of Computer Science at Politecnico di Torino, Italy, where he leads the speech technology research group. He has published over 150 papers in the area of pattern recognition, artificial intelligence, and spoken language processing. His research interests include all aspects of automatic speech recognition and its applications.