Analysis of Large-Scale SVM Training Algorithms for Language and Speaker Recognition

Analysis of Large–Scale SVM Training Algorithms for Language and Speaker Recognition

Sandro Cumani and Pietro Laface

Abstract

This paper compares a set of large scale Support Vector Machine training algorithms for language and speaker recognition tasks.

We analyze five approaches for training phonetic and acoustic SVM models for language recognition. We compare the performance of these approaches as a function of the training time required by each of them to reach convergence, and we discuss their scalability towards large corpora.

Two of these algorithms can be used in speaker recognition to train a SVM that classifies pairs of utterances as either belonging to the same speaker or to two different speakers.

Our results show that the accuracy of these algorithms is asymptotically equivalent, but they have different behavior with respect to the time required to converge. Some of these algorithms not only scale linearly with the training set size, but are also able to give their best results after just a few iterations.

State–of–the–art performance has been obtained in the female subset of the NIST 2010 Speaker Recognition Evaluation extended core test using a single SVM system.

Index Terms

Support Vector Machines, Language recognition, Speaker recognition, Large–scale training.

I. Introduction

Support Vector Machines (SVM) provide successful discriminative models in the field of language and speaker recognition. The main limitation of SVMs comes from their demands in terms of training time

The authors are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10143 Torino, Italy (e-mail: sandro.cumani@polito.it, pietro.laface@polito.it).

and memory requirements. Fast algorithms for training SVMs have been proposed, often relying on the so–called kernel trick [1], where a kernel matrix is computed and loaded in main memory. The kernel trick not only allows speeding up the computation required to find the solution of the SVM objective function but also makes possible nonlinear classification in very high dimension spaces.

This approach, however, does not scale well because the required amount of memory grows quadratically with the number of training samples. Since several real–world classification tasks, including speaker and language recognition, are performed on large datasets, the allocation of the kernel matrix in main memory becomes unfeasible even for large memory computers. We are thus interested in training approaches with time complexity and memory occupation scaling linearly with the number of training patterns.

In this paper we analyze a set of SVM training algorithms suitable for training large corpora using as testbeds a language recognition task and a speaker recognition task. In particular, we compare the performance of several algorithms in terms of training time, scalability towards large corpora and possibility of multi-threaded implementation.

The algorithms themselves are not strictly novel, and their proposers have performed comparative analyses on large benchmarks [2], [3]. However, this work offers several original contributions of interest to the speaker and language recognition community. In particular, it is the first report, to our knowledge, that is devoted to an in-depth evaluation of large-scale SVM training algorithms both for closed–set (language) and open-set (speaker) classification tasks, using standard corpora, state–of–the–art algorithms and systems, and considering largely different acoustic and phonetic models.

An extension to the Bundle Methods algorithm [3] has been devised to provide the dual solution to the SVM problem, which is needed by the pushed–GMM approach in language recognition [4].

Finally, the most interesting training algorithms have been implemented from scratch and compared with their multi–threaded or multi–process implementation, tailored to the specific task for achieving the fastest training time.

The paper is organized as follows: Section II gives a short overview of the SVM classifier. Section III and IV describe our language and speaker recognition models, respectively, highlighting the large dimensions of the features and of the corpora that have been used. Section V illustrates five different approaches suitable to large scale SVM training. The details of their implementation is given in Section VI. The experimental results are presented and commented in Section VII, and conclusions are drawn in Section VIII.

II. Support Vector Machines

A Support Vector Machine [5] is a binary classifier which estimates the hyperplane that best discriminates two given classes of patterns according to a maximum separation margin criterion. The separation hyperplane is obtained by solving an unconstrained *regularized risk minimization* problem

$$\min_{\mathbf{w}} \left[\frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \sum_{i=1}^{n} l\left(\mathbf{w}, \mathbf{x_i}, y_i\right) \right] \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ denotes a (d–dimensional) training pattern with associated label $y_i \in \{-1, +1\}$. This objective function is the sum of two terms. The second term in (1) is the empirical risk evaluated on the training set and weighted by parameter C. The first term is a regularization contribution, given by the squared $L2$ norm of the separating hyperplane $\mathbf{w}$, related to the generalization capability of the model [5].

The standard loss function for the SVM problem, which gives the maximum soft–margin classifier, is the so–called hinge (L1) loss function

$$l_{L1} = \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i) \tag{2}$$

In the following we will focus on L1–SVMs only, however, different classifiers can be obtained by changing the loss function, for example the squared soft–margin SVM defined by the squared hinge loss (L2) function

$$l_{L2} = \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right)^2 \tag{3}$$

or a classifier based on regularized logistic regression, characterized by the loss function

$$l_{log} = \log\left(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}\right) \tag{4}$$

From (1) and (2) the standard *primal* SVM formulation is

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i) \tag{5}$$

While it is possible to solve the SVM optimization problem in its primal formulation, many approaches prefer to solve the dual problem by estimating the Lagrange multipliers $\boldsymbol{\alpha}^* = (\alpha_1^*, \ldots, \alpha_n^*)^T$ that minimize

the objective function

$$\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H}\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \tag{6}$$
$$\text{subject to} \quad 0 \leq \alpha_i \leq C \qquad \forall i = 1, \ldots, n$$

where n is the number of training patterns, $\mathbf{e}$ is a vector of ones and $\mathbf{H}$ is the matrix of dot–products $H_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$.

Solving the dual problem looking for the Lagrange multipliers $\boldsymbol{\alpha}^*$ allows the class separation hyperplane to be expressed in terms of a linear combination of the training vectors as

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x_i} \tag{7}$$

This formulation shows that the separation hyperplane is a function of a subset of training patterns, namely the ones associated with non null Lagrange multipliers: the so–called Support Vectors.

The classification score of a given pattern is obtained by its dot–product with the separation hyperplane, but it can also be evaluated, as shown in (7), from its dot–products with the set of the support vectors. Non–linear classification can be performed, therefore, through linear classification in a higher–dimensional space described by the mapping $\Phi(x_i)$, provided that we are able to evaluate the dot–products $\Phi(x_i)^T\Phi(x_j)$ in the mapped feature space. No explicit expansion of training patterns is actually necessary because the dual problem (6) can be solved by replacing dot–products $x_i^T x_j$ with kernel evaluations $\Phi(x_i)^T\Phi(x_j)$. This can be accomplished by replacing the entries of matrix $\mathbf{H}$ with the entries of the kernel matrix $\mathbf{K}$, with $K_{ij} = \Phi(x_i)^T\Phi(x_j) = K(x_i, x_j)$, multiplied by the corresponding element of the matrix of labels $Y_{ij} = y_i y_j$ (i.e. $H_{ij} = K_{ij} y_i y_j$). Care has to be taken in the definition of mapping $\Phi(x_i)$ so that kernel $\mathbf{K}$ satisfies Mercer's condition.

III. Language recognition models

In this section we summarize two state–of–the art language recognition approaches that stem from the works [6], [7], and [8]. The task is to verify an unknown utterance according to its language given training utterances for a specified set of possible target languages. Models are estimated for each language, and a test utterance is scored against each of these models.

State–of–the–art systems rely on two main approaches. In the phonetic approach, the language models are estimated collecting statistics of the occurrence of sequences of n phones (n–grams) obtained by running one or more phonetic decoders [6], [8], [9]. The acoustic approach relies, instead, on Gaussian

Mixture Models of acoustic features (GMMs) [7], [10], discriminatively trained [11], or used in combination with SVM classifiers as in [12], [13], [4]. In the latter approach, the models are trained by means of SVMs estimating, for each language, the hyperplane that separates the patterns of the target language from the pooled patterns of all the other (non–target) languages.

Training these models requires to deal with large features, and large–scale datasets as introduced in the next two subsections and detailed in Section VII.

A. Phonetic models

In the phonetic approach, a phonetic decoder estimates the most likely phone transcription or a lattice of phone hypotheses, organized in a graph where each hypothesis has associated its likelihood and time boundaries. The decoder language is not necessarily included in the set of the target languages. The estimated occurrence of the phone n–grams can be obtained from the 1–best transcription or from the lattice phone hypotheses. An utterance is represented by stacking into a vector the estimated frequency of the different n–grams appearing in the utterance.

Since linear classification of unnormalized n–gram counts does not yield good recognition accuracy, a suitable linear kernel has to be used for classification. Many different kernels have been proposed in the last years, the most popular being the Term Frequency Log Likelihood Ratio (TFLLR) kernel [14], [4], which approximates a log–likelihood ratio, computed from n–gram statistics, between the target and the background set. Since TFLLR kernel is linear, it can be expressed as a normalization of n–gram features

$$\hat{\mathbf{x}}^i = \sqrt{\frac{1}{f_i}}\mathbf{x}^i, \quad f_i = \frac{1}{n}\sum_{k=1}^{n}\mathbf{x}_k^i \tag{8}$$

where superscript i denotes the i–th component of pattern $\mathbf{x}$.

In the experiments reported in this paper, a single phonetic decoder has been used, i.e. the Loquendo–ASR recognizer for Italian language [15], which generates a lattice of hypotheses. We estimate n–gram statistics up to the third order from these hypotheses leading to a 44135–dimensional feature space. Not only the patterns have high dimensions, but also the size of the corpora is constantly increasing with more demanding applications or variety of languages and recording conditions. As detailed in Section VII, in our experiments we have 20543 utterances for training the phonetic models of 23 languages.

B. Acoustic models

The acoustic models that we have trained combine generative GMM models with SVMs, in the so–called pushed–GMM approach [13], [4], where the discriminative GMM models are derived from the

generative ones, by exploiting the information provided by the non–null Lagrange multipliers obtained by training a SVM. In particular, a probabilistic model of frame–level acoustic features is built based on GMMs. For each utterance, the distribution of acoustic features is estimated by Maximum A Posteriori adaptation from a common Universal Background Model (UBM) using a low relevance factor [7]. Starting from these generative models, a single vector representation for each utterance is obtained by stacking the GMM mean vectors into the *supervector* $\mathbf{g}$

$$\mathbf{g} = \left[\boldsymbol{\mu}_1^T \boldsymbol{\mu}_2^T \dots \boldsymbol{\mu}_m^T\right]^T \tag{9}$$

where $\boldsymbol{\mu}_i$ represents the mean of the i-th component of the mixture.

A possible linear kernel for GMM supervectors is the KL kernel [16]

$$K(\mathbf{g}_a, \mathbf{g}_b) = \sum_{i=1}^{m} \left(\sqrt{c_i}\boldsymbol{\Sigma}_i^{-\frac{1}{2}}\boldsymbol{\mu}_{ai}\right)^T \left(\sqrt{c_i}\boldsymbol{\Sigma}_i^{-\frac{1}{2}}\boldsymbol{\mu}_{bi}\right) \tag{10}$$

where c_i and $\boldsymbol{\Sigma}_i$ are the weight and the covariance matrix of the i–th Gaussian. This kernel is derived from the Kullback–Leibler divergence between two GMMs and adapted in order to satisfy Mercer's condition.

While the class separation hyperplane could be used to perform classification of unknown utterances as in the phonetic system approach (GMM–SVM) [12], better results have been obtained by combining the generative and the discriminative systems [4]. In particular, SVM classification is first performed using the GMM-SVM approach, then two GMMs are created for each target language: one for the language $\mathbf{g}^+$ and the other for the non–target languages (anti–model) $\mathbf{g}^-$ according to

$$\mathbf{g}^+ = \frac{1}{\sum_{i|y_i>0}\alpha_i} \sum_{i|y_i>0} \alpha_i \mathbf{g_i} \tag{11}$$

$$\mathbf{g}^- = \frac{1}{\sum_{i|y_i<0}\alpha_i} \sum_{i|y_i<0} \alpha_i \mathbf{g_i} \tag{12}$$

where $\mathbf{g_i}$ is the GMM of the i–th utterance of the target language. Thus, the target model is a weighted combination of the GMMs belonging to the target language, and the weights are the Lagrange multipliers $\boldsymbol{\alpha}$ obtained from the dual solution of the SVM problem. The target language anti–model is a weighted combination of the GMMs of the non–target languages utterances.

It is worth noting that these weights are needed even if we solve the SVM optimization in its primal formulation.

In our experiments, detailed in Section VII-A, 2048–Gaussians mixtures are used for the UBM and

the language GMMs, while the feature space is 56–dimensional. In this configuration a supervector has 114688 components, and the training dataset includes the same 20543 utterances provided for training the phonetic system.

IV. Speaker recognition models

In speaker recognition the task is to verify whether the identity of the speaker in a given test utterance corresponds to a target speaker, given one or more enrollment utterances for the target speaker. Several successful approaches have been proposed in the last years based on GMM supervectors [7], [17], [18]. A discriminative approach using GMM supervectors as features for training SVMs has also been proposed for this task in [16]. The problem with this paradigm is that it estimates a speaker model using the few utterances (often just one) that are usually available for the target speaker.

In the following we outline an alternative discriminative approach that tries to overcome this problem by changing the recognition paradigm. Rather than modeling the speaker classes, a binary classifier is trained to classify a pair of utterances as either belonging to the same speaker or to two different speakers [19], [20], [21].

This approach has been made feasible by a recently introduced compact representation of spoken utterances, the i–vectors [22], [23]. An i–vector is a low–dimensional (few hundred components) representation of a spoken utterance, which can be estimated using the same techniques introduced for Joint Factor Analysis [18]. These features, much smaller than supervectors, have made possible training Probabilistic Linear Discriminant Analysis (PLDA) generative models [24], [25], leading to state–of–the–art speaker recognition systems. The goal of such systems is to model the underlying distribution of the speaker and of the channel components of the i–vectors in a Bayesian framework. From these distributions it is possible to evaluate the likelihood ratio between the “same speaker” hypothesis and “different speakers” hypothesis for a pair of i–vectors.

The same paradigm can be used to train discriminative systems where the observation patterns are pairs of i–vectors. In this approach a non–linear SVM model is built to discriminate between ”same speaker pair“ (”target“ class) and ”different speaker pair“ (”non–target“ class). The feature expansion is derived from the PLDA model. The evaluation of log–likelihood ratios between the same–speaker ($\mathcal{H}_s$) and different–speaker ($\mathcal{H}_d$) hypotheses can be expressed, for Gaussian PLDA models, as a quadratic form of

the i–vector pair (ϕ_1, ϕ_2) as

$$\begin{aligned} s(\phi_1, \phi_2) &= \log \frac{P(\phi_1, \phi_2|\mathcal{H}_s)}{P(\phi_1, \phi_2|\mathcal{H}_d)} \\ &= \phi_1^T \mathbf{\Lambda} \phi_2 + \phi_2^T \mathbf{\Lambda} \phi_1 + \phi_1^T \mathbf{\Gamma} \phi_1 + \phi_2^T \mathbf{\Gamma} \phi_2 + \\ &\quad (\phi_1 + \phi_2)^T \mathbf{c} + k \end{aligned} \tag{13}$$

where $\mathbf{\Lambda}$, $\mathbf{\Gamma}$, $\mathbf{c}$ and k are directly derived from PLDA model parameters estimated on a training set. The same parameters can be interpreted as the components of a linear hyperplane in a higher dimensional feature space [20], [21], by stacking the PLDA parameters in a vector as

$$\mathbf{w} = \begin{bmatrix} vec(\mathbf{\Lambda}) \\ vec(\mathbf{\Gamma}) \\ \mathbf{c} \\ k \end{bmatrix} \tag{14}$$

where $vec(\mathbf{A})$ is the operator which stacks the columns of matrix $\mathbf{A}$ into a column vector. The non–linear feature mapping of the i–vector pair (ϕ_1, ϕ_2) given by

$$\Phi(\phi_1, \phi_2) = \begin{bmatrix} vec(\phi_1 \phi_2^T + \phi_2 \phi_1^T) \\ vec(\phi_1 \phi_1^T + \phi_2 \phi_2^T) \\ \phi_1 + \phi_2 \\ 1 \end{bmatrix} \tag{15}$$

allows the pair score $s(\phi_1, \phi_2)$ to be computed as a dot–product between the hyperplane $\mathbf{w}$ and the expanded pattern $\Phi(\phi_1, \phi_2)$,

$$s(\phi_1, \phi_2) = \mathbf{w}^T \Phi(\phi_1, \phi_2) \tag{16}$$

The pairwise SVM approach estimates the parameter $\mathbf{w}$ in order to minimize the SVM loss function. In this approach the number of pairs grows quadratically with the number of the training i–vectors. This would make unmanageable training a SVM due to memory and time constraints. Moreover, the dimensionality of the expanded feature space grows quadratically with the size of the i–vectors. Thus the solution cannot rely on the explicit evaluation of the feature mapping to generate the training dataset for the SVM, because the database size would grow as $O(n^2 d^2)$, where d is the i–vector dimension. Although the i–vector dimension is of the order of hundreds, the total memory required would be really huge because the number of training i–vectors n easily reaches tens of thousands. Online feature mapping

would be, instead, computationally expensive because it has to be performed for each iteration of the training algorithm. In [20] a memory and time efficient approach to solve this problem has been described, which allows the SVM loss function and its gradient to be computed in $O(n^2d)$.

In the next section we present several SVM training algorithms for these large–scale language and speaker recognition tasks, whereas in Sections VI and VII we will analyze their feasibility, implementation details, and performance.

V. Large–scale Support Vector Machines

The ever–increasing size of the training corpora for real–world classification tasks makes it impractical to solve the SVM problem resorting to medium–scale techniques which assume that the entire dataset can be stored in main memory. We have shown in the previous sections that a large number of high dimensional patterns are needed to train speaker and language recognition systems. We are thus interested in SVM training approaches with time complexity scaling linearly with the number of the training patterns, and with reasonable memory requirements.

Many algorithms have been proposed to handle SVM optimization for large–scale problems, most of them are efficient only for *linear* kernel SVMs. In this section we present five algorithms, focusing on their complexity and possible speedup by means of threaded implementations.

A. Dual solvers

We first analyze three dual SVM problem solvers, whereas in Section V-B we will describe two primal solvers and the steps to derive their corresponding dual solutions. In the following, C denotes the regularization parameter of the SVM and ε denotes the optimization accuracy.

1) SVM^{Light}: A popular "fast" linear–space SVM solver is SVM$^{\text{Light}}$ [26]. The performance of the models trained with this solver are our reference in this work. SVM$^{\text{Light}}$ decomposes the SVM problem into a set of subproblems and iteratively optimizes such subproblems. Its memory occupation scales linearly with the number of training patterns and of support vectors. Since SVM$^{\text{Light}}$ solves the dual problem it provides the Lagrange multipliers needed in language recognition by the pushed–GMM approach.

The main limitation of this algorithm comes from its time complexity, which has been empirically shown to be $O(n^2d)$. If memory is not a constraint, a fast implementation of SVM$^{\text{Light}}$ can be obtained by caching all the kernel evaluations. Of course, the kernel matrix has a size, and thus a computational cost, which still grows quadratically with the training set size.

2) Dual Coordinate Descent: The dual problem is solved in [2] by means of a coordinate descent approach, referred to in the following as Dual Coordinate Descent Method (DCDM). The multivariate problem is split into a sequence of univariate optimizations which are iteratively solved until convergence to the optimal multivariate solution is attained. Assuming that a sub–optimal solution $\boldsymbol{\alpha}$ of the dual problem (6) is known, the i–th component of the optimal solution, given all the other coordinates, can be evaluated by solving

$$\min_h f(\boldsymbol{\alpha} + h\mathbf{e}_i) \quad \text{subject to } 0 \leq \alpha_i + h \leq C \tag{17}$$

The objective function is quadratic in h

$$f(\boldsymbol{\alpha} + h\mathbf{e}_i) = \frac{1}{2}H_{ii}h^2 + \nabla_i f(\boldsymbol{\alpha}) + K \tag{18}$$

for a given constant K. The minimization of this function leads to the update rule [2], [27]

$$\alpha_i \leftarrow \min\left[\max\left(\alpha_i - \frac{\nabla_i f(\boldsymbol{\alpha})}{H_{ii}}, 0\right), C\right] \tag{19}$$

where the gradient $\nabla_i f(\boldsymbol{\alpha})$ is

$$\nabla_i f(\boldsymbol{\alpha}) = \sum_{j=1}^{n} H_{ij}\alpha_j - 1 \tag{20}$$

The computation of the gradient is in general expensive, but for linear SVMs simplifies as

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{w}^T \mathbf{x}_i - 1 \tag{21}$$

The cost of evaluating $\mathbf{w}$ given $\boldsymbol{\alpha}$ would be linear with the size of the training set. However, by keeping the previous value of $\mathbf{w}$ it can be updated according to

$$\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \alpha_i^{old}) y_i \mathbf{x}_i \tag{22}$$

where α_i^{old} refers to the value the parameter α_i had before being updated.

Since DCDM solves the dual formulation of the SVM problem, it directly provides the Lagrange multipliers required by the pushed–GMM approach.

The time complexity of the algorithm is $O\left(nd\log\left(\frac{1}{\varepsilon}\right)\right)$. DCDM is very fast, however it cannot take advantage of a distributed environment because the solution is updated after each pattern is processed.

3) SVM^{Perf}: One of the most effective linear–time SVM solver is SVM$^{\text{Perf}}$ [28], [29], [30]. Though the package provides different algorithms for solving the SVM problem, its main innovation is the Cutting–Plane Subspace–Pursuit (CPSP) approach [30]. Cutting–Plane algorithms are based on a different

formulation of the primal problem (5)

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi \\ \text{subject to} \quad & \forall \hat{y_1} \dots \hat{y_n} \in \{-1, +1\} : \\ & \frac{1}{2}\mathbf{w}^T \left[\sum_{i=1}^{n} (y_i \mathbf{x}_i - \hat{y_i}\mathbf{x}_i) \right] \geq \sum_{i=1}^{n} \Delta(y_i, \hat{y_i}) - \xi \end{aligned} \tag{23}$$

where $\Delta(y, \hat{y})$ is the zero–one loss function, which takes value 1 when its arguments are equal, and value 0 otherwise.

The solution is found by iteratively building a working set of constraints over which a Quadratic Problem (QP) is solved. An accuracy of ε can be obtained using at most $O\left(\frac{1}{\varepsilon}\right)$ constraints. The CPSP algorithm modifies the traditional Cutting–Plane algorithm by iteratively building a set of *basis vectors* $\{\mathbf{b}_1, \dots \mathbf{b}_k\}$ whose span is approximatively the sub–space where the optimal solution lies. The approximate solution is thus given by

$$\mathbf{w}^* \approx \sum_{i=1}^{k} \beta_i \mathbf{b}_i \tag{24}$$

The rationale for the introduction of the basis vectors is to reduce the number of kernel evaluations for non–linear classification. Since basis vectors are associated with the Cutting Plane constraints, which are supposed to be a constant number with respect to the training set, the hyperplane can be represented similarly to (7) but using a much smaller set of patterns. It is worth noting that since the basis vectors do not belong to the training set, it is not possible to exploit the possible speedup coming from the pre-computation of the kernel matrix. Moreover, computing the kernel matrix is unfeasible for large datasets.

Due to the nature of the algorithm, it can be easily modified to be executed in a distributed environment.

B. Primal solvers

In this section we describe two primal solvers and the steps to derive their corresponding dual solutions, which are necessary for the pushed–GMM approach in language recognition.

1) Pegasos: The first solver is based on gradient descent in the primal solution space. Standard gradient descent techniques try to reach the minimum of the objective function by iteratively moving an approximate solution along the direction that gives the greatest decrease of the objective function. For

the L1–loss function this leads to the update rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left[\mathbf{w}_t + C \sum_{i=1}^{n} \nabla_{\mathbf{w}} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)|_{\mathbf{w}_t} \right] \tag{25}$$

where η_t is the learning rate at iteration t. The selection of the learning rate values is crucial for fast convergence of the algorithm. Since the L1–loss function is not completely differentiable but still convex, a *subgradient* of the loss function can be computed as

$$\nabla_{\mathbf{w}} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i) = \begin{cases} -y_i & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

Stochastic Gradient Descent (SGD) approximates the gradient computation step by evaluating the subgradient of the objective function on a pattern (or on a small subset of patterns)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left[\mathbf{w}_t + nC \nabla_{\mathbf{w}} l(\mathbf{w}, \mathbf{x}_{i_t}, y_{i_t})|_{\mathbf{w}_t} \right] \tag{27}$$

where i_t is chosen randomly for each iteration.

Pegasos [31] combines SGD for the SVM L1–loss function with a projection step ensuring faster convergence to the optimal solution. A set of training patterns $\mathcal{A}_t$ is randomly chosen at each iteration. The subgradient of the objective function is estimated from this subset as

$$\nabla_t = \mathbf{w}_t - \frac{nC}{|\mathcal{A}_t|} \sum_{\substack{i | \mathbf{x}_i \in \mathcal{A}_t \\ y_i \mathbf{w}^T \mathbf{x}_i < 1}} y_i \mathbf{x_i} \tag{28}$$

and the hyperplane is updated by

$$\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla_t \tag{29}$$

The optimal SVM solution is bounded by $\|\mathbf{w}\| \leq \sqrt{nC}$ [31]. Thus, the current solution is projected onto a ball of radius $\sqrt{nC}$ by scaling $\mathbf{w}_{t+1/2}$ according to

$$s_t = \min \left\{ 1, \frac{\sqrt{nC}}{\|\mathbf{w}_{t+1/2}\|} \right\} \tag{30}$$

$$\mathbf{w}_{t+1} = s_t \mathbf{w}_{t+1/2} \tag{31}$$

This projection step, combined with a fast–decaying learning rate, allows bounding to $O\left(\frac{1}{\varepsilon}\right)$ the average number of iterations required to achieve ε optimization accuracy .

Since Pegasos solves the primal formulation of the SVM problem it does not produce the Lagrange

multipliers, which are necessary in the pushed–GMM approach. In [31] the authors propose an extension of their algorithm that allows the hyperplane to be estimated as a linear combination of training patterns $\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$ where the set of $\boldsymbol{\alpha}$'s are iteratively obtained as

$$\boldsymbol{\alpha}_{t+1} = \left[\boldsymbol{\alpha}_t - \eta_t \left(\boldsymbol{\alpha}_t + \frac{nC}{|\mathcal{A}_t|}\chi_t^i\right) s_t\right] \tag{32}$$

with

$$\chi_t^i = \begin{cases} 1 & \text{if } x_i \in \mathcal{A}_t \\ 0 & \text{if } x_i \notin \mathcal{A}_t \end{cases} \tag{33}$$

Since Pegasos is based on stochastic gradient descent, which performs sequential updates, it cannot take advantage of a distributed environment.

2) Bundle Methods: Bundle methods approximate a convex function by means of a set of tangent hyperplanes (subgradients) and solve the simpler optimization problem on the approximated function. The approach is similar to SVM$^{\text{Perf}}$, where a small and incremental subset of constraints is built until the solution approximates the optimal solution up to a given error. Bundle Methods for Regularized Risk Minimization (BMRM) [32], [3] offer a general and easily extensible framework to general risk regularization problems, of which SVM is an example. In particular, an incremental working set of approximate solutions $\{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \dots\}$ is built by defining, at each iteration, the set of hyperplanes which are tangent to the objective function in the working set points (starting from $\mathbf{w}_0 = \mathbf{0}$)

$$f_t(\mathbf{w}) = l_{emp}(\mathbf{w}_t) + \nabla l_{emp}(\mathbf{w}_t) \cdot (\mathbf{w} - \mathbf{w}_t) \tag{34}$$

where $l_{emp}(\mathbf{w}) = \sum_{i=1}^{n} l(\mathbf{w}, \mathbf{x_i}, y_i)$ is the empirical loss. At each iteration, a new working point is selected as the minimizer of the approximated function

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \left[\frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \max\left(0, \max_{t' \leq t+1} f_{t'}(\mathbf{w})\right)\right] \tag{35}$$

In [32] it is shown that this problem is equivalent to the dual quadratic problem

$$\begin{aligned} \min_{\beta} D_i(\beta) \quad &= \tfrac{C}{2}\beta^T A^T A\beta - \beta^T \mathbf{b} \\ \text{subject to} \quad &\beta \geq \mathbf{0}, \quad \mathbf{e}^T\beta \leq 1 \end{aligned} \tag{36}$$

where A is the matrix $[a_1 a_2 \dots a_i]$ of gradients $a_{t+1} = \nabla l_{emp}(\mathbf{w}_t)$ and $\mathbf{b}$ is the vector $[b_1 b_2 \dots b_i]^T$ of

offsets $b_{t+1} = l_{emp}(\mathbf{w}_t) - a_{t+1}^T \mathbf{w}_t$, and the new solution is obtained as

$$\mathbf{w_{t+1}} = -CA\beta \tag{37}$$

This quadratic problem is not expensive because its complexity does not increase with the size of the training dataset, but with the number of iterations only.

The BMRM algorithm does not directly provide the Lagrange multipliers for the dual SVM problem. Thus, we devised a method for evaluating an approximation of the $\boldsymbol{\alpha}$ values.

The L1 loss function can be rewritten to make explicit its dependency on the dot product between $\mathbf{w}$ and a given pattern $\mathbf{x}$ as

$$l(\mathbf{w}, \mathbf{x}, y) = \widetilde{l}(\mathbf{w}^T \mathbf{x}, y) \tag{38}$$

and its gradient with respect to $\mathbf{w}$ as

$$\nabla_{\mathbf{w}} l(\mathbf{w}, \mathbf{x}, y) = \frac{\partial \widetilde{l}(\mathbf{w}^T \mathbf{x}, y)}{\partial \mathbf{w}^T \mathbf{x}} \mathbf{x} \tag{39}$$

Defining an array $\widetilde{a}_t = [\widetilde{l}_1 \widetilde{l}_2 \dots \widetilde{l}_n]^T$, where $\widetilde{l}_i = \frac{\partial \widetilde{l}(\mathbf{w}^T \mathbf{x_i}, y_i)}{\partial \mathbf{w}^T \mathbf{x}}$, we can express a_t as

$$a_t = \mathbf{X} \widetilde{a}_t \tag{40}$$

where $\mathbf{X}$ is the complete set of training patterns represented as a matrix. Matrix A in (36) can then be evaluated as $A = \mathbf{X}\widetilde{A}$ with $\widetilde{A} = [\widetilde{a}_1 \widetilde{a}_2 \dots \widetilde{a}_t]$, and (37) can be rewritten as

$$\mathbf{w_{t+1}} = -C\mathbf{X}\widetilde{A}\beta \tag{41}$$

Setting $\boldsymbol{\alpha} = -CY^{-1}\widetilde{A}\beta$, where Y is the diagonal matrix of the target labels $Y_{ii} = y_i$, allows obtaining the separation hyperplane in terms of a linear combination of the training patterns $\mathbf{w} = \mathbf{X}Y\boldsymbol{\alpha} = \sum_i y_i \mathbf{x}_i \alpha_i$ as in (7).

The BMRM algorithm converges to its optimal solution up to the accuracy ε in $O\left(\frac{1}{\varepsilon}\right)$ iterations. Usually the number of required iterations is small, thus the time required to solve the sub–problems (36) can be neglected, and the global complexity of the algorithm is $O\left(\frac{nd}{\varepsilon}\right)$.

Similarly to SVM$^{\text{Perf}}$, BMRM incrementally builds a working set of approximate solutions, thus its algorithm can be easily modified to run in a distributed environment.

VI. SVM ALGORITHMS IMPLEMENTATION

SVM training for language and speaker recognition was performed using tailored or new implementations of the algorithms presented in the previous section.

The DCDM, BMRM and Pegasos algorithms have been implemented from scratch using a Python/C framework, where the modules requiring expensive computations are either written in C language or are evaluated by means of fast and parallelized NumPy/BLAS functions [33]. Due to the overhead that is introduced by the mixed Python/C framework, training is performed in "bunch" mode, i.e. by loading into main memory and processing a fraction of the dataset. In particular, Python prepares and loads the patterns, while the computation intensive tasks are performed by fast external libraries (NumPy/BLAS) and C code. Thus large bunches minimize the communication overhead between Python and the library routines. Memory occupation of our algorithms is essentially determined by the size of the bunches, which can be limited by the user, possibly to a single pattern in a different implementation.

A. Algorithms for language recognition

For language recognition, all the algorithms, with the exception of SVM$^{\text{Light}}$, have been implemented to jointly train all the language models in parallel in order to minimize the disk accesses. SVM$^{\text{Light}}$ has been modified to effectively compute by means of multi threaded NumPy/BLAS libraries the kernel matrix, and to cache it in memory. Thus, all the language models share the same kernel matrix computed just once. This approach is faster than caching kernel computations on the fly.

DCDM has been implemented as illustrated in [2]. The shrinking technique [26] has been exploited to further speedup the execution time. Shrinking tries to reduce the quadratic problem size by ignoring a subset of the bounded variables, thus reducing the number of dot products that have to be computed, with no impact on the classification performance.

Since all language models are trained simultaneously, a multi–threaded implementation (referred in the following as MTDCDM) has also been developed, where each thread is devoted to training a single language model. However, the obtained speedup is not relevant due to the sequential nature of the algorithm.

The BMRM algorithm has been implemented as described in Section V-B2. The quadratic problem (36) is solved by means of the CVXOPT Python solver [34]. The shrinking technique was not adopted in the BMRM algorithm, which solves the problem in its primal formulation, thus each iteration of the algorithm performs a full scan of the training dataset. The α's needed for the pushed–GMM approach have been evaluated as outlined in Section V-B2.

Since, at each iteration, the gradient computations can be performed in parallel, a distributed version of BMRM has been implemented (referred to as MPBMRM in the following) where the dataset is split among different processes, and each process exploits the multi–threading capabilities of the NumPy/BLAS functions.

Pegasos has been slightly modified with respect to the algorithm presented in Section V-B1. In order to reduce the number of accesses to secondary memory, training is performed in epochs. At the beginning of each epoch a block of randomly selected training patterns is loaded in memory, and the classification hyperplane update is performed using only the patterns belonging to this subset. Due to its sequential update rule we did not implement a multi–threaded version of Pegasos.

Finally, SVM$^{\text{Perf}}$ was modified to fit the format of the other implementations and to read the training patterns on demand rather than loading all the dataset in main memory. A multi–threaded implementation (referred to as MTSVM$^{\text{Perf}}$ in the following) was obtained by modifying the original code with the OpenMP [35] instructions that allow core computations to be parallelized.

B. Algorithms for speaker recognition

The feasibility of training large datasets using the discriminative i–vector pairs approach for speaker recognition described in Section IV requires some additional considerations. Since patterns are pairs of i–vectors, the number of the training trials grows as $O(n^2)$. The feature mapping described in Section IV produces mapped features having $O(d^2)$ components, thus the global dataset size is $O(n^2d^2)$. Caching the complete kernel matrix is impractical even for relatively small sized datasets because it would require $O(n^4)$ memory.

SVM training of the i–vector pairs by means of SVM$^{\text{Light}}$ is not viable because, as shown in Section V-A1, its time complexity is $O(n^4d)$.

In DCDM the hyperplane is updated for each pattern, thus it requires either the complete dataset of mapped features ($O(n^2d^2)$ memory) or online feature mapping ($O(n^2d^2)$ operations for each iteration).

Since in our experiments $d = 400$ and n is approximately 20000, these dual algorithms cannot be used to train the models for our discriminative approach.

Training is feasible, instead, by using primal solvers such as BMRM and Pegasos because it is possible to efficiently evaluate the loss function and its gradient with respect to $\mathbf{w}$ over appropriate subsets of trials. Given a subset $\mathcal{A}$ including m training i–vectors, the loss function and its gradient with respect to $\mathbf{w}$ over all trials, consisting of pairs of i–vectors in $\mathcal{A}$, can be evaluated in $O(m^2d + md^2)$ using the

strategy presented in [20] without the need to compute the mapping of the i–vectors. For BMRM, the set $\mathcal{A}$ includes all the training i–vectors.

Pegasos has been slightly modified: instead of randomly picking a bunch of trials among all possible trials, at each iteration all the i–vector instances of a set of random speakers are selected to build the set $\mathcal{A}$, and the hyperplane is adapted according to the approximation of the gradient evaluated on this set of trials only. This reduces the risk of random selecting "different speaker pair" trials only.

Due to the small size of the i–vectors, the dataset of training utterance can easily be loaded in main memory. The evaluation of loss functions and gradients in these algorithms requires matrix–by–matrix multiplications of large matrices ($n * n$), however it is not necessary to store the complete matrices in main memory because the computations can be performed through block decomposition of the matrices.

As reported in the next section, using this discriminative approach we reach state–of–the–art results for the NIST SRE 2010 extended core condition [36].

Finally, SVM$^{\text{Perf}}$ complexity theoretically grows linearly with the number of patterns, and a solution similar to the BMRM one can be devised for online evaluation of kernels (dot–products with the mapped basis vectors). However, we did not implemented a solution similar to the BMRM one because SVM$^{\text{Perf}}$ does not allow us to easily control class balancing without replicating copies of the i–vectors, which would further increase the size of the dataset.

VII. Experiments

In this section we compare the performance of the described algorithms by training language models for the closed–set NIST Language Recognition Evaluation 2009 (LRE-09) [37] and for the extended tel-tel condition (condition 5) of the NIST Speaker Recognition Evaluation 2010 (SRE-10) [36] tasks. The reported results for the speaker recognition task refer to the more difficult subset of the female speakers.

A. Language recognition task

The LRE-09 core condition task consists in the detection of the language of given test segments among a set of 23 possible target languages including accented languages such as American English and Indian English. Three test conditions have been defined according to the nominal utterance duration (30s, 10s and 3s). Test data consist of both Conversational Telephone Speech (CTS) and telephone bandwidth broadcast radio speech segments. Achieving state–of–the–art recognition performance would require the combination of different acoustic and phonetic systems, possibly trained taking into account that the recordings were collected through two different channels (CTS and narrowband broadcast). However,

since the aim of this work is to compare the relative performance of the SVM classifiers, we have trained just a single–decoder channel–independent phonetic system and a single channel–independent acoustic system.

The training set for the acoustic system consists of 17521 utterances taken from the Callfriend corpus, the corpora provided by NIST for the 2003, 2005 and 2007 Language Recognition Evaluations, the Russian through switched telephone network, the OGI corpus, and the Voice of America corpora. References for these data and the details on the selection process for training patterns are given in [38]. In our experiments we use 2048–Gaussians mixtures for the UBM and the language GMMs. The acoustic features are 7 Mel frequency cepstral coefficients and their 7–1–3–7 Shifted Delta (SDC) coefficients [10], 56 acoustic features in total, compensated for nuisances in the feature domain as in [39].

The phonetic system is trained using the same dataset, though 2005 LRE utterances were split into chunks of approximately 30s. This results in 20543 training utterances.

The first issue in training SVM classifiers is class balancing. It can be faced by appropriately filtering the dataset, or by replicating the patterns of the less populated classes, or even better, by giving different weights to the loss function of patterns belonging to different classes. The first method is not attractive because it reduces the amount of training patterns. The second approach increases the secondary memory accesses, and makes difficult jointly training the different language models because language dependent datasets have to be generated from the full dataset. We trained all the SVMs using the third method of class balancing, except for SVM$^{\text{Perf}}$, which does not provide a simple and direct way to apply this technique.

The second issue is the selection of an appropriate value for the regularization parameter C. In our experiments it has been set according to the default value provided by our reference solver SVM$^{\text{Light}}$. It is estimated as $C = \left(\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^T\mathbf{x}_i\right)^{\frac{1}{2}}\right)^{-2}$ and has proven to produce good models and reasonable results in a large variety of experiments.

System performance is presented in terms of Equal Error Rate (EER) and average cost C_{avg} as defined by NIST [37]. Scores are normalized by means of a Gaussian back–end trained on a held–out set [40]. The performance is given as a function of time by testing models obtained after a variable number of iterations. Timings were evaluated on a HP DS160G5 server equipped with two Xeon X5472 3 GHz quad-core processors, 32 GB of DDR2-800 RAM and a SATA 7200 RPM hard disk. All results are given in terms of wall clock time.

1) Phonetic system: In this section we compare the behavior of the language models trained by means of the techniques illustrated in Section V using phonetic features.

TABLE I: PHONETIC SYSTEM: ASYMPTOTIC VALUES FOR C_{avg} AND EER

Algorithm	30s	10s	3s
SVM^{Light}	0.0375 3.972%	0.0858 8.881%	0.2037 20.772%
BMRM	0.0375 3.941%	0.0860 8.981%	0.2032 20.842%
DCDM	0.0376 3.965%	0.0861 8.948%	0.2031 20.785%
SVM^{Perf}	0.0434 4.583%	0.0944 9.782%	0.2061 21.057%
Pegasos	0.0392 4.198%	0.0879 9.116%	0.2032 20.979%

TABLE II: PHONETIC SYSTEM: TIME REQUIRED TO ACHIEVE 1% SVM^{Light} C_{avg} ACCURACY ("-" MEANS NOT REACHED)

Algorithm	30s	10s	3s
SVM^{Light}	6991s	6991s	6991s
BMRM	6672s	5563s	1598s
MPBMRM	1493s	1148s	210s
DCDM	184s	143s	96s
MTDCDM	85s	66s	46s
SVM^{Perf}	-	-	-
$MTSVM^{Perf}$	-	-	-
Pegasos	-	-	3364s

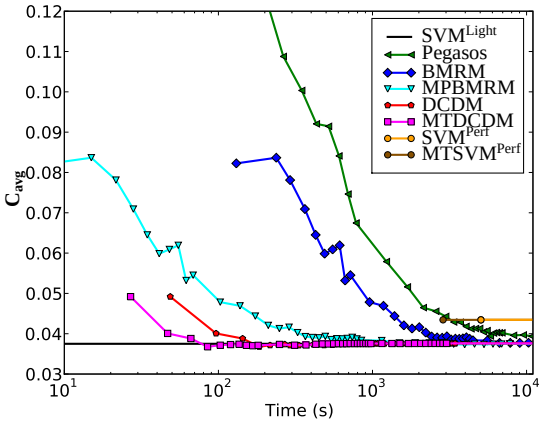


Fig. 1: Phonetic system: C_{avg} as a function of the training time for the 30s condition

SVM^{Light} and SVM^{Perf} models are tested only after convergence because training SVM^{Light} is slow and we are interested only in its classification accuracy that we consider our reference. As far as SVM^{Perf} is concerned, it is less attractive for our applications due to the difficulties in class balancing, which led to

worse recognition results as reported in the following.

Our first set of results shows the asymptotic classification performance of the different algorithms, i.e. the results obtained using a model estimated after a high number of iterations. Table I compares the accuracy of the different phonetic models in the 30, 10, and 3 sec conditions of the NIST LRE09 tests. Both DCDM and BMRM models converge, in terms of C_{avg} and EER, to the results provided by our baseline SVM^{Light}. Models trained with Pegasos give slightly worse performance, and SVM^{Perf} does not produce models as good as the other approaches due to the lack of class balancing.

Fig. 1 shows, in logarithm scale and for the 30s condition, the time required by each algorithm to reach convergence to the reference C_{avg}. The convergence properties of the different technique can be appreciated looking at Table II, which reports the time required by each algorithm to estimate a model that reaches an accuracy within 1.0% of the C_{avg} value obtained using a model trained by SVM^{Light}. The conditions which did not reach the desired accuracy at the end of training are denoted by "-". The same trend has been obtained for the EER and for all the segment durations (30s, 10s and 3s).
Considering the single–threaded implementations, we can observe that, in the 30s condition, DCDM is the fastest algorithm: it reaches more than satisfactory results in just 184 seconds, after 5 iterations. The other solvers are much slower, and BMRM is faster and slightly better than Pegasos.

The parallel implementation of DCDM and BMRM takes into account the different characteristics of the two approaches. In particular, 23 threads are used for MTDCDM (each thread is devoted to training the model of one language) whereas MPBMRM computations are distributed among 4 processes, each one using 8 threads for NumPy/BLAS operations. As expected, DCDM cannot take much advantage of the increased number of cores, being intrinsically sequential, whereas BMRM benefits from multiple processes because training is done in batch mode. It takes also advantage of multi–threading for the NumPy/BLAS operations. For this dataset, single–thread DCDM has faster convergence rate than multi–process BMRM. Obviously, massive parallelization of BMRM could easily outperform DCDM for larger datasets.

2) Pushed–GMM system: The pushed–GMM models are evaluated by reporting their performance as a function of the training time, with the exception of the baseline SVM^{Light}. Since testing the acoustic models is expensive, fewer models were trained and tested compared to the phonetic ones. We recall again that SVM^{Perf} cannot be used for the pushed–GMM approach because it produces basis vectors rather than support vectors. Thus it cannot be used to evaluate the required models and anti–models.

The results are presented in Table III, and in Fig. 2 for the 30s condition. The three entries associated to each algorithm in Table III refer to C_{avg}, EER and time required to achieve 1% SVM^{Light} C_{avg}, respectively.

TABLE III: PUSHED–GMM: ASYMPTOTIC VALUES FOR C_{avg} AND EER, AND TIME REQUIRED TO ACHIEVE 1% SVM^{Light} C_{avg} ACCURACY

Algorithm	30s	10s	3s
SVM^{Light}	0.0276 3.120% 2871s	0.0621 6.592% 2871s	0.1616 16.594% 2871s
BMRM	0.0276 3.153% 1194s	0.0626 6.587% 723s	0.1618 16.579% 311s
DCDM	0.0276 3.157% 655s	0.0626 6.596% 655s	0.1619 16.592% 121s
Pegasos	0.0272 3.112% 8433s	0.0617 6.543% 1800s	0.1605 16.623% 938s

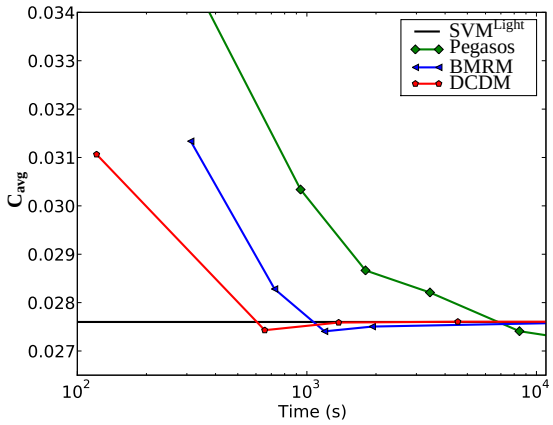


Fig. 2: Pushed–GMM: C_{avg} as a function of the model training time for the 30s condition

Since a single SVM^{Light} model is trained for the three test conditions, and the model is tested only after convergence has been achieved, the times shown in the first row of Table III are the same. It is also worth noting that the training times are sampled at the end of the algorithm iterations, thus the measured times are the same when the target C_{avg} is reached at the end of the same iteration.

Given enough training time, all algorithms, including the primal ones, estimate sets of Lagrange multipliers $\boldsymbol{\alpha}$ that allow good pushed–GMM models to be generated.

The DCDM algorithm is the fastest to reach convergence, followed by the BMRM algorithm. Pegasos is much slower but its models give slightly better results, probably due to the fact that its algorithm for the evaluation of the (approximate) Lagrange multipliers does not impose constraints to the $\boldsymbol{\alpha}$ values.

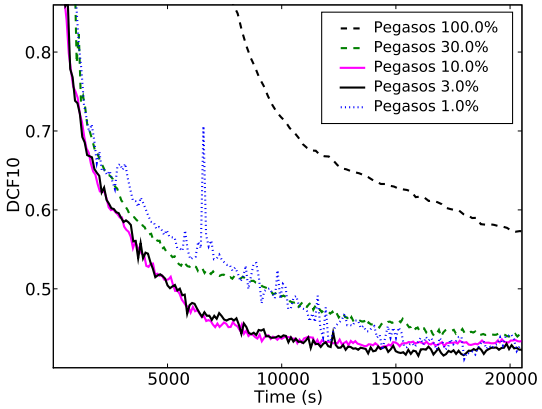


Fig. 3: SRE-10 DCF of Pegasos models as a function of their training time for different bunch sizes (% of the training dataset)

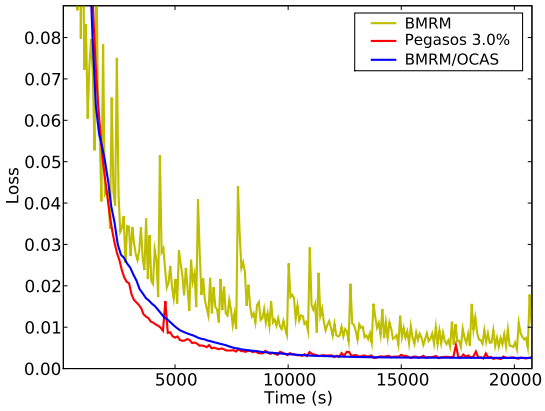


Fig. 4: SRE10 loss function of Pegasos, BMRM, and OCAS models as a function of their training time

B. Speaker recognition task

In this section we compare the performance of the speaker models trained according to the pair-wise discriminative approach presented in Section IV. It is worth recalling that training is feasible using two solvers only: BMRM and Pegasos.

The training set for this task consists of 21663 segments from 1384 female speakers taken from NIST 2004, 2005, and 2006 Speaker Recognition Evaluations, Switchboard II Phase 2 and 3 and Switchboard Cellular Parts 1 and 2 [41].

The i–vectors estimated from 60–dimensional 2048–mixture GMMs have 400–dimensions. The UBM and the i–vector extractor are trained on NIST 2004, 2005, and 2006 SRE corpora, and on Switchboard

TABLE IV: SPEAKER RECOGNITION TASK: SRE-10 FEMALE TEL-TEL TEST PERFORMANCE

Algorithm	Model	EER	DCF08	DCF10
BMRM/OCAS	asymptotic	2.54%	0.120	0.416
Pegasos	asymptotic	2.54%	0.119	0.420
BMRM/OCAS	$\approx 7000s$	2.32%	0.110	0.401
Pegasos	$\approx 7000s$	2.62%	0.121	0.434
BMRM/OCAS	best	2.19%	0.106	0.369

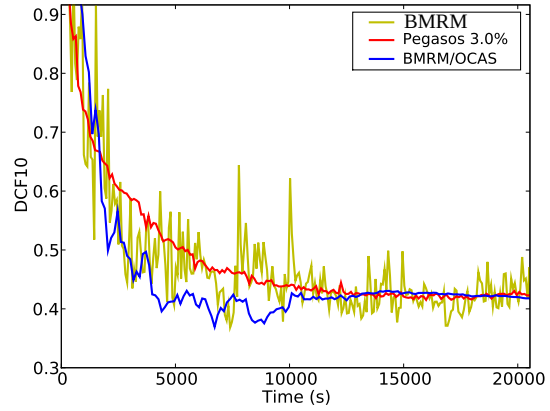
and Fisher data. More details about the i–vector extraction procedure can be found in [42]. System performance is presented in terms of Equal Error Rate and minimum Detection Cost Functions defined by NIST for the 2008 (DCF08) and for the more challenging 2010 (DCF10) evaluations [36]. The two functions differ for the relative cost attributed to False Alarms with respect to Miss Classification errors (much higher for DCF10).

Fig. 3 compares the DCF10 performance of Pegasos for different bunch size as a function of the training time. The slowest converging approach is standard Gradient Descent (Pegasos 100%), where all patterns are considered at each iteration before updating the hyperplane. Stochastic Gradient Descent applied to bunch of patterns - shown as a fraction of the total dataset in Fig. 3 - allows faster convergence, but the bunch size cannot be reduced to small fractions of the dataset (less than 3% in these experiments) without loosing the benefits of the efficient training strategy introduced in [20], [21].

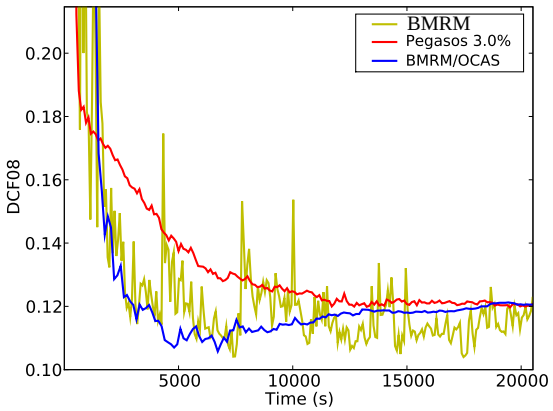
In the following, the BMRM models will be compared with the ones produced by using a bunch size of 3% in Pegasos.

Fig. 4 shows the SVM loss function value with respect to training time for the BMRM and Pegasos approaches. BMRM requires more time to reach convergence compared to Pegasos, moreover, its loss function is unstable even after several iterations. We did not take care of this behavior in the phonetic system because DCDM was sensibly faster. This is not the case for the speaker recognition system as can be seen in Fig. 5 (a) where the BMRM DCF10, plotted as a function of the time spent to train its models, shows large fluctuations. This behavior is not surprising because at each iteration BMRM finds a cutting plane which approximates the objective function at the current solution. However the objective function does not necessarily decrease at each iteration. These fluctuations, of course, reduce toward convergence, and the algorithm reaches the asymptotic performance of Pegasos.

A solution to this problem has been proposed in the Optimized Cutting Plane Algorithm (OCAS) approach [43], [3], which improves the BMRM by trying to simultaneously optimize the original and the approximated objective function and to select cutting planes that have higher chance to actively contribute



(a)



(b)

Fig. 5: DCF10 (a) and DCF08 (b) as a function of the training time for models produced by BMRM, Pegasos, and OCAS

to the approximation of the objective function around its optimum. OCAS choice of the cutting planes allows reducing the number of iterations needed for convergence, at the expense of an higher execution time per iteration. In particular, OCAS complexity is $O(n \log n)$, however it can be considered linear, because usually $\log n \ll d$ (this holds also in our experiments), thus the global complexity of the algorithm is dominated by the factor $O(nd)$ needed for the computation of dot–products and gradients.

Using BMRM and the OCAS approach for cutting planes selection allows obtaining a loss function, shown in Fig. 4, with a fast and smooth convergence similar to the one obtained by Pegasos.

As far as the DCF is concerned, the asymptotic performance of BMRM/OCAS and Pegasos is similar, as reported in the first two rows of Table IV. However, the comparison of the DCF10 and DCF08 in Fig. 5 (a)

TABLE V: COMPARISON OF THE PROPERTIES OF THE SELECTED SVM ALGORITHMS

Algorithm	Complexity	Parallel	Provides α's	PLDA SVM
SVM$^{\text{Light}}$	$O(n^2 d)$	Yes	Yes	No
SVM$^{\text{Perf}}$	$O(\frac{nd}{\epsilon})$	Yes	No	Yes
DCDM	$O(nd \log \frac{1}{\epsilon})$	No	Yes	No
Pegasos	$\tilde{O}(\frac{d}{\delta\epsilon})$*	No	Yes	Yes
BMRM/OCAS	$O(\frac{nd}{\epsilon})$	Yes	Yes	Yes

*Reaches ϵ accuracy with probability $1 - \delta$

and (b), respectively, shows that Pegasos reaches slowly its asymptotic performance confirming the results obtained in the language recognition experiments. BMRM has large fluctuations, thus finding an early stopping criterion is difficult. However, BMRM has also the potential to give much better generalization results on the evaluation data if its models are trained with a reduced number of iterations. BMRM/OCAS allows to fit the best performance of BMRM using models obtained by stopping the iterations when the loss function decrease is less than 3%. This condition, which is usually achieved after a few tens of iterations, avoids over-fitting that manifests itself in the asymptotic convergence region. The stopping criterion value was estimated using the NIST-SRE 2008 evaluation data as held–out development set. In these experiments the stopping criterion has been reached in 50 iterations corresponding to approximately 7000 seconds of training time. Table IV compares the results of the models trained by BMRM/OCAS and by Pegasos in this same amount of time, showing that the performance of Pegasos is worse compared not only to BMRM/OCAS, but also to its asymptotic results. Last row of the table shows, for reference, the results of the best model.

VIII. CONCLUSIONS

An analysis of five large–scale SVM training algorithms has been presented. With the exception of the pushed-GMM approach, which requires a relatively expensive likelihood scoring, an accurate analysis was possible for the other test experiments because SVM scoring is extremely fast, which makes viable testing thousands of different models.

A comparison of the properties of the SVM algorithms that we analyzed is given in Table V. The algorithms scale linearly with the training set size, and reach approximately the same classification performance given enough training time, but they have different characteristics with respect to their speed of convergence and scalability. Moreover, the selection of the best algorithm for a task depends also on the two paradigms we have presented: the standard one that builds a model for each class, and

the pair-wise discriminative approach of Section IV, which estimates a single discriminative model.

A. Classical SVM

SVM$^{\text{Perf}}$ is fast to reach convergence, behind DCDM only, and can be parallelized, but it has shown two drawbacks for our applications: classes cannot be easily balanced and there are no means to estimate Lagrange multipliers for the pushed-GMM approach.

Pegasos is not attractive because it is slower than the other implementations even though it can be easily extended to different loss functions.

BMRM is faster than Pegasos and even faster with a multi-threaded implementation.

DCDM is the preferred solution for training these classifiers: it is the fastest to converge on a single processor and it allows a multi–threaded implementation, where a set of threads can be devoted to training in parallel the models of different classes, even though the core utilization is not optimal. However, it cannot exploit distributed architectures and is much less flexible than Pegasos and BMRM.

B. Pair-wise SVM

As remarked in Section VI-B only primal solvers, such as BMRM/OCAS and Pegasos, are good candidates for pair-wise SVM training. Using these algorithms, the training complexity is reduced from $O(n^2d^2)$ to $O(n^2d)$, where n^2 is the number of the training patterns.

BMRM/OCAS is the preferred choice for fast training of a pair-wise discriminative model because Pegasos reaches its asymptotic performance slowly, thus stopping its iterations before convergence does not produce models as good as the BMRM/OCAS models obtained after a comparable time.

ACKNOWLEDGMENTS

We would like to thank the organizers and the participants to the BOSARIS Workshop, in particular Ondřej Glembek, Pavel Matějka and Oldřich Plchot from Brno University of Technology, for providing us with the i-vectors used in the speaker recognition experiments, Niko Brümmer and Lukáš Burget for introducing us to the pair-wise discriminative approach, and Fabio Castaldo, Daniele Colibro and Claudio Vair from Loquendo, for giving us several tools, advices and data for the language recognition task.

REFERENCES

[1] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, 1992.

[2] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large–scale linear SVM," in *Proceedings of ICML 2008*, pp. 408–415, 2008.

[3] C. H. Teo, A. Smola, S. V. Vishwanathan, and Q. V. Le, "Bundle methods for regularized risk minimization," *J. Mach. Learn. Res.*, vol. 11, pp. 311–365, March 2010.

[4] W. M. Campbell, "A covariance kernel for SVM language recognition," in *Proceedings of ICASSP 2008*, pp. 4141–4144, 2008.

[5] C. J. C. Burges, "A tutorial on Support Vector Machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998.

[6] M. A. Zissman, "Comparison of four approaches to automatic language identification of telephone speech," *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 1, pp. 31–44, 1996.

[7] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian Mixture Models," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 31–44, 2000.

[8] J.-L. Gauvain, A. Messaoudi, and H. Schwenk, "Language recognition using phone lattices," in *Proceedings of Interspeech 2004*, pp. 25–28, 2004.

[9] J.-L. Gauvain, A. Messaoudi, and H. Schwenk, "Language recognition with word lattices and Support Vector Machines," in *Proceedings of ICASSP 2007*, pp. 15–20, 2007.

[10] P. A. Torres-Carrasquillo, D. A. Reynolds, E. S. M. A. Kohler, R. J. Greene, and J. J. R. Deller, "Approaches to language identification using Gaussian Mixture Models and shifted delta cepstral features," in *Proceedings of ICSLP 2002*, pp. 89–92, 2002.

[11] L. Burget, P. Matějka, and J. Cernocky, "Discriminative training techniques for acoustic language identification," in *Proceedings of ICASSP 2006*, pp. 209–212, 2006.

[12] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-Carrasquillo, "Support Vector Machines for speaker and language recognition," *Computer Speech and Language*, vol. 20, no. 1-3, pp. 210–229, 2006.

[13] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Acoustic language identification using fast discriminative training," in *Proceedings of Interspeech 2007*, pp. 346–349, 2007.

[14] W. Campbell, J. Campbell, T. Gleason, D. Reynolds, and W. Shen, "Speaker verification using Support Vector Machines and high–level features," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 7, pp. 2085–2094, 2007.

[15] D. Albesano, R. Gemello, and F. Mana, "Hybrid HMM-NN modeling of stationary-transitional units for continuous speech recognition," in *Proceeding of Neural Information Processing 2007*, pp. 1112–1115, 1997.

[16] W. M. Campbell, D. E. Sturim, D. A. Reynolds, and A. Solomonoff, "Svm based speaker verification using a GMM supervector kernel and NAP variability compensation," in *Proceedings of ICASSP 2006*, pp. 97–100, 2006.

[17] K. Berkling, J.-F. Bonastre, and J. P. Campbell, "Introduction to the special section on speaker and language recognition," *IEEE Transaction on Audio Speech and Language Processing*, vol. 15, no. 7, 2007.

[18] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel, "Joint Factor Analysis versus eigenchannels in speaker recognition," *IEEE Transaction on Audio Speech and Language Processing*, vol. 15, no. 4, pp. 1435–1447, 2007.

[19] L. Burget *et al.*, "Robust speaker recognition over varying channels," in *Johns Hopkins University CLSP Summer Workshop Report*, 2009. Available at http://www.clsp.jhu.edu/workshops/ws08/documents/jhu report main.pdf.

[20] S. Cumani, N. Brümmer, L. Burget, and P. Laface, "Fast discriminative speaker verification in the i–vector space," in *Proceedings of ICASSP 2011*, 2011.

[21] L. Burget, O. Plchot, S. Cumani, O. G. P. Matějka, and N. Brümmer, "Discriminatively trained Probabilistic Linear Discriminant Analysis for speaker verification," in *Proceedings of ICASSP 2011*, 2011.

[22] N. Dehak, R. Dehak, P. Kenny, N. Brümmer, and P. Ouellet, "Support Vector Machines versus fast scoring in the low-dimensional total variability space for speaker verification," in *Proceedings of Interspeech 2009*, pp. 1559–1562, 2009.

[23] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front–end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. PP Issue:99, pp. 1–1, 2010.

[24] N. Dehak, R. Dehak, P. Kenny, N. Brümmer, and P. Ouellet, "Probabilistic Linear Discriminant Analysis for inferences about identity," in *Proceedings of 11th International Conference on Computer Vision*, pp. 1–8, 2007.

[25] 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.

[26] T. Joachims, "Making large–scale Support Vector Machine learning practical," in *Advances in Kernel Methods – Support Vector Learning*, pp. 169–184, MIT–Press, 1999.

[27] S. Cumani, F. Castaldo, P. Laface, D. Colibro, and C. Vair, "Comparison of large–scale SVM training algorithms for language recognition," in *Proceedings of Odyssey 2010, The Speaker and Language Recognition Workshop*, pp. 222–229, 2010.

[28] T. Joachims, "A support vector method for multivariate performance measures," in *Proceedings of ICML 2005*, pp. 377–384, 2005.

[29] T. Joachims, "Training linear SVMs in linear time," in *Proceedings of KDD 2006*, pp. 217–226, 2006.

[30] T. Joachims and C.-N. J. Yu, "Sparse kernel SVMs via cutting–plane training," *Machine Learning*, vol. 76, no. 2–3, pp. 179–193, 2009.

[31] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Primal estimated sub–gradient solver for SVM," in *Proceedings of ICML 2007*, pp. 807–814, 2007.

[32] C. H. Teo, A. Smola, S. V. Vishwanathan, and Q. V. Le, "A scalable modular convex solver for regularized risk minimization," in *Proceedings of KDD 2007*, pp. 727–736, 2007.

[33] Available at http://numpy.scipy.org.

[34] Available at http://abel.ee.ucla.edu/cvxopt.

[35] Available at http://www.openmp.org.

[36] Available at http://www.itl.nist.gov/iad/mig/tests/sre/2010/NIST_SRE10 _evalplan.r6.pdf.

[37] Available at http://www.itl.nist.gov/iad/mig/tests/lre/2009/LRE09_Eval Plan_v6.pdf.

[38] F. Castaldo, D. Colibro, S. Cuman, E. Dalmasso, P. Laface, and C. Vair, "Loquendo-politecnico di Torino system for the 2009 NIST Language Recognition Evaluation," in *Proceedings of ICASSP 2010*, pp. 5002–5005, 2010.

[39] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Compensation of nuisance factors for speaker and language recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 7, pp. 1969–1978, 2007.

[40] P. Matějka, L. Burget, O. Glembek, P. Schwarz, V. Hubeika, M. Fapso, T. Mikolov, O. Plchot, and J. Cernocky, "But language recognition system for NIST 2007 evaluations," in *Proceedings of Interspeech 2007*, pp. 739–742, 2007.

[41] Available at http://www.ldc.upenn.edu/Catalog/.

[42] P. Matějka, O. Glembek, F. Castaldo, O. Plchot, P. Kenny, L. Burget, and J. Cernocky, "Full-covariance ubm and heavy-tailed plda in i–vector speaker verification," in *Proceedings of ICASSP 2011*, pp. 4828–4831, 2011.

[43] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for Support Vector Machines," in *Proceedings of ICML 2008*, pp. 320–327, 2008.

Pietro Laface received the M.S. degree in Electronic Engineering from the Politecnico di Torino, Torino, Italy, in 1973.

Since 1988 it has been full Professor of Computer Science at the Dipartimento di Automatica e Informatica of Politecnico di Torino, where he leads the speech technology research group. He has published over 120 papers in the area of pattern recognition, artificial intelligence, and spoken language processing. His current research interests include all aspects of automatic speech recognition and its applications, in particular speaker and spoken language recognition.

Sandro Cumani received the M.S. degree in Computer Engineering from the Politecnico di Torino, Torino, Italy, in 2008, and is currently completing his Ph.D. degree in Computer and System Engineering of Politecnico di Torino. His current research interests include machine learning, speech processing and biometrics, in particular speaker and language recognition.