Comparison of Large–scale SVM Training Algorithms for Language Recognition

Comparison of Large–scale SVM Training Algorithms for Language Recognition

Sandro Cumani [*], *Fabio Castaldo* [o], *Pietro Laface* [*], *Daniele Colibro* [o], *Claudio Vair* [o]

[*] Politecnico di Torino, Italy
{Sandro.Cumani, Pietro.Laface}@polito.it
[o] Loquendo, Italy
{Fabio.Castaldo,Daniele.Colibro,Claudio Vair}@loquendo.com

Abstract

This paper compares the performance of large scale Support Vector Machine training algorithms tested on a language recognition task.
We analyze the behavior of five SVM approaches for training phonetic and acoustic models, and we compare their performance in terms of number of iterations to reach convergence, training time and scalability towards large databases. 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 on the database.

1. Introduction

Fast algorithms for training Support Vector Machines (SVM) often rely on the so called kernel trick, 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 nonlinear classification in very high dimension spaces to be performed. 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 are performed on large dimension feature vectors and large databases, the allocation of the kernel matrix in main memory becomes unfeasible even for large memory modern computers. We are thus interested in SVM approaches scaling linearly with the number of training patterns. In this paper we compare large scale Support Vector Machine training algorithms using as testbed a language recognition task. In particular, we train both phonetic and acoustic SVM models, comparing the performance of several algorithms in terms of training time, number of iterations required to reach convergence, and scalability towards large databases.

The paper is organized as follows: Section 2 gives a short overview of the SVM classifier. Section 3 illustrates four different approaches that have been proposed to solve large scale SVM problems. Section 4 summarizes the acoustic and phonetic features and models that have been used, and highlights that we have to deal with high dimension patterns. The details of the implementation of some SVM algorithms and the experimental results are presented and commented in Section 5, and conclusions are drawn in Section 6.

2. Support Vector Machines

A Support Vector Machine [1] is a two–class classifier which looks for 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

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

where vector $\mathbf{w}$ is a vector perpendicular to the hyperplane and the squared $L2$–norm of $\mathbf{w}$ is the regularization term. The second term, where $\mathbf{x}_i \in \mathbb{R}^d$ denotes a (d–dimensional) training pattern with associated label $y_i \in \{-1, +1\}$, is the loss function, which is an approximation of the generalization error of the classifier estimated on the training set itself weighted by parameter C.
The standard loss function for the SVM problem, which gives the maximum (soft–)margin classifier, is the so–called L1–loss function

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

though other choices are possible, such as the L2–loss function (squared soft margin SVM)

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

The standard *primal* SVM formulation is then

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

While it is possible to solve the SVM optimization problem in its primal form, many approaches prefer to solve the dual problem

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) = \tfrac{1}{2}\alpha^T H \alpha - \mathbf{e}^T \alpha \\ \text{subject to} \quad & 0 \le \alpha_i \le C \qquad \forall i \end{aligned}$$

where α are the Lagrange multipliers, $\mathbf{e}$ is a vector of ones, and H is the matrix of dot–products $H_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ for L1–loss. For L2–loss $H_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j + D_{ij}$ and D is the diagonal matrix $D_{ii} = \frac{1}{2C}$.
The dual formulation exposes the direct dependency of the optimal hyperplane on the dot–products between pairs of training patterns, which in turns allows easily extending SVM to non–linear classification. This is done by performing non–linear classification in the feature space X by means of linear classification in a higher dimensionality space F. The so–called

kernel trick allows mapping $\Phi : X \rightarrow F$ to be performed implicitly by simply substituting the dot–product matrix H by the kernel matrix K defined as $K_{ij} = \Phi(x_i)\Phi(x_j) = K(x_i, x_j)$ given that the resulting matrix is positive semi–definite.

3. Large–scale Support Vector Machines

The ever–increasing size of the training databases 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. Since in speaker and language recognition a large number of high dimensional patterns have to be learned, we are then interested in SVM training approaches whose time complexity and memory consumption scale at most linearly with the number of training patterns. Many algorithms have been proposed to handle SVM optimization for large–scale problems. Most of these algorithms are efficient only for *linear* kernel SVMs, but these are actually the ones we need in language recognition, as will be illustrated in Sections 4.1 and 4.2. In this Section we present five training algorithms focusing on their complexity and possibility to be executed on a distributed environment. In the following, n will denote the number of training patterns, d the dimensionality of such patterns, C the regularization parameter of the SVM and ε the optimization accuracy.

3.1. SVM$^{\text{Light}}$

SVM$^{\text{Light}}$ [2] is one of the first proposed "fast" linear–space SVM solvers. It decomposes the SVM problem into a set of subproblems and works with only a small subset of such subproblems at a time. Memory occupation for this algorithm scales linearly with the number of training patterns and of support vectors. Since SVM$^{\text{Light}}$ solves the dual problem, it provides a way to easily include kernels and it provides the Lagrange multipliers needed in language recognition by the pushed–GMMs approach that will be described in Section 4.2.1. The main limitation of this algorithm comes from its time complexity, which has been empirically shown to be in the order of $O(n^2 d)$. When memory occupation is not a constraint, a fast implementation of SVM$^{\text{Light}}$ can be obtained by caching all the kernel evaluations. However, the resulting kernel matrix has a size, and thus a computational cost, which grow quadratically with the training set size.

3.2. SVM$^{\text{Perf}}$

SVM$^{\text{Perf}}$ [3], [4], [5] is one of the most popular linear–time SVM solvers. Though the package provides different algorithms for solving the SVM problem, the main innovation is its Cutting–Plane Space–Pursuit (CPSP) approach [5]. Cutting–Plane algorithms are based on a different formulation of the problem [4]

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

where $\Delta(y, \hat{y})$ is the zero–one loss function.
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 algorithms by iteratively building, along with the solution, a set of *basis vectors* $\mathbf{b}_i$ whose span is approximatively the sub–space where the optimal solution lies. The solution, thus is given by

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

Basis vectors are similar to support vectors, the difference being that they are not taken among the training set patterns.
The idea of using basis vectors is actually motivated by non–linear SVMs, since the use of kernels usually induces a quadratic behavior in those algorithms which are linear for linear kernels. Since the basis vectors are associated with the Cutting Plane constraints, which are a constant number with respect to the training set, the number of basis vectors is actually independent from the training set size. This algorithm can be easily modified to be executed in a distributed environment.

3.3. Pegasos

Standard gradient descent (GD) 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 (L1–loss) objective function

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t (\mathbf{w}_t + C \sum_{i=1}^{n} \nabla \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i))$$

where η_t is the learning parameter, whose value is crucial for fast convergence of the algorithm. Since the L1–loss function is not completely differentiable but still convex we actually compute a *subgradient* of the loss function

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

Stochastic Gradient Descent (SGD) approximates the gradient computation step by evaluating the gradient of the objective function on a single sample (or on a small subset of samples)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t (\mathbf{w}_t + C \nabla l(\mathbf{w}, \mathbf{x}_{i_t}, y_{i_t}))$$

where i_t is chosen randomly for each iteration.

An interesting application of stochastic subgradient descent to L1–loss SVM has been proposed in [6]. The Pegasos algorithm combines stochastic gradient descent with a projection step ensuring that we get closer to the optimal solution. A set of training patterns $\mathcal{A}_t$ is chosen at each iteration, and the subgradient of the objective function is evaluated on this subset as

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

and the hyperplane is then updated by

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

where η_t is the learning rate at iteration t. The next step consists in projecting the solution onto a ball of radius $\sqrt{C}$, since it is possible to show that the optimal solution lies inside this ball. This is done by simply scaling $\mathbf{w}_{t+\frac{1}{2}}$ according to

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

$$\mathbf{w}_{t+1} = s_t \mathbf{w}_{t+\frac{1}{2}}$$

This projection step, combined with a fast–decaying learning rate, allows to bound the average number of iterations required to achieve ε optimization accuracy in $O\left(\frac{1}{\varepsilon}\right)$.
Since Pegasos solves the primal formulation of the SVM problem it does not produce the Lagrange multipliers, which are necessary in the pushed–GMMs approach. In [6] 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 α's are obtained as

$$\alpha_{t+1} = \left[\alpha_t - \eta_t \left(\alpha_t + \frac{C}{\|\mathcal{A}_t\|}\chi_t^i\right) s_t\right]$$

where

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

Since Pegasos is based on stochastic gradient descent it cannot take advantage of a distributed environment.

3.4. Dual Coordinate Descent

In [7] the authors have proposed a coordinate descent approach applied directly to the dual problem, referred to in the following as DCDM. Coordinate descent splits the multivariate problem into a sequence of univariate optimizations which are iteratively solved until convergence to the optimal multivariate solution is attained. Dual coordinate descent applies this idea to the dual formulation of the SVM problem. Assuming that a sub–optimal solution α is known, the i–th component of the optimal solution given all the other coordinates can be evaluated by solving

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

where $\mathbf{e}_i$ is the i–th versor of the solution space and U is an upper bound depending on the loss function (in particular, $U = C$ for L1–loss and $U = +\infty$ for L2–loss). This function is quadratic in h and its Taylor expansion is given by

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

for a given constant K. This function has a minimum in $h = 0$ if and only if

$$\nabla_i^P f(\alpha) = 0$$

where $\nabla_i^P f(\alpha)$ is the projected gradient

$$\nabla_i^P f(\alpha) = \begin{cases} \nabla_i f(\alpha) & \text{if } 0 < \alpha_i < U \\ \min(0, \nabla_i f(\alpha)) & \text{if } \alpha_i = 0 \\ \max(0, \nabla_i f(\alpha)) & \text{if } \alpha_i = U \end{cases}$$

The projection of the gradient is required to ensure that boundary constraints are met. If the projected gradient is not 0, however, the unconstrained optimal solution is given by

$$h = -\frac{\nabla_i f(\alpha)}{H_{ii}}$$

which leads to the update rule

$$\alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\nabla_i f(\alpha)}{H_{ii}}, 0\right), U\right)$$

The computation of the gradient requires the evaluation of

$$\nabla_i f(\alpha) = \sum_{i=1}^{n} H_{ij}\alpha_j - 1$$

which is computationally expensive, but simplifies for linear SVMs as

$$\nabla_i f(\alpha) = y_i \mathbf{w}^T \mathbf{x}_i - 1 + D_{ii}\alpha_i$$

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

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

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–GMMs approach.
The complexity of the algorithm is $O\left(nd \log\left(\frac{1}{\varepsilon}\right)\right)$. DCDM is very fast, but like Pegasos, cannot take advantage of a distributed environment.

3.5. Bundle Methods

Bundle methods approximate a convex function by means of tangential hyperplanes (sub–gradients) and solving the simpler optimization problem on the approximated function. The approach is similar to what is done in SVM$^{\text{Perf}}$ [4], where a small and incremental subset of constraints is built until the solution approximates the optimal solution up to a given error. However, Bundle Methods for Regularized Risk Minimization (BMRM) as presented in [8] 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 (which are computed incrementally starting from $\mathbf{w}_0 = \mathbf{0}$)

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

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

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

This formulation can been shown to be equivalent to the dual problem

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

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

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

It is interesting to observe that even though the algorithm does not provide the Lagrange multipliers for the dual SVM problem, it is possible to evaluate an approximation of the α values of the SVM allowing to reconstruct the hyperplane provided by the BMRM algorithm and satisfying the SVM constraints.
Both L1 and L2–loss functions can be rewritten to make explicit their dependency on the dot product between $\mathbf{w}$ and a given pattern $\mathbf{x}_i$ as

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

Using this formulation, the gradient of l can be expressed as

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

If we define the array $\widetilde{a}_t = [\widetilde{l}_1 \widetilde{l}_2 \dots \widetilde{l}_n]^T$ where $\widetilde{l}_i = \nabla_{\mathbf{w}^T\mathbf{x}} \widetilde{l}(\mathbf{w}^T\mathbf{x}_\mathbf{i}, y_i)$ we can express a_t as

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

where $\mathbf{X}$ is the complete set of training patterns represented as a matrix. Matrix A can then be evaluated as $A = \mathbf{X}\widetilde{A}$ with $\widetilde{A} = [\widetilde{a}_1 \widetilde{a}_2 \dots \widetilde{a}_t]$. It is easy to see that the resulting expression for $\mathbf{w} = -CA\beta$ can now be rewritten as

$$\mathbf{w} = -C\mathbf{X}\widetilde{A}\beta$$

Setting $\alpha = -CY^{-1}\widetilde{A}\beta$, where Y is the diagonal matrix of target labels, allows expressing the hyperplane as a linear combination of the training patterns $\mathbf{w} = \mathbf{X}Y\alpha = \sum_i y_i \mathbf{x}_i \alpha_i$.

It can be shown [8] that the 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 sub–problems 1 can be neglected. In this case, the global complexity of the algorithm is $O\left(\frac{nd}{\varepsilon}\right)$.
Since BMRM, as SVM$^{\text{Perf}}$, incrementally builds a working set of approximate solutions, its algorithm can be easily modified to be executed in a distributed environment.

In the following Sections we will show the results obtained by training SVM models for a language recognition task with these algorithms.

4. Language recognition models

Two main approaches are used in state of the art systems for Language Recognition, both can be referred to the works [9] and [10].
In the first approach the language models are estimated collecting the statistics of the occurrence of n–grams obtained by running one or more phonetic decoders that produce strings or lattices [11, 12] of phonetic tokens.
The second approach relies on Gaussian Mixture Models of acoustic features (GMMs) [13] discriminatively trained [14], or used in combination with SVM classifiers [15, 16, 17].
The next subsections give more details about the two approaches, and present the features and models that have been used for our language recognition task.

4.1. Phonetic models

Phonetic systems model sentences by bags of n–grams extracted by a phonetic decoder, either by looking for the best matching sequence of tokens or by exploiting a lattice of word/phoneme hypotheses. Each sentence is then represented by the set of probabilities of n–gram occurrences given the sentence stacked in a vector. Although good performance can been obtained by classifying the test utterances by means of log–likelihood scoring, state of the art results are obtained by means of linear SVM classifiers. Among the SVM kernels that have been proposed, the most used is the TFLLR kernel [18, 19, 17], which approximates a log–likelihood ratio computed from n–gram statistics. TFLLR is a linear kernel that can be obtained by a simple normalization of the n–gram probabilities

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

where the superscript i denotes the i–th component of $\mathbf{x}$.
Using the probabilities of the most frequent n–grams up to the third or even higher order, the dimension of the resulting vector can be of the order of a hundred thousand, with a high degree of sparsity. It is possible to improve phonetic language recognition by using the information provided by lattice decoding, rather than the information provided by the best decoded phonetic string for each utterance, however the resulting SVM patterns are much more dense.

In the experiments reported in Section 5 the phonetic system includes a single recognizer producing a phone lattice. The phone tokenizer is the Loquendo–ASR recognizer for Italian language [20]. It uses a hybrid ANN–HMM model where each phonetic unit is described by a single or 2–state left–to–right automaton with self–loops and uniform transitions. The phonetic ANN is a four–layer perceptron trained to estimate the posterior probability of each state (phone class) given a 7 frame context window of acoustic features including Mel coefficients and their first and second derivatives. The hidden layers consist of 315 and 300 sigmoidal units respectively, while the last layer has 733 units with a softmax activation function. The n–gram probabilities are estimated from the lattice [11] and then normalized using the TFLLR kernel. In our experiments, the 3–gram vectors have a dimension of 44135.

4.2. Acoustic models

The acoustic models are obtained by estimating Gaussian Mixture Models $g(\mathbf{x}) = \sum_{i=1}^{m} c_i \mathcal{N}(\mathbf{x}, \mu_i, \mathbf{\Sigma}_i)$ obtained from a common Universal Background Model using Maximum A Posteriori adaptation with a low relevance factor [10]. In our experiments, the UBM and the language GMMs consist of mixtures of 2048 Gaussians. The observation vectors consist of 56 parameters (7 Mel frequency cepstral coefficients and their 7–1–3–7 Shifted Delta (SDC) coefficients [13]. Nuisance compensation is performed in the feature domain by means of factor analysis [19]).
A supervector of the stacked Gaussian means $\mathbf{g} = \left[\mu_1^T \mu_2^T \dots \mu_m^T\right]^T$ can be used as a pattern representing a speaker utterance, and classification can be done by estimating SVM models [17] using the Kullback–Leibler kernel [21, 19]

$$K(\mathbf{g}_a, \mathbf{g}_b) = \sum_{i=1}^{m} \left(\sqrt{c_i}\mathbf{\Sigma}_i^{-\frac{1}{2}}\mu_{a\,i}\right)^T \left(\sqrt{c_i}\mathbf{\Sigma}_i^{-\frac{1}{2}}\mu_{b\,i}\right)$$

Again, appropriately normalized features can be used with a linear SVM.

4.2.1. Pushed–GMMs

Better performance can be obtained by an hybrid discriminative–generative approach as proposed in [17]. In particular, two GMMs are estimated for each class: one modeling the data of the target class $g^+(\mathbf{x})$ and the other modeling all the non target classes (anti–model) $g^-(\mathbf{x})$. The models g^+ and g^- are obtained using a linear combination of the supervectors where the combination weights are the Lagrange multipliers associated to the support vectors of the SVM

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

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

Table 1: Phonetic system: asymptotic values for DCF and EER for the 30, 10 and 3 sec conditions

ALGORITHM	DCF 30s	EER 30s	DCF 10s	EER 10s	DCF 3s	EER 3s
SVM$^{\text{Light}}$	0.0375	3.972%	0.0858	8.881%	0.2037	20.772%
SVM$^{\text{Perf}}$	0.0434	4.583%	0.0944	9.782%	0.2061	21.057%
Pegasos	0.0393	4.221%	0.0880	9.139%	0.2029	20.908%
BMRM L1	0.0375	3.941%	0.0860	8.981%	0.2032	20.842%
DCDM L1	0.0376	3.965%	0.0861	8.948%	0.2031	20.785%
BMRM L2	0.0362	3.893%	0.0853	8.843%	0.2037	20.771%
DCDM L2	0.0362	3.890%	0.0853	8.847%	0.2037	20.770%

Table 2: Phonetic system: time (seconds) required to reach SVM$^{\text{Light}}$ accuracy within a given error percentage

Algorithm	DCF 30s			DCF 10s			DCF 3s		
	10%	1%	0.1%	10%	1%	0.1%	10%	1%	0.1%
SVM$^{\text{Light}}$	9372	9372	9372	9372	9372	9372	9372	9372	9372
SVM$^{\text{Perf}}$	> 334	> 334	> 334	> 334	> 334	> 334	334	> 334	> 334
Pegasos	2759	> 14930	> 14930	1825	> 14930	> 14930	897	2058	7230
BMRM L1	1901	6971	12199	1470	5773	6971	527	1684	3377
DCDM L1	86	169	169	86	131	131	41	86	86
BMRM L2	2720	3878	3878	2220	3878	4822	667	2960	4114
DCDM L2	91	186	186	91	140	186	42	91	91

Scoring is performed by computing the likelihood ratio between model and anti–model.

Since the α's of the dual problem are necessary to apply this technique, we need to evaluate them even when the SVM optimization is solved in its primal formulation.

5. Experiments

In this section we compare the above described algorithms by training both phonetic and acoustic models for the NIST Language Recognition Evaluation (LRE) 2009 [22]. The evaluation task is the detection of the language spoken in a set of speech segments. The segments are subdivided in categories of 30s, 10s and 3s nominal speech length. In the closed–set core test condition 23 languages must be recognized, including accented languages such as American English and Indian English.
System performance is reported in terms of Equal Error Rate and minimum Detection Cost Function (DCF) for each language defined by NIST [22].
It is worth noting that in these experiments we use just a single acoustic system based on pushed–GMMs, and a single tokenizer for the phonetic models. The models were trained with a set of 17521 patterns. Both the phonetic and acoustic system scores are normalized by means of a Gaussian back–end trained over a separate development set [23].

5.1. SVM training algorithm implementations

SVM training was done using tailored or new implementations of the algorithms presented in Section 3. The implementations were done using a Python/C framework. All computational intensive parts were written in C in order to minimize the impact of the Python interpreter, with the exception of the matrix–by–matrix multiplications required by the kernel matrix for SVM$^{\text{Light}}$, which were performed using machine–optimized numpy/BLAS libraries [24].

Our baseline training algorithm is SVM$^{\text{Light}}$ ([2]). To speed up the training process, the available software was modified to use a pre–computed kernel matrix stored in main memory.
DCDM has been implemented as presented in [7] and shrinking has been exploited to further speed up the execution time. The shrinking technique [2] tries to reduce the QP problem size by ignoring a subset of the bounded variables, thus greatly reducing the number of dot products that have to be computed, with no impact on the detection performance.
Due to the overhead that can be introduced by the C/Python infrastructure, training is performed in "batch" mode. A fraction of the database is loaded into main memory and then gradient computations are performed over the loaded data, thus the memory occupation of our algorithm is essentially determined by the size of these blocks (however the algorithm itself does not need to actually store more than a single pattern at once).
The same approach was followed for BMRM: training is performed in batch mode, while the QP problem is solved by means of the `cvxopt` [25] Python solver. Since the BMRM algorithm solves the problem in its primal formulation, no shrinking technique was adopted, thus each iteration of the algorithm actually performs a full scan of the training database. The α's needed for pushed–GMMs approach have been evaluated as in Section 3.5.
Pegasos has been implemented with a slight variation with respect to the algorithm presented in Section 3.3. Training is performed in epochs, where each epoch corresponds to training patterns taken randomly from a subset of the database that is loaded in main memory. The size of the block determines the memory occupation of the algorithm and allows for a reduction of the number of accesses to secondary memory.
Finally, SVM$^{\text{Perf}}$ was modified to read training patterns only on demand rather than loading all the dataset in main memory. Moreover, the database format was modified to fit the other implementations.
It is worth noting that in all the methods that we have implemented from scratch we have taken care to train the language models in parallel, minimizing disk accesses.

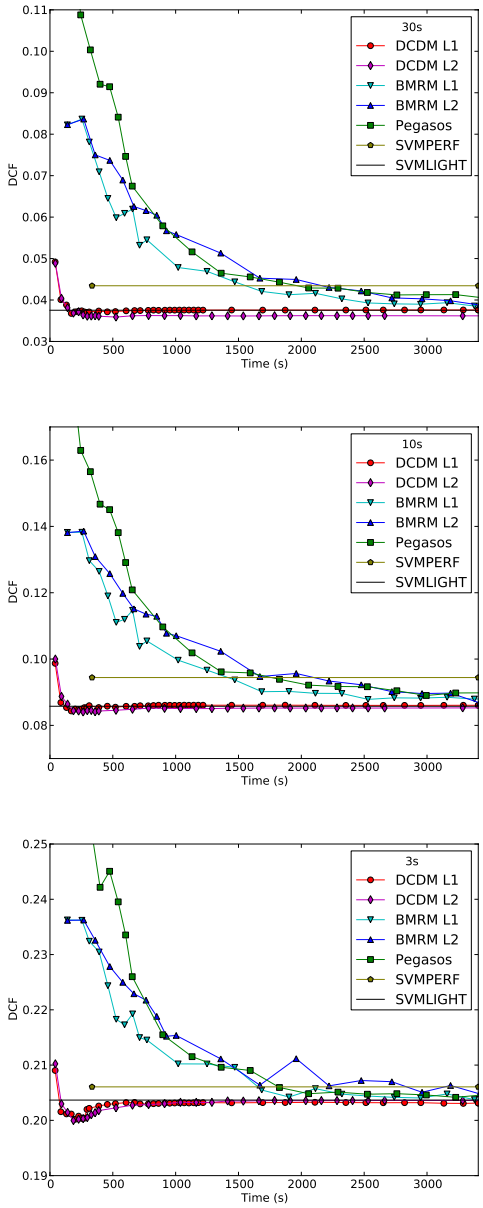

Figure 1: DCF performance of models as a function of their training time for the phonetic system (30, 10 and 3 sec conditions)

5.2. System performance comparisons

In this subsection we compare the behavior of the above mentioned techniques during the training of the phonetic system. To evaluate the quality of the trained models, we evaluated the models obtained after a given numbers of iterations. We used dense sampling for the first iterations and sparser sampling for the last ones. For SVM$^{\text{Light}}$ and SVM$^{\text{Perf}}$ we used just the final models because the former is our baseline system, and the latter is not attractive because it cannot be used for the pushed–GMMs approach, and moreover does not allow class balancing. All the algorithms, with the exception of SVM$^{\text{Perf}}$, have been

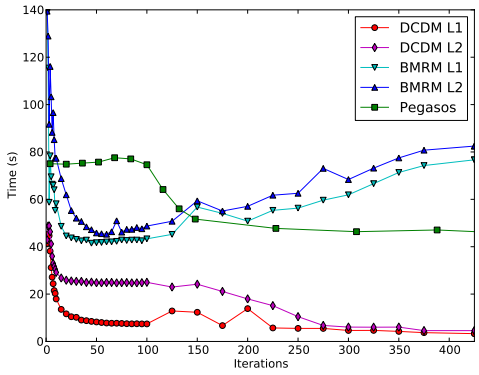

Figure 2: Time required for a single iteration over the database for the phonetic system

executed using a balance factor for the cost function which simulates the balance between the number of true and false samples. This is done by simply scaling the cost function for the true samples. SVM$^{\text{Perf}}$ does not provide a simple and direct way to perform class balancing, thus unbalanced models were created. Most of the performance loss of SVM$^{\text{Perf}}$ with respect to the other approaches is due to this problem, even though unbalanced models created with SVM$^{\text{Light}}$ still perform better than SVM$^{\text{Perf}}$ models.

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. Computations were performed using single-threaded processes. Table 1 shows the classification performance of the different phonetic models in the 30, 10, and 3 sec conditions of the NIST LRE09 tests. The SVM parameter C is estimated as $C = \left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i^T\mathbf{x}_i\right)^{-1}$. The asymptotic behavior of DCDM and BMRM converges to the DCF and EER obtained by SVM$^{\text{Light}}$. Pegasos behaves slightly worse, although the results it obtains before the Gaussian back–end are better of the ones achieved by the other techniques, possibly because it produces less calibrated scores. SVM$^{\text{Perf}}$ performance is behind the other approaches because of the lack of a class balance mechanism.

A second set of experiments has been performed to analyze the convergence properties of the different techniques. The results are shown in Table 2, which compares the time required to reach an accuracy within 10%, 1.0% and 0.1% with respect to our SVM$^{\text{Light}}$ baseline, and in Figure 1 that shows the DCF improvement as a function of the training time. Symbol $>$ denotes those conditions which did not meet the convergence condition at the end of training.

The convergence of the DCDM approach to its optimal value is much faster than the other two techniques. The obtained results are more than satisfactory after just five iterations (corresponding to 201 seconds of training time. These models are even better than the asymptotic ones possibly because they do not overfit the training data. It is worth noting that BMRM with L1–loss and SGD have a similar performance if we consider just the raw results, but BMRM is both faster and gives better performance after back–end normalization of its scores.

Figure 2 shows that the time required for a single iteration over the database (or over an equivalent number of patterns for the SGD) rapidly decreases from its initial value for all the techniques to a point where it remains almost constant. The reason

Table 3: Pushed–GMMs system: asymptotic values for DCF and EER for pushed–GMMs

ALGORITHM	DCF 30s	EER 30s	DCF 10s	EER 10s	DCF 3s	EER 3s
SVM^{Light}	0.0276	3.120%	0.0621	6.592%	0.1616	16.594%
Pegasos	0.0272	3.112%	0.0617	6.543%	0.1605	16.623%
BMRM L1	0.0276	3.153%	0.0626	6.587%	0.1618	16.579%
DCDM L1	0.0276	3.157%	0.0626	6.596%	0.1619	16.592%
BMRM L2	0.0272	3.090%	0.0609	6.429%	0.1589	16.359%
DCDM L2	0.0272	3.090%	0.0609	6.431%	0.1589	16.350%

is that less computation is actually required because most of the gradients become zero approaching the optimal solution.
As far as the BMRM technique is concerned, although the algorithm requires the solution of a QP problem at each iteration, it does not show a quadratic trend due to the small number of iterations that are performed compared to the time spent for processing the database samples.
The time per iteration required by DCDM is much lower than the BMRM one. This was expected because the former approach uses the shrinking technique.

5.3. Pushed–GMMs

Due to the time complexity of the tests on acoustic models, fewer models were trained and tested in the pushed–GMMs approach with respect to the phonetic system. Again, we evaluated the models obtained after a given number of iterations excluding the baseline models produced by SVM^{Light}.
It is worth noting that SVM^{Perf} cannot be used for the pushed–GMMs approach because it produces basis vectors whose associated weights cannot be mapped to support vectors, neither can be used to evaluate the required models and anti–models.

To speed–up the experimentation a faster but less precise algorithm for the selection of the best N–best Gaussians per frame has been used with respect to the one used for NIST LRE 2009. The results are presented in Figure 3 and Table 3. Looking at the asymptotic behavior of the training algorithms, it can be noticed that all of them — even BMRM and SGD, which do not directly solve the dual SVM problem — estimate sets of α values that allow good pushing–GMMs to be generated.

The use of L2–loss function improves performance, possibly because the Lagrange multipliers do not saturate to the C value as it often happens optimizing for the L1–loss function.
Surprisingly, good or even better results are obtained, for the 3s and 10s conditions, even by models whose pushing weights are far from the SVM optimum. Even more surprising are the results given by the first iteration of the BMRM, which actually corresponds to evaluating model and anti–model as the arithmetic mean of the true and false GMMs respectively.

6. Conclusions

A comparison of four SVM training algorithms for large scale features problems has been presented. All these algorithms scale linearly with the training set size.
We have shown that the accuracy of these algorithms is asymptotically equivalent, in that they all reach approximately the same classification performance given enough training time, but that they have different convergence and scalability properties.

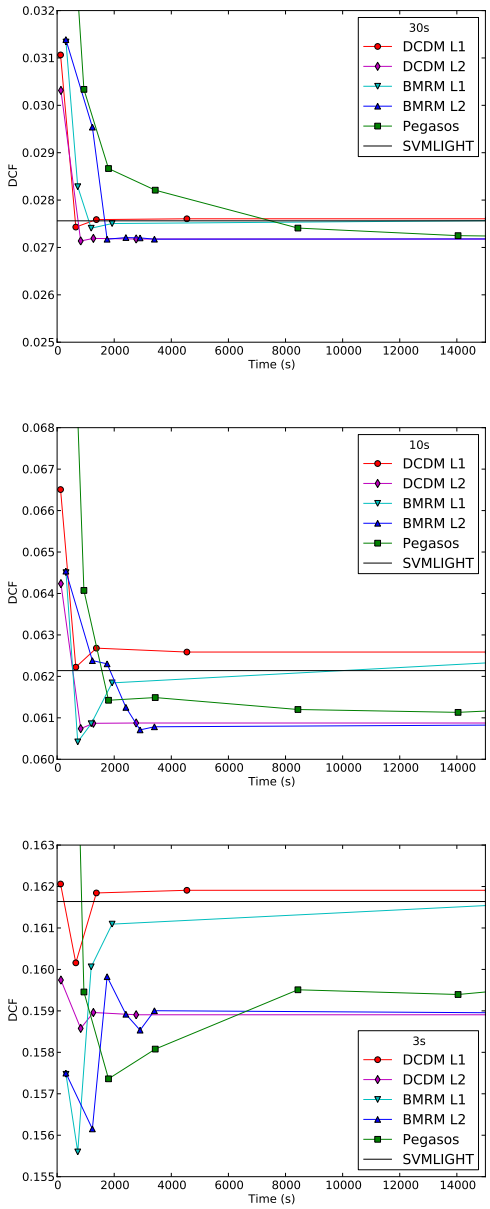

Figure 3: DCF performance of models as a function of their training time for the pushed–GMMs system (30, 10 and 3 sec conditions)

Table 4: Comparison of the properties of the SVM algorithms

ALGORITHM	Formulation	Complexity	Parallelizable	Provides α's
SVM$^{\text{Light}}$	Dual	$O(n^2 d)$	Yes	Yes
SVM$^{\text{Perf}}$	Dual*	$O(\frac{nd}{\epsilon})$	Yes	No
Pegasos	Primal	$\tilde{O}(\frac{nd}{\delta\epsilon})$**	No	Yes
BMRM	Primal	$O(\frac{nd}{\epsilon})$	Yes	Yes
DCDM	Dual	$O(nd \log \frac{1}{\epsilon})$	No	Yes

*Solves the problem for basis vectors, **Reaches accuracy ϵ with probability $1 - \delta$

A comparison of the properties of the SVM algorithms that we analyzed are summarized in Table 4.
SVM$^{\text{Perf}}$ is fast to reach convergence, behind DCDM only, and can be parallelized, but it has two drawbacks: classes cannot be easily balanced and there are no means to estimate Lagrange multipliers for the pushing approach.
Pegasos is not attractive because it is slower than the other implementations and cannot be parallelized.
BMRM is almost as slow as Pegasos but it can be implemented to exploit a distributed environment, furthermore it is the most flexible with respect to the loss function beyond L1 and L2.
DCDM is the fastest to converge both in time and number of required iterations on a single processor, but it cannot exploit a distributed architectures.

7. References

[1] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," Data Mining and Knowledge Discovery, Vol. 2, pp. 121–167. 1998.

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

[3] T. Joachims, "A support vector method for multivariate performance measures," Proc. ICML 2005, pp. 377–384, 2005.

[4] T. Joachims, "Training linear SVMs in linear time," Proc. KDD 2006 pp. 217–226, 2006.

[5] 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.

[6] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub–gradient solver for SVM," Proc. ICML 2007, pp. 807–814,, 2007.

[7] 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," Proc. ICML 2008, pp. 408–415, 2008.

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

[9] M. A. Zissman, "Comparison of Four Approaches to Automatic Language Identification of Telephone Speech," IEEE Trans. SAP, Vol. 4, n. 1, , pp. 31–44, 1996.

[10] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian Mixture Models," in Digital Signal Processing, vol. 10, no. 1-3, pp. 19 – 41, 2000.

[11] J.-L. Gauvain, A. Messaoudi, and H. Schwenk, "Language recognition using phone lattices," Proc. Interspeech 2004, pp. 2528.

[12] Campbell W.M. , Richardson F., Reynolds D.A., "Language recognition with word lattices and Support Vector Machines," Proc. ICASSP 2007, pp. 15–20, 2007.

[13] P. A. Torres-Carrasquillo, D. A. Reynolds, E. Singer, M. A. Kohler, R. J. Greene, and J. R. Deller, Jr., "Approaches to Language Identification Using Gaussian Mixture Models and Shifted Delta Cepstral Features," Proc. ICSLP 2002, pp. 89–92, 2002.

[14] L. Burget, P. Matejka, and J. Cernocky, "Discriminative Training Techniques for Acoustic Language Identification," Proc. ICASSP 2006, Vol I, pp. 209–212, 2006.

[15] W. M. Campbell, J. R. 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, pp. 210–229, 2006.

[16] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface and C. Vair, "Acoustic Language Identification Using Fast Discriminative Training," Proc. Interspeech 2007, pp. 346–349, 2007.

[17] W. M. Campbell, "A covariance kernel for SVM language recognition," Proc. ICASSP 2008, pp. 4141–4144, 2008.

[18] W.M. Campbell, J.P. Campbell , T.P. Gleason , D.A. Reynolds , W. Shen, "Speaker Verification Using Support Vector Machines and High–Level Features," IEEE Trans. ASLP, Vol 15, n. 7, pp. 2085–2094, 2007.

[19] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Compensation of nuisance factors for speaker and language recognition,", IEEE Trans. ASLP, Vol. 15, pp. 1969–1978, 2007.

[20] www.loquendo.com/en/technology/asr.htm

[21] 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," Proc. of ICASSP, 2006, pp. 97–100, 2006.

[22] www.itl.nist.gov/iad/mig/tests/lre/2009/LRE09_EvalPlan_v6.pdf

[23] P. Matejka, L. Burget, O. Glembek, P. Schwarz, V. Hubeika, M. Fapso, T. Mikolov, O. Plchot, J. Cernocky, "BUT language recognition system for NIST 2007 evaluations," Proc. Interspeech 2007, pp. 739–742, 2007.

[24] http://numpy.scipy.org/

[25] http://abel.ee.ucla.edu/cvxopt/