Solving Linear SVMs with Multiple 1D Projections

Johannes Schneider *
ABB Research, Zurich

Jasmina Bogojeska
IBM Research-Zurich

Michail Vlachos
IBM Research-Zurich

ABSTRACT

We present a new methodology for solving linear Support Vector Machines (SVMs) that capitalizes on multiple 1D projections. We show that the approach approximates the optimal solution with high accuracy and comes with analytical guarantees. Our solution adapts on methodologies from random projections, exponential search, and coordinate descent. In our experimental evaluation, we compare our approach with the popular liblinear SVM library. We demonstrate a significant speedup on various benchmarks. At the same time, the new methodology provides a comparable or better approximation factor of the optimal solution and exhibits smooth convergence properties. Our results are accompanied by bounds on the time complexity and accuracy.

Categories and Subject Descriptors

I.5.3 [**Classification**]: Algorithms

Keywords

Support Vector Machines (SVM), Classification, Data Mining, Random Projections, Coordinate Descent

1. INTRODUCTION

Classification is a key task in data analysis, and support vector machines (SVMs) belong to the state-of-art techniques for data classification. SVMs can be broadly categorized into linear and nonlinear (e.g., kernel-based). Linear SVMs fit linear boundaries in the original attribute space, whereas nonlinear SVMs first transform the data into a new, higher-dimensional space. Because linear boundaries in the transformed space correspond to nonlinear ones in the original space, nonlinear classification is made possible. Several research efforts have attested that nonlinear SVMs offer better classification accuracy than the linear ones [34, 17]. However, this comes at the expense of significantly higher computation cost [25], which renders them prohibitively expensive for large datasets.

Nonlinear SVMs do not offer additional benefits for datasets having more features than observations, where classes are linearly separable. This scenario is very frequent in genomic studies, where one typically collects only few data samples that contain measurements for thousands of genes.

In general, linear SVMs are better suited:

- when the data have more attributes than observations, then we do not really have enough data to fit a complex function, and a simple linear approach is most reasonable;

- for Big Data analytics and for exploratory/interactive data analysis because of their superior performance, i.e. their ability to quickly extract rudimentary data statistics; and

- in applications that require either a real-time response rate or conservation of energy resources (e.g., sensors). In such cases, obtaining a rough solution quickly may be preferable over getting an optimal solution slowly.

In this work, we present approaches for speeding up linear SVMs further while maintaining accuracy. The main idea is based on the fact that many problems, while difficult in high-dimensional spaces, may indeed be solved optimally and in a simple manner in one dimension. Consider for example, the general K-Means problem; even though it is NP-hard in high dimensions, 2-Means clustering can be solved exactly and efficiently for a single dimension [5].

Support vector machines seek to maximize the margin between the hyperplane that separates two classes. This problem can be solved efficiently in 1D [30]. Therefore by combining multiple 1D solutions, we can progressively bound the error of the SVM solution. In general, we make the following **contributions**:

1. We present a new and simple-to-implement methodology for solving linear SVMs. The technique capitalizes on multiple 1D projections and SVM solutions to estimate the class-separating hyperplane in the original data dimensionality. First, we examine a naive random projection approach. This serves as an initial testbed for establishing a pessimistic bound on the number of 1D projections that are required, and the number of 1D separating hyperplane problems that need to be solved, to approximate the original hyperplane solution with arbitrary accuracy. The second approach is of general practical interest. It uses a local search mechanism to probe rotations around the currently estimated hyperplane to improve the current solution. This second

*Work conducted while at IBM Research - Zurich

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2661994 .

algorithm in addition exploits notions from coordinate descent [6] and exponential search [3].

2. By working on 1D, we can reduce the computational complexity because distances between points are faster to compute. More importantly we offer analytical guarantees on the quality of the solution provided; a solution that is computed on 1D projections but estimates the original, high-dimensional, linear SVM problem.

3. We compare the runtime and quality of these new algorithms with the popular `liblinear` SVM library. For an extended number of benchmarks, we show that the new methods achieve higher accuracy and lower runtime than the state-of-art algorithms for solving linear SVMs.

In general, our methodology offers new directions for solving linear SVM problems and owing to its low computational demands, it is particularly applicable on very large datasets.

The remainder of the paper is structured as follows: First we give the formulation of linear SVMs in high dimensions and review related work. In Section 3, we show how the linear SVM problem can be solved efficiently in the primal space on a single dimension and offer a complexity analysis. In Section 4, we combine multiple 1D SVMs and provide an error bound on the number of 1D projections that are required. In Section 5, we present a more intelligent approach based on local search. Finally, in the experimental section, we provide scalability and accuracy comparisons against various algorithms from the `liblinear` SVM library.

2. BACKGROUND

We describe the general formulation of linear SVMs and ways to solve the optimization problem. Assume that the training data consist of N points $X = \{x_i | x_i \in \mathbb{R}^d, i \in [1..N]\}$, each point x_i having a binary class label $y_i \in \{-1, 1\}$. The goal is to compute a hyperplane $H_{opt} := (w_{opt}, b_{opt})$ that separates the two classes with maximum margin as shown in Figure 1.

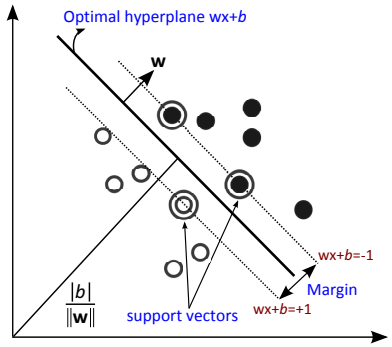

Figure 1: A Linear Support Vector Machine classifier for two classes.

The class of a point a for a hyperplane $H := (w, b)$ is given by the sign of the term $w \cdot a + b$. As point sets are in general not perfectly separable, each point x_i is accompanied by a slack variable ζ_i that accommodates misclassification of points. Formally speaking, we want to find (w, b) with $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that we

$$\text{minimize } ||w||^2/2 + c \cdot \sum_{i \in [1,N]} \zeta_i^p \quad (1)$$

subject to

$$y_i(x_i w + b) \geq 1 - \zeta_i, \forall i \in [1, N] \quad (2)$$

$$\zeta_i \geq 0, \forall i \in [1, N]. \quad (3)$$

Parameter c determines the trade-off between misclassifying a point x_i (which is the case for $\zeta_i > 0$) and maximizing the margin. Setting $p = 1$ we get the L1-SVM and for $p = 2$ the L2-SVM. Note that L1-SVM is a convex optimization problem, and L2-SVM is a strictly convex optimization problem.

Solving high-dimensional SVMs: To solve the above optimization problem efficiently, often the dual formulation is considered. Prevalent methods for solving SVMs include coordinate descent techniques [6, 33, 12], cutting plane techniques [14], the finite Newton method [16], alternations between stochastic gradient descent steps and projection steps [29], and the trust region Newton method [21]. Iterative, anytime SVM approaches using Cholesky decompositions are presented in [9]. A survey on large-scale linear classification focusing on L1/L2 regularized L1/L2-loss SVM and logistic regression can be found in [35].

A key design choice is whether to optimize the primal or the dual formulation. Although there are many algorithms based on the dual SVM formulation, there are also arguments in favor of using the primal one [7].

Using Random Projections: In our approach we use multiple 1D random projections. Random projections are often associated with the Johnson–Lindenstrauss Lemma [15]: points in a high-dimensional Euclidean space can be projected into a low-dimensional Euclidean space with tight bounds on distance preservation. A random projection L is a random vector originating from the origin and going to a randomly chosen point in the d-dimensional unit sphere. Random projections have been already used in the field of SVMs. Osadchy et al. [23] assume that the negative class is very large and approximate it with a Gaussian distribution. They separate the (1D) projection of this distribution from the positive samples using a "hybrid" prior. In the Krishnan et al. [18] show that by using random projections with $k = O(1/\epsilon^2 \log N)$ dimensions one can approximate the optimal SVM solution within a factor of $(1 + \epsilon)$. Boutsidis et al. [24] demonstrate that when using random projections for linear SVMs, the margin and minimum enclosing ball in the feature space are preserved within a small relative error. Finally, [26] uses random features to approximate nonlinear shift-invariant kernel functions with sinusoids randomly chosen from the Fourier transform of the kernel function.

Note also that for the above techniques, the recommended logarithmic (to the number of objects) projected dimensionality may prove impractically high for Big Data applications. In contrast to the above, we only perform projections on 1D, where all operations (sorting, searching and distance computations) are faster and simpler.

To simplify notation and exposition of ideas, we focus on binary SVMs in the remainder of the paper, which can also be used as the basis for constructing multi-class SVMs [8, 28, 32].

3. SOLVING THE 1D LINEAR SVM

We first describe how linear SVMs can be solved efficiently on one dimension. The process basically resorts to a sorting of points. We use the primal formulation of the optimization problem as described in Section 2. Using a Lagrange multiplier α_i for each constraint in (2) and β_i for each constraint in (3) results in the following Lagrange primal function for minimization

$$l_p = ||w^2||/2 + c\sum_i \zeta_i - \sum_i \alpha_i(y_i(x_i \cdot w + b) - 1 + \zeta_i) - \sum_i \beta_i \zeta_i. \quad (4)$$

The Karush–Kuhn–Tucker (KKT) conditions [4] provide the necessary conditions to minimize the Lagrange function. They also allow us to formally prove several interesting simplifications for the 1D case [30][1]. The two most important observations are:

i Let n^+ be the number of points of class 1 that are wrongly classified, i.e., with $\zeta_i > 0$ (and analogously n^- the number of misclassified points of class -1). We have $n^+ = n^-$, i.e., the number of wrongly classified points is the same for each class.

ii Let α^+ be the sum of all α_i for the positive support vectors (and α^- for the negative support vectors). It holds $\alpha^+ = \alpha^-$.

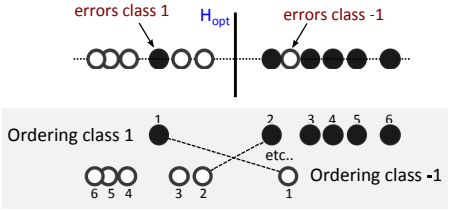

Figure 2: One-dimensional SVM with optimal hyperplane H_{opt}. To compute H_{opt}, it suffices to consider all (up to) $N/2$ pairs of possible support vectors. Pair i consists of point i from class 1 and point i from class -1.

Therefore, it suffices for 1D SVMs to perform a linear scan through all sorted points, see Figure 2 for an example. One can commence from the two most distant points from each class, i.e., one point from class 1 and one point from class -1, and assume that they are the two support vectors. For these two points, the weight, the bias and the Lagrangian l_p are computed as follows:

The 1D weight w^{1d} at iteration i is the midpoint of the current support vectors. To compute the bias b, one can use a constraint for the current support vectors, i.e.,

$$1 = w^{1d} \cdot PLS_i - b$$

where PLS_i is a support vector. To compute the sum of the slack variables $\zeta := \sum_i \zeta_i$, we use the equation

$$\zeta_i := 1 - y_i(w^{1d} \cdot L \cdot P_i - b)$$

[1]In [30] Observation 4 states that "All positive support vectors have the same coordinate.", which does not hold in general for non-linearly separable data. In fact, the authors mean that all points with $y_i \cdot (x_i \cdot w + b) = 1$ have the same coordinate.

and simplify it using the above observations (i-ii), which leads to

$$\zeta := 2 \cdot (n - i) + w^{1d} \cdot (S_1(i) - S_{-1}(i))$$

where n is the number of points in class 1, and $S_1(i)$ and $S_{-1}(i)$ are sums of coordinates of points from class 1 and -1, respectively, as defined in line 5 of Algorithm SVM1D. Once the loss function (Lagrangian) is computed, the previously considered points are removed. From the remaining ones, we choose again the two most distant points from each class. The procedure is repeated for all remaining pairs. The resulting solution is given by the pair of points resulting in the minimum *loss* l_p (4).

The pseudocode of the above procedure is given in Algorithm SVM1D.

Algorithm 1 SVM1D(input: points $\mathcal{P}$, labels $\mathcal{LA}$, projection line L; output: minimum weight w^{1d}_{min}, bias b_{min} and loss l_{min})

1: $PL := \{L \cdot P | P \in \mathcal{P}\}$ {Project points onto line L}
2: Sort projected points PL
3: $PLS_1 := (L \cdot P_0, L \cdot P_1, ..., L \cdot P_{|\mathcal{P}|} | L \cdot P_i \leq L \cdot P_{i+1} \wedge \mathcal{LA}(P_i) = 1)$ {Ascending projected points of class 1}
4: $PLS_{-1} := (L \cdot P_0, L \cdot P_1, ..., L \cdot P_{|\mathcal{P}|} | L \cdot P_i \geq L \cdot P_{i+1} \wedge \mathcal{LA}(P_i) = -1)$ {Descending projected points of class -1}
5: $S_{c \in \{1,-1\}}(i) := \sum_{k>i, L \cdot P_k \in PLS_c} L \cdot P_k$
6: $l_{min} := \infty$
7: **for** $i = 1..\min(|PLS_{-1}|, |PLS_1|)$ **do**
8: $w^{1d} := 2/(PLS_1(i) - PLS_{-1}(i))$ {weight in 1D}
9: **if** $w^{1d} < 0$ **then**
10: $b := 1 - PLS_1(i) \cdot w^{1d}$
11: $\zeta := 2 \cdot (n - i) + w^{1d} \cdot (S_1(i) - S_{-1}(i))$
12: $l_p := w^{1d} \cdot w^{1d}/2 + c \cdot \zeta$ {Loss with regularization constant c}
13: **if** $l_p < l_{min}$ **then**
14: $l_{min} := l_p$
15: $b_{min} := b$
16: $w^{1d}_{min} := w^{1d}$
17: **end if**
18: **end if**
19: **end for**
20: $w_{min} := w^{1d}_{min} \cdot L$ {high-dimensional scaled normal vector of hyperplane}

Now, we analyze the complexity of the SVM1D algorithm. We prove that the algorithm returns the optimal result for a given direction. If the direction is far from the optimal direction, the solution might also be far from the overall optimal solution because the direction given is not modified by the algorithm.

THEOREM 3.1. *Algorithm SVM1D executes in time $O(N \log N + N \cdot s)$ for a projection L with s nonzero entries. It computes an optimal hyperplane (w^{1d}, b), with $w^{1d}, b \in \mathbb{R}$ for the given projection L.*

PROOF. Projecting all N points in d dimensions onto random line L with s nonzero entries requires $N \cdot s$ time. The sorting of the projected points requires $O(N \log N)$. Iterating through all N points (lines 7–19 in Algorithm SVM1D) requires time $O(N)$. This yields an overall time complexity of $O(N \log N + N \cdot s)$. To show correctness, let $w_{opt}(L) := w^{1d}_{opt} \cdot L$ and b_{opt} be an optimal separating hyperplane for a given projection L minimizing equation (4), where w^{1d}_{opt} is the optimal scaling factor. Therefore a point

x_i is classified as either class 1 or class -1 using

$$sign(w_{opt}x_i + b_{opt}) = sign((w_{opt}^{1d} \cdot L)x_i + b_{opt})$$
$$= sign(w_{opt}^{1d} \cdot (L \cdot x_i) + b_{opt}).$$

Therefore to solve the problem optimally for a given direction L in high-dimensional space, we can derive a classifier for the projected points $L \cdot x_i$ in one dimension. □

Using Algorithm SVM1D, the linear support vectors can be discovered optimally on 1D. The interested reader can find the proof in [30].

Now a question that naturally arises is: Given multiple 1D SVM solutions, can we somehow combine them to approximate the original, high-dimensional hyperplane? We address this question in the next sections.

4. APPROACH 1: NAIVE RANDOM PROJECTIONS

Here we examine how to combine multiple solutions of 1D random projections for solving the high-dimensional SVM problem. We perform an initial analysis that reveals a (pessimistic) upper bound on the number of random projections required for approximating the linear SVM with arbitrarily high accuracy. Later we improve on this solution by performing a more intelligent search process.

Assume that we pick a random projection line of unit length, which serves as the normal vector of the hyperplane. We project all points onto the random normal vector. In the preceding section we showed how to solve the linear SVM problem optimally in one dimension. Thus, the problem reduces to finding two scalars for the hyperplane chosen: the optimal bias b_{opt} and the optimal 1D weight w_{opt}^{1d} computed by algorithm SVM1D (Section 3). The weight is used to scale the random line chosen.

One can repeat the process of choosing a hyperplane, projecting points and solving the 1D SVM problem such that in each iteration the hyperplane is chosen independently of any previously selected random planes. In this way, the data are projected multiple times, and also the 1D problem is solved multiple times. Random projections are chosen independently, and the error (i.e., the loss function) for each trial is computed. The final result is the projection with the smallest loss. The process is summarized in Figure 3.

Each of the above iterations is very efficient. Choosing a random hyperplane and projecting all points onto the hyperplane takes $(N+1) \cdot d$ time. The simplest random projection is given by choosing each entry of the vector independently from the standard normal distribution $\mathcal{N}(0,1)$. There is no need for normalization, as we only consider the direction of the vector and not its magnitude. Computing the support vectors in one dimension is also computationally inexpensive, requiring $O(N \log N)$, as was shown in Theorem 3.1.

Error Analysis: Here we provide probabilistic bounds on the trials required so that a chosen vector v is within an angle ϕ_0 of the optimal vector w_{opt} (see Figure 4). The actual prediction error and its sensitivity depend on the margin of the classifier and therefore on the distribution of the data.

Denote $\phi(a,b)$ as the angle between vectors a and b.

THEOREM 4.1. *For any fixed vector a after at most $2c_0 \cdot e^{2\sqrt{d}+2} \log N$ choices of random vectors at least one vector v is such that $\phi(a,v) < \phi_0$ with probability $1 - 1/N^{c_0/2}$ for an arbitrary constant c_0.*

PROOF. The proof can be found in the appendix. □

The theorem basically states that the algorithm will find a solution that is arbitrarily close to the desired high-dimensional hyperplane, but the exploration space can be vast: we have an exponential dependence on the number of dimensions. The following section revisits the search process, and shows that the search space can be substantially pruned by accommodating an intelligent selection of the 1D hyperplanes.

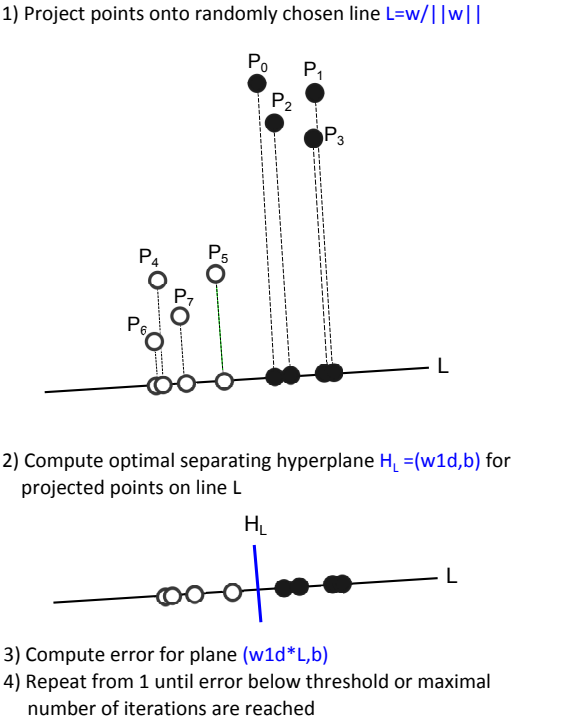

Figure 3: Illustration of algorithm `ProjectSVM` for 8 points taken from two classes

Algorithm 2 ProjectSVM(input: points X, class labels Y, maximal iterations $maxIter$; output: weights w_{min}, bias b_{min})

```
1: l_min := ∞
2: for i = 1..maxIter do
3:    Choose a random projection line L ∈ R^d with each coordinate chosen from N(0,1)
4:    (w^1d, b, l) := SVM1d(X, Y, L)
5:    if l < l_min then
6:       l_min := l
7:       b_min := b
8:       w_min := w^1d · L/||L|| {scaled normal vector of hyperplane}
9:    end if
10: end for
```

5. APPROACH 2: LOCAL SEARCH

In this section, we present our main algorithm `LocalSVM`, which improves on the preceding naive approach by selecting

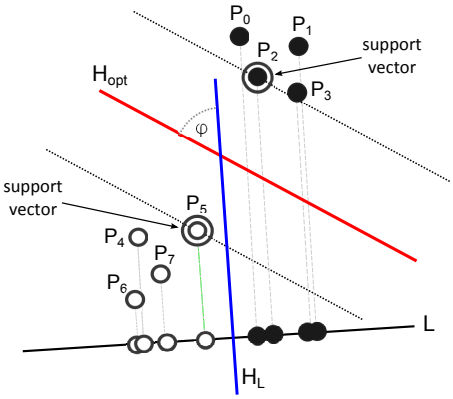

Figure 4: The angle ϕ between the optimal hyperplane for the point set and the hyperplane guessed by algorithm `ProjectSVM`.

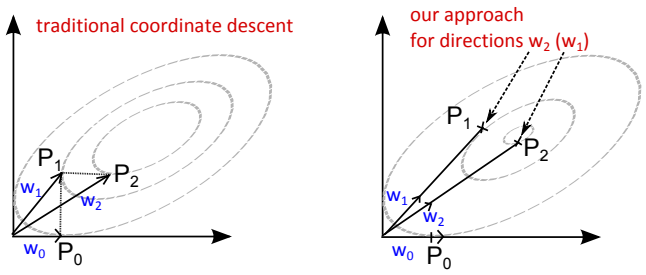

Figure 5: On the left, we show three iterations of traditional coordinate descent techniques for an elliptic function. The right panel shows two solution vectors of our local search approach. For each direction w_0, w_1 and w_2, we compute the optimal solution in 1D, which yields points P_0, P_1 and P_2.

each projection line via intelligent local search. This ensures fast and smooth convergence.

The `LocalSVM` algorithm follows a similar outline as before: points are projected onto a line, and the problem is solved optimally on that line. Now, however, we implement a local search that sequentially modifies each coordinate on the currently best projection known. This can be regarded as an alternative way of doing coordinate descent. Standard techniques for coordinate descent solve a target problem optimally for a single coordinate while keeping all the remaining coordinates fixed. Our approach, however, differs in the following way: We also modify the value of a single coordinate of the currently best hyperplane known to select a new direction. But then we consider arbitrary magnitudes in the new direction to determine its optimal solution, which modifies all coordinates of the current solution vector rather than a single one. This is illustrated by the example in Figure 5. Note also that while our approach considers potential future directions with arbitrary lengths and small angle to the previous one, standard coordinate descent approaches generally consider rather shorter solutions with larger angle to the previous one. As we show in the experimental section, our approach leads to smoother convergence.

The main *motivation* for considering each coordinate separately is computational efficiency. Recall that the complexity of our naive algorithm `ProjectSVM` is heavily influenced by the time spent on projecting points. Without any assumptions a single projection requires $O(N \cdot d)$ time. However, we can exploit the linearity of the scalar product to reduce the time complexity. For a projection w composed of two other projections, i.e., $w := w_{min} + w'$ and any point x_k, we have

$$w \cdot x_k = (w_{min} + w') \cdot x_k = w_{min} \cdot x_k + w' \cdot x_k.$$

Because $w_{min} \cdot x_k$ has been precomputed (in a prior iteration or before the first iteration), the time to compute $w' \cdot x_k$ is proportional to the nonzero entries in w'. Thus, choosing a sparse vector w' has a positive impact on the performance. For a projection onto a singular coordinate, there is only one nonzero entry in w'.

Algorithm `LocalSVM`: Here we provide more details on the `LocalSVM` method. Let w denote the currently best direction. Each iteration i consists of one or two passes performed for each coordinate. In the first pass, t is added to the i-th coordinate of w to yield w'. Then the loss for the projected values $(x_i \cdot w', i \in [1, N])$ is evaluated by solving the SVM1d problem. If it significantly decreases the loss, the new direction is scaled with the weights obtained from SVM1d (w^{1d}). If not, a second pass is performed where the i-th coordinate is evaluated in the same way, now by subtracting instead of adding t from the i-th coordinate of w. It is not essential that t is added (or subtracted). A coordinate might as well be multiplied or divided by a value. However, it is important that the value is adjusted in the right manner, i.e. using an exponential scheme. The set of d iterations through all dimensions denotes one *phase*. Following that, the algorithm commences the next phase and considers one coordinate after the other, i.e., in the i-th iteration of the j-th phase, we modify the i-th coordinate. During a phase, the value t that is added to (or subtracted from) each coordinate of the solution vector w remains fixed. After every phase, t is reduced or increased by a constant factor c_t using an exponential-like search as follows: The value t is multiplied by either a factor $1/c_t$ or a factor c_t. When in the prior phase t was multiplied by $v \in \{1/c_t, c_t\}$ and there was a "sufficient" decrease of the loss in the current phase, then t is again multiplied by the same value v. If the change was not sufficient then we multiply by $1/v$. Note that if the loss is not sufficiently decreased in two subsequent phases the threshold *thres* in the Algorithm `LocalSVM` is decreased substantially to avoid that t repeatedly alternates between two values.

Convergence: Now, we formulate the convergence properties of the `LocalSVM` algorithm. We show that our method converges for any strictly convex function, such as the l_2 loss function [1].

THEOREM 5.1. *For strictly convex functions f, the algorithm `LocalSVM` converges towards the optimal solution.*

PROOF. For f a strictly convex function, for any non-optimal point w there must be a $t^* > 0$ and a coordinate j such that

$$f(w + t^* e_j) < f(x) \quad \text{or} \quad f(w - t^* e_j) < f(x)$$

where e_j is the unit vector with all coordinates set to zero and coordinate j set to 1. Let w_i be the best solution found up to phase i. Thus, if there is no improvement for

Algorithm 3 LocalSVM(input: points X, class labels Y, $nIter$; output: weights w_{min}, bias b_{min} and loss l_{min})

$t := 1$
$c_t := 2$ {value to increase/decrease t}
$curr := c_t$ {factor by which t is changed, either c_t or $1/c_t$}
$thres := 1.1$ {Initial threshold for change in loss function}
$w_{min} := (0, 0, ..., 0)$ {Initial vector for hyperplane w}
$l_{last} = 0$ {Value of loss in the last phase}
$l_{min} = \infty$ {Initial value of the minimum loss}
$NoLossDecrease := false$
for $i = 0..nIter$ **do**
  $coord := i \mod d + 1$ {Coordinate that is modified}
  {Check if a new phase begins}
  if $coord == 1$ **and** $i > 0$ **then**
    if $l_{last} < l_{min} \cdot thres$ **then**
      if $NoLossDecrease == true$ **then**
        $thres := 1 + (thres - 1)/c_t^4$
        $curr := 1/c_t$
      else
        $curr := 1/curr$
      end if
      $NoLossDecrease := true$
    else
      $NoLossDecrease := false$
    end if
    $t := t \cdot curr$
    $l_{last} := l_{min}$
  end if
  $isMinDir := false$
  for each $sign \in \{-1, +1\}$ **and if** $isMinDir = false$ **do**
    $w' :=$ d-dimensional vector with zeros except $coord$th coordinate being $sign \cdot t$
    $w := w_{min} + w'$
    $wX := (x_1 \cdot w, x_2 \cdot w, ..., x_N \cdot w)$
    $(w^{1d}, b, l) := SVM1d(X', Y, w)$
    if $l \leq l_{min}$ **then**
      $isMinDir := true$
      $l_{min} := l$ {New best solution}
      $b_{min} := b$
      $w_{min} := w \cdot w^{1d}$
    end if
  end for
end for

phase i then $w_i = w_{i-1}$. Define an unsuccessful phase as a phase in which the loss has not decreased "sufficiently": $f(w_i) \cdot thres > f(w_{i-1})$. Let u be the number of consecutive unsuccessful phases starting at phase i. Let $scale_i$ (denoted as t in the algorithm) and $thres_i$ be the values used in phase i of the algorithm LocalSVM. For $u = 2$, the threshold (minus 1) is decreased by c_t^4 with $c_t > 1$: $thres_{i+2} := 1 + (thres_i - 1)/c_t^4$. Furthermore, the current update value $curr$ for $scale_i$ will be set to $1/c_t$, i.e., $scale$ will be decreased by a factor of $1/c_t$. To get an upper bound for $scale_{i+2}$, assume that $scale$ was increased by c_t in phase i and decreased by c_t in phase $i + 1$, thus $0 < scale_{i+2} \leq scale_i/c_t^2$. For $u = 4$, we have that $thres$ and $scale$ are both decreased: $0 < scale_{i+4} \leq scale_i/c_t^4$ and $thres_{i+4} := 1+(thres_i-1)/c_t^8$. This pattern repeats for consecutive unsuccessful phases, i.e., for $u > 4$ we get $0 < scale_{i+u} \leq scale_i/c_t^u$ and $thres_{i+u} := 1 + (thres_i - 1)/c_t^{4\lfloor u/2 \rfloor}$. Thus, at some point, $scale$ must be smaller than t^* which yields the function f decreased. As $thres - 1$ decreases much faster than $scale$ (by a factor c_t^4 versus c_t^2 for two consecutive unsuccessful phases), we have that $thres$ reaches 1 much faster than $scale$ reaches 0. Eventually, for some phase j, the (rela-

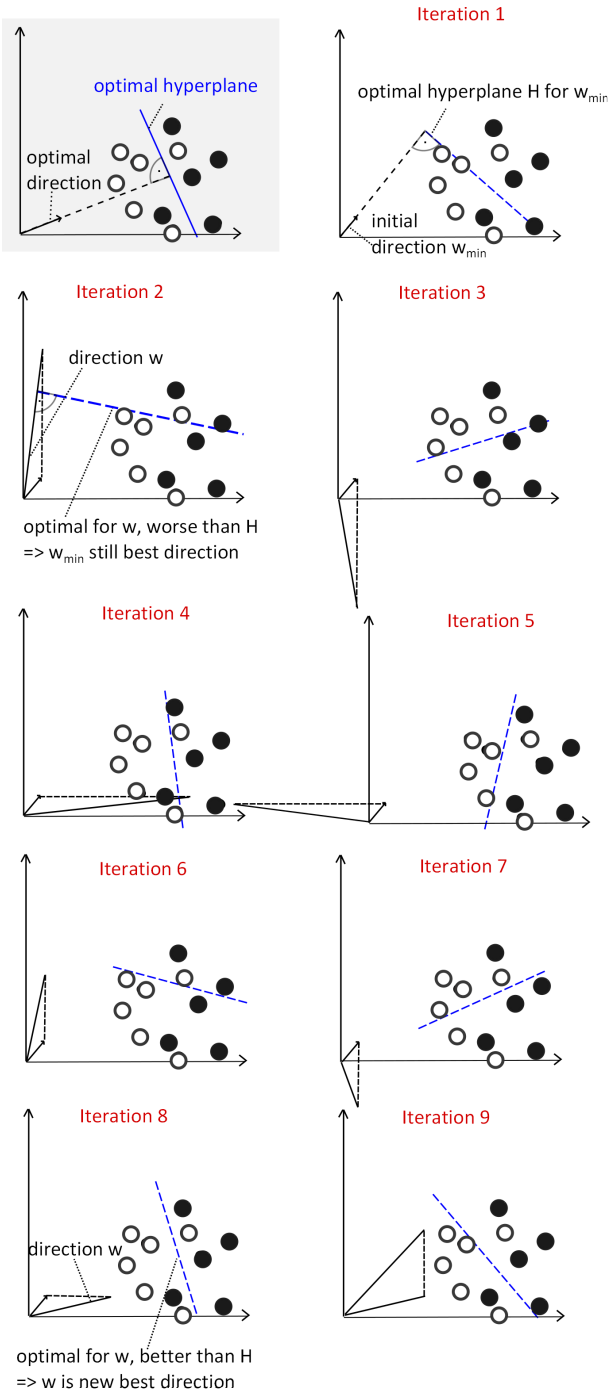

Figure 6: Illustration of Algorithm `LocalSVM`. After computing the optimal hyperplane for the initial direction v, one coordinate of the currently best solution known is modified in each iteration. At Iterations 2 and 3, the x-coordinate is changed by a fixed magnitude t. In Iterations 4 and 5, the y coordinate is changed by the same t. At Iteration 6, the process repeats for a smaller $t := t/c_t = t/2$. At Iteration 8, the classifier for v' becomes the new best classifier. At Iteration 9, the x-coordinate is changed by t.

tive) change of function f for some coordinate is larger than $thres$: $f(w_i) \cdot thres < f(w_{i-1})$. This phase j is successful. Then for some k, $scale_{j+k} \geq scale_j$ and all $r \in [j, j+k]$, there exists a coordinate i such that
$f(w_r + scale_r e_i) \cdot thres < f(w_{r-1})$
or $f(w_r - scale_r e_i) \cdot thres < f(w_{r-1})$.
In other words, in each phase $r \in [j, j+k]$, the solution w_r gets improved. Eventually, $thres$ gets arbitrarily close to 1, and the current best solution w_r gets arbitrarily close to the optimal solution w_{opt}. □

Above, we showed the convergence for (strictly) convex functions [22, 12, 33, 6]. Note, however, that it is hard and still largely an open problem to prove the rate of convergence for cyclic coordinate search in general [22].

As we will show through extensive evaluations in the following section, the runtime of `LocalSVM` is superior to that of the naive approach `ProjectSVM`. This makes it an attractive method for solving large-scale linear SVM problems.

6. EMPIRICAL EVALUATION

Here we evaluate the runtime and accuracy of the proposed methods. We compare their performance with that of two algorithms for linear SVMs implemented in the `liblinear` framework [10]:

- Algorithm `L2RL1LOSSSVCDUAL` by Hsieh et al. [12]. This method uses a dual coordinate descent approach. For each coordinate, it performs several iterations until the optimal solution has been found. We compare with this algorithm, because in [12] the authors show that their approach outperforms other state-of-the-art techniques such as Pegasos [29] (one of the de-facto stochastic gradient methods for SVMs), as well as TRON [21] and SVM^{perf} [14].
- Algorithm `L2RL1LOSSSVRDUAL` [34], which considers support vector regression, in which only a single slack variable is optimized for all constraints. It uses Newton steps.

Our algorithms are the naive `ProjectSVM` and the more intelligent `LocalSVM`. All code is implemented in Java and experiments have been conducted on a 2.6 GHz Intel CPU with 16 GB RAM [2].

Datasets: We consider a variety of datasets that have been used for evaluating linear binary classifiers from the UCI machine learning repository[3] and datasets that come with the `liblinear` framework [10].[4] We purposefully focus on *dense* data, that is, data with (mainly) nonzero values, so as not to give an unfair advantage to our approach; because sparse high-dimensional data can be compressed efficiently using random projections with very little loss of accuracy [2, 20]. A summary of the datasets used in our experiments is given in Table 1.

Benchmark set-up: For each algorithm, we run each benchmark for $(1, 2, 4, ..., 2^{18})$ iterations. For our algorithms, we used the parameters stated in the pseudocodes. For the solvers from `liblinear`, we retained the default settings.

[2]Java is a registered trademark of Oracle and/or its affiliates. Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.
[3]`http://archive.ics.uci.edu/ml/datasets.html`
[4]`www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Name	Type	Objects	Dim
Record Linkage[3]	Person records	5,734,488	12
SkinNonSkin[3]	Images	245,057	4
Epsilon [4]	PASCAL '08 challenge	100,000	2000
Codrna [31]	Gene data	59,535	8
MiniBooNE [27]	Astronomy	50,000	50
Magic04[3]	Astronomy	19,020	10
Musk[3]	Astronomy	6,598	168
Svmguide1 [13]	Artificial	3,089	4
Madelon [11]	NIPS '03 challenge	2,000	500
Gisette [11]	Letters	6,000	5,000
Svmguide3 [13]	Artificial	1,243	22
Splice[4]	DNA	1,000	60
Diabetes[4]	Medical	768	8
Coloncancer[4]	Medical	62	2,000

Table 1: Datasets used in the experiments.

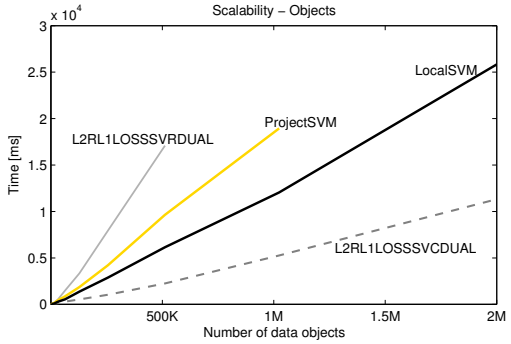

Figure 7: Scalability experiment for increasing number of data objects.

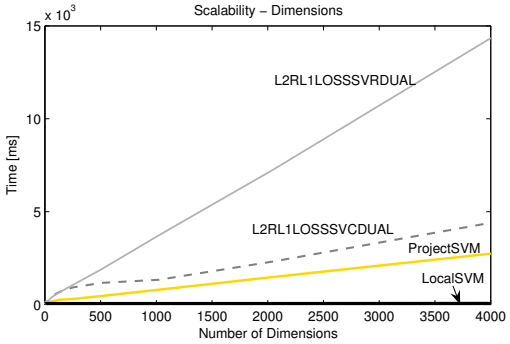

Figure 8: Scalability experiment for increasing data dimensionality.

Scalability: We examine scalability both for increasing number of objects and for increasing data dimensionality using a fixed number of iterations.

Figure 7 illustrates the first experiment, in which we evaluate runtime against increasing object cardinality (for fixed dimensionality). For this experiment, we used the 'Record Linkage Comparison Patterns' dataset[3], which contains more than 5 million people records. We observe that the runtime of our two methods is not substantially different from that of the competing approaches. Time complex-

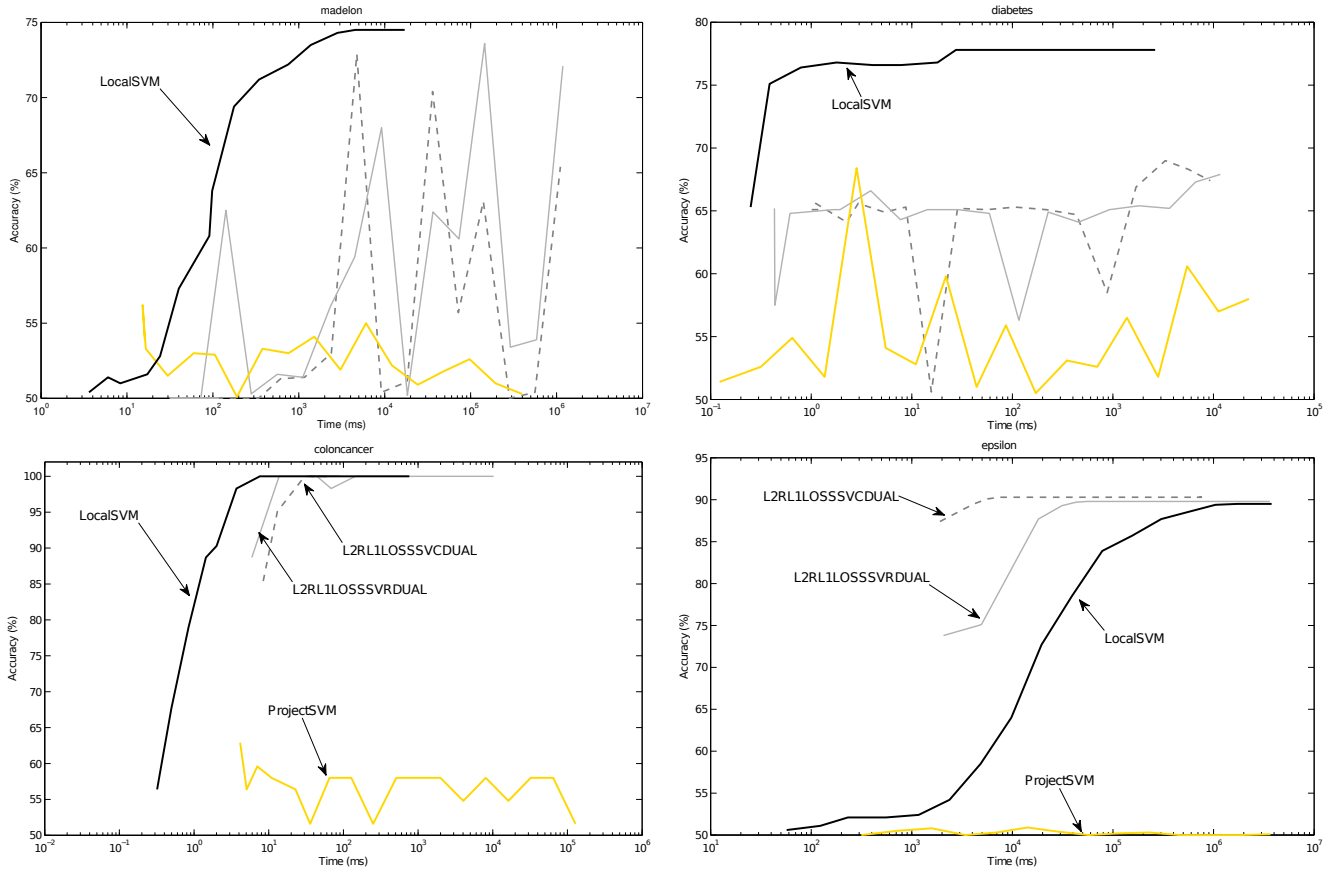

Figure 9: Time versus accuracy graphs. Notice the smooth convergence of the `LocalSVM` approach proposed.

ity per object is higher for `LocalSVM` than for `ProjectSVM` because of the point sorting involved in the former. Note that for this experiment we do not report the classification accuracy, which may have been different for each approach. We examine this trade-off between speed and accuracy in the following section. The goal of this first experiment was merely to show that the techniques presented exhibit a runtime that is on par with existing approaches.

In the second experiment, we fix the number of objects and test the runtime for increasing data dimensionality. For this experiment, we used the Gisette dataset because it is the one with the highest dimensionality. Figure 8 summarizes the results. We note the excellent performance of our approaches, and in particular of the `LocalSVM` approach, whose runtime remains at the same level when increasing the number of dimensions. The time complexity of algorithm `ProjectSVM` grows linearly with the number of points and dimensions, as every additional dimension must be taken into account in every projection (iteration).

In summary, because our techniques always operate on the projected 1D space, their performance is not substantially affected by the data dimensionality. Therefore, we believe that the algorithms presented constitute good candidates for disciplines faced with high-dimensional data (medicine, multimedia, etc).

Accuracy versus time: The motivation of our work is to provide faster (approximate) solutions to linear SVMs while not compromising the accuracy. Figure 9 shows the trade-off between accuracy and time for four exemplary datasets, and for up to 1024 iterations per run for each method. For all benchmarks, our `LocalSVM` algorithm outperforms both solvers of the `liblinear` framework in the accuracy range above $65 - 70\%$.

What is also noteworthy is that `LocalSVM` exhibits very *smooth convergence* properties. This is clearly depicted in all four graphs of Figure 9. In contrast, the general accuracy behavior of the `liblinear` solvers exhibits an oscillating pattern. We define the smoothness θ as follows: For each benchmark, we run each algorithm for a sequence of iterations, i.e., 2, 4, 8, etc. This yields a sequence of accuracies $\mathbf{a} := (a_0, a_1, ..., a_m)$. The smoothness is given by

$$\theta := \frac{\sum_{i=0}^{m-1} |a_{i+1} - a_i|}{\max(\mathbf{a}) - \min(\mathbf{a})},$$

where $\max(\mathbf{a}) - \min(\mathbf{a})$ is the difference between minimum and maximum accuracy. Table 2 lists the smoothness θ across all datasets. In the table, we present normalized θ values, where values closer to 1 (one) are better and higher numbers indicate worse results.

Finally, we run an experiment in which we let each technique run until it reaches up to 1% of the optimal accuracy or until a maximum runtime of 1 min is reached. This is similar to the experiment conducted in [12]. In Table 2, we report normalized accuracy results, where 100 indicates that the method achieved optimal accuracy. Note that for the majority of datasets, `LocalSVM` achieves a better accuracy than the other techniques.

	Smoothness			Maximal Accuracy[%]			$t_{max.Accuracy}$[ms]		
Name	LocalSVM	SVCDUAL	SVRDUAL	LocalSVM	SVCDUAL	SVRDUAL	LocalSVM	SVCDUAL	SVRDUAL
MiniBooNEPID	**1.00**	1.45	1.20	84	87	**89**	4296.1	**1604.1**	23545.9
Musk	**1.00**	1.30	1.13	96	**100**	96	**7340.4**	25822.8	9317.9
RecordLinkage	**1.00**	**1.00**	**1.00**	**100**	**100**	**100**	2292.1	**973.1**	1051.0
SkinNonSkin	**1.00**	2.09	1.57	**84**	82	82	311.8	**184.4**	282.7
codrna	**1.00**	2.20	1.43	**94**	89	90	**2919.0**	5007.2	4290.0
coloncancer	1.35	**1.00**	1.06	**100**	**100**	**100**	**1.7**	7.5	6.3
diabetes	**1.00**	1.45	1.47	**77**	67	67	**1.8**	6535.1	13311.6
gisette	1.50	1.07	**1.00**	**100**	**100**	**100**	3161.5	1211.0	**741.7**
madelon	**1.00**	1.17	1.13	**75**	62	71	1205.8	12806.0	**768.6**
magic04	**1.00**	1.36	1.65	**79**	76	74	263.8	38.9	**33.5**
splice	**1.00**	1.09	2.34	**85**	84	82	41.8	**2.6**	308.2
svmguide1	**1.00**	2.32	2.07	**95**	84	83	**1.7**	380.0	3.8
svmguide3	1.06	1.01	**1.00**	**82**	79	78	46.8	0.5	**0.5**

Table 2: Benchmark results for normalized smoothness of convergence (1 is better), maximal accuracy (100 is better) and time until the accuracy is within 1% of the maximal accuracy

7. CONCLUSIONS

Previous work on random projections examined the theoretical dimensionality on which to project high-dimensional data so that the SVM problem can be solved faster and with provable guarantees [24, 18]. In practice, the recommended logarithmic (to the number of points) projected dimensionality may prove impractically high for Big Data applications. In this work, we have shown that performing multiple one-dimensional projections provides fast and provably good results for linear SVMs. Further contributions of this work include the following:

1. Analytical bounds on the approximation quality of our algorithms and a comprehensive complexity analysis of the methods presented.

2. Extensive empirical comparison with algorithms from the `liblinear` framework. Our results suggest that the `LocalSVM` algorithm presented exhibits smooth convergence properties and leads to important computational savings.

Finally, our method has two direct and exciting implications: a) Distributed SVM execution, where each execution site holds a different 1D projection, and b) Inherent support for data obfuscation and data privacy. Because each site only has access to its individual 1D projection, it becomes very challenging to make inferences about the original high-dimensional data.

Acknowledgements: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 259569.

8. REFERENCES

[1] S. Abe. Analysis of support vector machines. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 89–98. IEEE, 2002.

[2] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

[3] J. Bentley and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[5] P. Brucker. On the complexity of clustering problems. In *Proc. of Optimization and Operations Research*, pages 45–54, 1977.

[6] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale L2-loss linear support vector machines. *Journal of Machine Learning Research*, 2008.

[7] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5), 2007.

[8] K. Crammer and Y. Singer. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[9] D. DeCoste. Anytime query-tuned kernel machines via Cholesky factorization. In *SIAM Int. Conf. on Data Mining*, 2003.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008.

[11] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. *Advances in Neural Information Processing Systems*, 17:545–552, 2004.

[12] 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 the 25th International Conference on Machine Learning*, 2008.

[13] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification, 2003.

[14] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[15] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. In *Contemporary Mathematics*, 1984.

[16] S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6(1):341, 2006.

[17] S. S. Keerthi and C.-J. Lin. Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. *Neural Computation*, 15(7):1667–1689, 2003.

[18] S. Krishnan, C. Bhattacharyya, and R. Hariharan. A randomized algorithm for large scale support vector learning. In *Neural Information Processing Systems (NIPS)*, 2008.

[19] B. Laurent and P. Massart. Adaptive estimation of a

quadratic functional by model selection. *The Annals of Statistics*, 2000.

[20] E. Liberty and S. W. Zucker. The mailman algorithm: A note on matrix–vector multiplication. *Information Processing Letters*, 2009.

[21] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region newton method for logistic regression. *The Journal of Machine Learning Research*, 2008.

[22] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *Core discussion papers*, 2010.

[23] M. Osadchy, D. Keren, and B. Fadida-Specktor. Hybrid Classifiers for Object Classification with a Rich Background. In *European Conference on Computer Vision–ECCV*, pages 284–297, 2012.

[24] S. Paul, C. Boutsidis, M. Magdon-Ismail, and P. Drineas. Random projections for support vector machines. *CoRR*, abs/1211.6085, 2012.

[25] J. C. Platt. In *Advances in Kernel Methods*. MIT Press, Cambridge, MA, USA, 1999.

[26] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 2007.

[27] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2):577–584, 2005.

[28] F. Schwenker. Hierarchical support vector machines for multi-class pattern recognition. In *Proc. on Knowledge-based Intelligent Engineering Systems and Aplied Technologies*, pages 561–565, 2000.

[29] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the 24th International Conference on Machine learning*, 2007.

[30] Y. Su, T. Murali, V. Pavlovic, and S. Kasif. Training support vector machines in 1D. *http://genomics10.bu.edu/yangsu/rankgene/one-svm.pdf*, 2002.

[31] A. V. Uzilov, J. M. Keegan, and D. H. Mathews. Detection of non-coding RNAs on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7(1):173, 2006.

[32] J. Weston and C. Watkins. Multi-class support vector machines. Technical report, CSD-TR-98-04, Royal Holloway University of London, 1998.

[33] Z. Yu, J.-B. Thibault, K. Sauer, C. Bouman, and J. Hsieh. Accelerated line search for coordinate descent optimization. In *Nuclear Science Symposium Conference Record*, 2006.

[34] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for L1-regularized logistic regression. *Journal of Machine Learning Research*, 2012.

[35] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE, 100(9)*, pages 2584–2603, 2012.

Appendix

Proof of Theorem 4.1:

Let $a := (a_0, ..., a_{d-1})$ with $||a|| = 1$ and let $v := (v_0, v_1, ..., v_{d-1})$ be the vector created by choosing each coordinate v_i independently from $\mathcal{N}(0,1)$. Then

$$E[a \cdot v] = E[\sum_i a_i \cdot v_i] = \sum_i a_i E[v_i] = 0$$

and

$$Var[a \cdot v] = Var[\sum_i a_i \cdot v_i] = \sum_i a_i^2 Var[v_i] = \sum_i a_i^2 = 1$$

($||a|| = 1$ so $\sum_i a_i^2 = 1$). From this, it follows $a \cdot v \sim \mathcal{N}(0,1)$.

Consider the Chi-square distribution with $k = d$ degrees of freedom: $u := \sum_{i \geq 0} v_i^2$. Using bounds from [19] (see equation (4.3)) it holds

$$P(u - d \geq 2\sqrt{dt} + 2t) \leq e^{-t}$$

for all $t > 0$. In particular for $t = 1$, we get

$$P(u - d \geq 2\sqrt{d} + 2) \leq 1/e,$$

i.e.,

$$P(u < 2\sqrt{d} + 2) > 1 - 1/e > 1/2.$$

Therefore,

$$||v|| = \sqrt{\sum_i v_i^2} < \sqrt{2\sqrt{d} + 2}$$

with probability larger than $1/2$.

For any variable x drawing from $\mathcal{N}(0,1)$ and $z > 0$:

$$\begin{aligned} P(x > z) &= \int_z^\infty 1/\sqrt{2\pi} e^{-t^2/2} dt \\ &\geq 1/\sqrt{2\pi} \cdot e^{-z^2/2} \cdot (1/z + 1/z^3) \geq e^{-z^2}. \end{aligned}$$

As $a \cdot v \sim \mathcal{N}(0,1)$, it follows $a \cdot v > z$ with probability at least e^{-z^2}.

The probability of both $a \cdot v > z$ and $||v|| < \sqrt{2\sqrt{d} + 2}$ is therefore at least $1/2 \cdot e^{-z^2}$. Then it holds that $P\left(\frac{a \cdot v}{||a|| \cdot ||v||} > z/\sqrt{2\sqrt{d} + 2}\right) > 1/2 \cdot e^{-z^2}$. Assume that $\frac{a \cdot v}{||a|| \cdot ||v||} > \frac{z}{\sqrt{2\sqrt{d}+2}}$. The angle between a and v is given by $\phi(a, v) = \cos^{-1}\left(\frac{a \cdot v}{||a|| \cdot ||v||)}\right)$ for $0 \leq \phi(a, v) \leq \pi$.

Let $\phi_0 < 1$ be a small positive value and set $\gamma = \phi_0^2/3 < \phi_0$. Using $\cos^{-1}(1 - \gamma) \leq \sqrt{3\gamma}$ and $\cos^{-1}\left(\frac{a \cdot v}{||a|| \cdot ||v||}\right) < \cos^{-1}\left(\frac{z}{\sqrt{2\sqrt{d}+2}}\right)$, it follows that $\phi(a, v) < \phi_0$ if $\frac{z}{\sqrt{2\sqrt{d}+2}} = 1 - \phi_0^2/3$, i.e., $z = (1 - \phi_0^2/3)\sqrt{2\sqrt{d} + 2}$.

This yields a total probability $P(\phi(a, v) < \phi_0) > 1/2 \cdot \exp(-((1 - \phi_0^2/3)^2(2\sqrt{d}+2)))$ when choosing a vector v randomly of $\mathcal{N}(0,1)$. Let X denote the event $\phi(a, v) < \phi_0$. Setting $n = 2 \cdot \exp((1 - \phi_0^2/3)^2(2\sqrt{d}+2)) \cdot c_0 \log(N)$ according to the Binomial distribution for a constant c_0 we expect at least $c_0 \log(N)$ vectors v such that $\phi(a, v) < \phi_0$. Using the Chernoff bound for $X \sim \mathcal{B}(n; p)$

$$P(X \leq k) \leq \exp\left(-\frac{1}{2p}\frac{(np - k)^2}{n}\right)$$

with $k = 1$, we get

$$\begin{aligned} P(X \leq 1) &\leq \exp\left(-\frac{1}{2}\frac{(c_0 \log(N) - 1)^2}{c_0 \log(N)}\right) \\ &\leq \exp(-\frac{1}{2}(c_0 \log(N) - 2)) \\ &= e \cdot N^{-c_0/2} \leq N^{-c_0/2}. \end{aligned}$$

From here, it follows that when choosing $2 \cdot \exp((1 - \phi_0^2/3)^2(2\sqrt{d}+2)) \cdot c_0 \log(N)$ vectors v randomly from $\mathcal{N}(0,1)$, the probability that at least one of them is such that $\phi(a, v) < \phi_0$ is at least $1 - \frac{1}{N^{c_0/2}}$.