SVM Optimization: Inverse Dependence on Training Set Size

Shai Shalev-Shwartz SHAI@TTI-C.ORG
Nathan Srebro NATI@UCHICAGO.EDU
Toyota Technological Institute at Chicago, 1427 East 60th Street, Chicago IL 60637, USA

Abstract

We discuss how the runtime of SVM optimization should **decrease** as the size of the training data increases. We present theoretical and empirical results demonstrating how a simple subgradient descent approach indeed displays such behavior, at least for linear kernels.

1. Introduction

The traditional runtime analysis of training Support Vector Machines (SVMs), and indeed most runtime analysis of training learning methods, shows how the training runtime *increases* as the training set size increases. This is because the analysis views SVM training as an optimization problem, whose size increases as the training size increases, and asks “what is the runtime of finding a very accurate solution to the SVM training optimization problem?”. However, this analysis ignores the underlying goal of SVM training, which is to find a classifier with low generalization error. When our goal is to obtain a good predictor, having more training data at our disposal should not increase the runtime required to get some desired generalization error: If we can get a predictor with a generalization error of 5% in an hour using a thousand examples, then given ten thousand examples we can always ignore nine thousand of them and do exactly what we did before, using the same runtime. But, can we use the extra nine thousand examples to get a predictor with a generalization error of 5% in *less* time?

In this paper we begin answering the above question. But first we analyze the runtime of various SVM optimization approaches in the data-laden regime, i.e. given unlimited amounts of data. This serves as a basis to our investigation and helps us compare different optimization approaches when working with very large data sets. A similar type of analysis for unregularized linear learning was recently presented by Bottou and Bousquet (2008)—here we handle the more practically relevant case of SVMs, although we focus on linear kernels.

We then return to the finite-data scenario and ask our original question: How does the runtime required in order to get some desired generalization error change with the amount of available data? In Section 5, we present both a theoretical analysis and a thorough empirical study demonstrating that, at least for linear kernels, the runtime of the subgradient descent optimizer PEGASOS (Shalev-Shwartz et al., 2007) does indeed decrease as more data is made available.

2. Background

We briefly introduce the SVM setting and the notation used in this paper, and survey the standard runtime analysis of several optimization approaches. The goal of SVM training is to find a linear predictor $\mathbf{w}$ that predicts the label $y \in \pm 1$ associated with a feature vector $\mathbf{x}$ as $\mathrm{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$. This is done by seeking a predictor with small empirical (hinge) loss relative to a large classification “margin”. We assume that instance-label pairs come from some source distribution $P(\mathbf{X}, Y)$, and that we are given access to labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ sampled i.i.d. from P. Training a SVM then amounts to minimizing, for some regularization parameter λ, the regularized empirical hinge loss:

$$\hat{f}_\lambda(\mathbf{w}) = \hat{\ell}(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \qquad (1)$$

where $\hat{\ell}(\mathbf{w}) = \frac{1}{m} \sum_i \ell(\mathbf{w}; (\mathbf{x}_i, y_i))$ and $\ell(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y \langle \mathbf{w}, \mathbf{x} \rangle\}$ is the hinge loss. For simplicity, we do not allow a bias term. We say that an optimization method finds an ϵ-accurate solution $\tilde{\mathbf{w}}$ if $\hat{f}_\lambda(\tilde{\mathbf{w}}) \leq \min_{\mathbf{w}} \hat{f}_\lambda(\mathbf{w}) + \epsilon$.

Instead of being provided with the feature vectors directly, we are often only provided with their inner products through a kernel function. Our focus here is on “linear kernels”, i.e. we assume we are indeed provided with the feature vectors themselves. This scenario is natural in several applications, including document analysis where the bag-of-words vectors provide a sparse high dimensional representation that does not necessarily benefit from the kernel trick. We use d to denote the dimensionality of the feature

Appearing in *Proceedings of the 25^{th} International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

vectors. Or, if the feature vectors are sparse, we use d to denote the average number of non-zero elements in each feature vector (e.g. when input vectors are bag-of-words, d is the average number of words in a document).

The runtime of SVM training is usually analyzed as the required runtime to obtain an ϵ-accurate solution to the optimization problem $\min_{\mathbf{w}} \hat{f}_\lambda(\mathbf{w})$.

Traditional optimization approaches converge linearly, or even quadratically, to the optimal solution. That is, their runtime has a logarithmic, or double logarithmic, dependence on the optimization accuracy ϵ. However, they scale poorly with the size of the training set. For example, a naïve implementation of interior point search on the dual of the SVM problem would require a runtime of $\Omega(m^3)$ per iteration, with the number of iterations also theoretically increasing with m. To avoid a cubic dependence on m, many modern SVM solvers use “decomposition techniques”: Only a subset of the dual variables is updated at each iteration (Platt, 1998; Joachims, 1998). It is possible to establish linear convergence for specific decomposition methods (e.g. Lin, 2002). However, a careful examination of this analysis reveals that the number of iterations before the linearly convergent stage can grow as m^2. In fact, Bottou and Lin (2007) argue that any method that solves the dual problem very accurately might in general require runtime $\Omega(dm^2)$, and also provide empirical evidence suggesting that modern dual-decomposition methods come close to a runtime of $\Omega(dm^2 \log(1/\epsilon))$. Therefore, for the purpose of comparison, we take the runtime of dual-decomposition methods as $O(dm^2 \log 1/\epsilon)$.

With the growing importance of handling very large data sets, optimization methods with a more moderate scaling on the data set size were presented. The flip side is that these approaches typically have much worse dependence on the optimization accuracy. A recent example is SVM-Perf (Joachims, 2006), an optimization method that uses a cutting planes approach for training linear SVMs. Smola et al. (2008) showed that SVM-Perf can find a solution with accuracy ϵ in time $O(md/(\lambda\epsilon))$.

Although SVM-Perf does have a much more favorable dependence on the data set size, and runs much faster on large data sets, its runtime still increases (linearly) with m. More recently, Shalev-Shwartz et al. (2007) presented PEGASOS, a simple stochastic subgradient optimizer for training linear SVMs, whose runtime does not at all increase with the sample size. PEGASOS is guaranteed to find, with high probability, an ϵ-accurate solution in time[1] $\tilde{O}(d/(\lambda\epsilon))$. Empirical comparisons show that PEGASOS is considerably faster than both SVM-Perf and dual decomposition methods on large data sets with sparse, linear, kernels (Shalev-Shwartz et al., 2007; Bottou, Web Page).

These runtime guarantees of SVM-Perf and PEGASOS are not comparable with those of traditional approaches: the runtimes scale better with m, but worse with ϵ, and also depend on λ. We will return to this issue in Section 4.

3. Error Decomposition

The goal of supervised learning, in the context we consider it, is to use the available training data in order to obtain a predictor with low generalization error (expected error over future predictions). However, since we cannot directly observe the generalization error of a predictor, the training error is used as a surrogate. But in order for the training error to be a good surrogate for the generalization error, we must restrict the space of allowed predictors. This can be done by restricting ourselves to a certain hypothesis class, or in the SVM formulation studied here, minimizing a combination of the training error and some regularization term.

In studying the generalization error of the predictor minimizing the training error on a limited hypothesis class, it is standard to decompose this error into:

- The **approximation error**— the minimum generalization error achievable by a predictor in the hypothesis class. The approximation error does not depend on the sample size, and is determined by the hypothesis class allowed.
- The **estimation error**—the difference between the approximation error and the error achieved by the predictor in the hypothesis class minimizing the training error. The estimation error of a predictor is a result of the training error being only an estimate of the generalization error, and so the predictor minimizing the training error being only an estimate of the predictor minimizing the generalization error. The quality of this estimation depends on the training set size and the size, or complexity, of the hypothesis class.

A similar decomposition is also possible for the somewhat more subtle case of regularized training error minimization, as in SVMs. We are now interested in the generalization error $\ell(\hat{\mathbf{w}}) = \mathbf{E}_{(\mathbf{X},Y)\sim P}\left[\ell(w;\mathbf{X},Y)\right]$ of the predictor $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \hat{f}_\lambda(\mathbf{w})$ minimizing the training objective (1). Note that for the time being we are only concerned with the (hinge) loss, and not with the misclassification error, and even measure the generalization error in terms of the hinge loss. We will return to this issue in Section 5.2.

- The approximation error is now the generalization error $\ell(\mathbf{w}^*)$ achieved by the predictor $\mathbf{w}^* = \arg\min_{\mathbf{w}} f_\lambda(\mathbf{w})$ that minimizes the *regularized* gen-

[1] The $\tilde{O}(\cdot)$ notation hides logarithmic factors.

eralization error:

$$f_\lambda(\mathbf{w}) = \ell(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 .$$

As before, the approximation error is independent of the training set or its size, and depends on the regularization parameter λ. This parameter plays a role similar to that of the complexity of the hypothesis class: Decreasing λ can decrease the approximation error.

- The estimation error is now the difference between the generalization error of $\mathbf{w}^*$ and the generalization error $\ell(\hat{\mathbf{w}})$ of the predictor minimizing the training objective $\hat{f}_\lambda(\mathbf{w})$. Again, this error is a result of the training error being only an estimate of the generalization error, and so the training objective $\hat{f}_\lambda(\mathbf{w})$ being only an estimate of the regularized loss $f_\lambda(\mathbf{w})$.

The error decompositions discussed so far are well understood, as is the trade-off between the approximation and estimation errors controlled by the complexity of the hypothesis class. In practice, however, we do not minimize the training objective exactly and so do not use the mathematically defined $\hat{\mathbf{w}}$. Rather, we use some optimization algorithm that runs for some finite time and yields a predictor $\tilde{\mathbf{w}}$ that only minimizes the training objective $\hat{f}_\lambda(\mathbf{w})$ to within some accuracy ϵ_{acc}. We should therefore consider the decomposition of the generalization error $\ell(\tilde{\mathbf{w}})$ of this predictor. In addition to the two error terms discussed above, a third error term now enters the picture:

- The **optimization error** is the difference in generalization error between the actual minimizer of the training objective and the output $\tilde{\mathbf{w}}$ of the optimization algorithm. The optimization *error* is controlled by the optimization *accuracy* ϵ_{acc}: The optimization accuracy is the difference in the training objective $\hat{f}_\lambda(\mathbf{w})$ while the optimization error is the resulting difference in generalization error $\ell(\tilde{\mathbf{w}}) - \ell(\hat{\mathbf{w}})$.

This more complete error decomposition, also depicted in Figure 1, was recently discussed by Bottou and Bousquet (2008). Since the end goal of optimizing the training error is to obtain a predictor $\tilde{\mathbf{w}}$ with low generalization error $\ell(\tilde{\mathbf{w}})$, it is useful to consider the entire error decomposition, and the interplay of its different components.

Before investigating the balance between the data set size and runtime required to obtain a desired generalization error, we first consider two extreme regimes: one in which only a limited training set is available, but computational resources are not a concern, and the other in which the training data available is virtually unlimited, but computational resources are bounded.

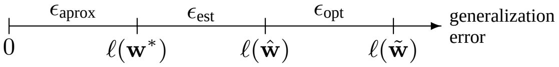

Figure 1. Decomposition of the generalization error of the output $\tilde{\mathbf{w}}$ of the optimization algorithm: $\ell(\tilde{\mathbf{w}}) = \epsilon_{\text{aprx}} + \epsilon_{\text{est}} + \epsilon_{\text{opt}}$.

Table 1. Summary of Notation

error (hinge loss)	$\ell(\mathbf{w};(\mathbf{x},y)) = \max\{0, 1-y\langle\mathbf{w},\mathbf{x}\rangle\}$
empirical error	$\hat{\ell}(\mathbf{w}) = \frac{1}{m}\sum_{(\mathbf{x},y)\in S}\ell(\mathbf{w};(\mathbf{x},y))$
generalization error	$\ell(\mathbf{w}) = \mathbf{E}\left[\ell(\mathbf{w};\mathbf{X},Y)\right]$
SVM objective	$\hat{f}_\lambda(\mathbf{w}) = \hat{\ell}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$
Expected SVM obj.	$f_\lambda(\mathbf{w}) = \ell(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$
Reference predictor	$\mathbf{w}_0$
Population optimum	$\mathbf{w}^* = \arg\min_{\mathbf{w}} f_\lambda(\mathbf{w})$
Empirical optimum	$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \hat{f}_\lambda(\mathbf{w})$
ϵ_{acc}-optimal predictor	$\tilde{\mathbf{w}}$ s.t. $\hat{f}_\lambda(\tilde{\mathbf{w}}) \leq \hat{f}_\lambda(\hat{\mathbf{w}}) + \epsilon_{\text{acc}}$

3.1. The Data-Bounded Regime

The standard analysis of statistical learning theory can be viewed as an analysis of an extreme regime in which training data is scarce, and computational resources are plentiful. In this regime, the optimization error diminishes, as we can spend the time required to optimize the training objective very accurately. We need only consider the approximation and estimation errors. Such an analysis provides an understanding of the sample complexity as a function of the target error: how many samples are necessary to guarantee some desired error level.

For low-norm (large-margin) linear predictors, the estimation error can be bounded by $O\left(\frac{\|\mathbf{w}^*\|}{\sqrt{m}}\right)$ (Bartlett & Mendelson, 2003), yielding a sample complexity of $m = O\left(\frac{\|\mathbf{w}^*\|^2}{\epsilon^2}\right)$ to get a desired generalization error of $\ell(\mathbf{w}^*) + \epsilon$ (tighter bounds are possible under certain conditions, but for simplicity and more general applicability, here we stick with this simpler analysis).

3.2. The Data-Laden Regime

Another extreme regime is the regime in which we have virtually unlimited data (we can obtain samples on-demand), but computational resources are limited. This is captured by the PAC framework (Valiant, 1984), in which we are given unlimited, on-demand, access to samples, and consider computationally tractable methods for obtaining a predictor with low generalization error. Most work in the PAC framework focuses on the distinction between polynomial and super-polynomial computation. Here, we are interested in understating the details of this polynomial dependence—how does the runtime scale with the parameters of interest? Discussing runtime as a function of data set size is inappropriate here, since the data set size is unlimited. Rather, we are interested in understanding the runtime as a function of the target error: How much runtime is required to guarantee some desired error level.

As the data-laden regime does capture many large data set situations, in which data is virtually unlimited, such an analysis can be helpful in comparing different optimization approaches. We saw how traditional runtime guarantees of different approaches are sometimes seemingly incomparable: One guarantee might scale poorly with the sample size, while another scales poorly with the desired optimization accuracy. The analysis we perform here allows us to compare such guarantees and helps us understand which methods are appropriate for large data sets.

Recently, Bottou and Bousquet (2008) carried out such a "data-laden" analysis for unregularized learning of linear separators in low dimensions. Here, we perform a similar type of analysis for SVMs, i.e. regularized learning of a linear separator in high dimensions.

4. Data-Laden Analysis of SVM Solvers

To gain insight into SVM learning in the data-laden regime we perform the following "oracle" analysis: We assume there is some good low-norm predictor $\mathbf{w}_0$, which achieves a generalization error (expected hinge loss) of $\ell(\mathbf{w}_0)$ and has norm $\|\mathbf{w}_0\|$. We train a SVM by minimizing the training objective $\hat{f}_\lambda(\mathbf{w})$ to within optimization accuracy ϵ_{acc}. Since we have access to an unrestricted amount of data, we can choose what data set size to work with in order to achieve the lowest possible runtime.

We will decompose the generalization error of the output predictor $\tilde{\mathbf{w}}$ as follows:

$$\begin{aligned}\ell(\tilde{\mathbf{w}}) &= \ell(\mathbf{w}_0) \\ &\quad + (f_\lambda(\tilde{\mathbf{w}}) - f_\lambda(\mathbf{w}^*)) \\ &\quad + (f_\lambda(\mathbf{w}^*) - f_\lambda(\mathbf{w}_0)) \\ &\quad + \frac{\lambda}{2}\|\mathbf{w}_0\|^2 - \frac{\lambda}{2}\|\tilde{\mathbf{w}}\|^2 \end{aligned} \tag{2}$$

The degradation in the regularized generalization error, $f_\lambda(\tilde{\mathbf{w}}) - f_\lambda(\mathbf{w}^*)$, which appears in the second term, can be bounded by the empirical degradation: For all $\mathbf{w}$ with $\|\mathbf{w}\|^2 \leq 2/\lambda$ (a larger norm would yield a worse SVM objective than $\mathbf{w} = 0$, and so can be disqualified), with probability at least $1-\delta$ over the training set (Sridharan, 2008):

$$f_\lambda(\mathbf{w}) - f_\lambda(\mathbf{w}^*) \leq 2\left[\hat{f}_\lambda(\mathbf{w}) - \hat{f}_\lambda(\mathbf{w}^*)\right]_+ + O\left(\frac{\log\frac{1}{\delta}}{\lambda m}\right)$$

where $[z]_+ = max(z, 0)$. Recalling that $\tilde{\mathbf{w}}$ is an ϵ_{acc}-accurate minimizer of $\hat{f}_\lambda(\mathbf{w})$, we have:

$$f_\lambda(\tilde{\mathbf{w}}) - f_\lambda(\mathbf{w}^*) \leq 2\epsilon_{\text{acc}} + O\left(\frac{\log\frac{1}{\delta}}{\lambda m}\right) \tag{3}$$

Returning to the decomposition (2), the third term is non-positive due to the optimality of $\mathbf{w}^*$, and regarding δ as a constant we obtain that with arbitrary fixed probability:

$$\ell(\tilde{\mathbf{w}}) \leq \ell(\mathbf{w}_0) + 2\epsilon_{\text{acc}} + \frac{\lambda}{2}\|\mathbf{w}_0\|^2 + O\left(\frac{1}{\lambda m}\right) \tag{4}$$

In order to obtain an upper bound of $\ell(\mathbf{w}_0) + O(\epsilon)$ on the generalization error $\ell(\tilde{\mathbf{w}})$, each of the three remaining terms on the right hand side of (4) must be bounded from above by $O(\epsilon)$, yielding:

$$\epsilon_{\text{acc}} \leq O(\epsilon) \tag{5}$$

$$\lambda \leq O\left(\frac{\epsilon}{\|\mathbf{w}_0\|^2}\right) \tag{6}$$

$$m \geq \Omega\left(\frac{1}{\lambda\epsilon}\right) \geq \Omega\left(\frac{\|\mathbf{w}_0\|^2}{\epsilon^2}\right) \tag{7}$$

Using the above requirements on the optimization accuracy ϵ_{acc}, the regularization parameter λ and the working sample size m, we can revisit the runtime of the various SVM optimization approaches.

As discussed in Section 2, dual decomposition approaches require runtime $\Omega(m^2 d)$, with a very weak dependence on the optimization accuracy. Substituting in the sample size required for obtaining the target generalization error of $\ell(\mathbf{w}_0) + \epsilon$, we get a runtime of $\Omega\left(\frac{d\|\mathbf{w}_0\|^4}{\epsilon^4}\right)$.

We can perform a similar analysis for SVM-Perf by substituting the requirements on ϵ_{acc}, λ and m into its guaranteed runtime of $O\left(\frac{dm}{\lambda\epsilon_{\text{acc}}}\right)$. We obtain a runtime of $O\left(\frac{d\|\mathbf{w}_0\|^4}{\epsilon^4}\right)$, matching that in the analysis of dual decomposition methods above. It should be noted that SVM-Perf's runtime has been reported to have only a logarithmic dependence on $1/\epsilon_{\text{acc}}$ in practice (Smola et al., 2008). If that were the case, the runtime guarantee would drop to $\tilde{O}\left(\frac{d\|\mathbf{w}_0\|^4}{\epsilon^3}\right)$, perhaps explaining the faster runtime of SVM-Perf on large data sets in practice.

As for the stochastic gradient optimizer PEGASOS, substituting in the requirements on ϵ_{acc} and λ into its $\tilde{O}(d/(\lambda\epsilon_{\text{acc}}))$ runtime guarantee yields a data-laden runtime of $\tilde{O}\left(\frac{d\|\mathbf{w}_0\|^2}{\epsilon^2}\right)$. We see, then, that in the data-laden regime, where we can choose a data set of arbitrary size in order to obtain some target generalization error, the runtime guarantee of PEGASOS dominates those of other methods, including those with a much more favorable dependence on the optimization accuracy.

The traditional and data-laden runtimes, ignoring logarithmic factors, are summarized in the following table:

Method	ϵ_{acc}-accurate	$\ell(\tilde{\mathbf{w}}) \leq \ell(\mathbf{w}_0) + \epsilon$
Dual decompositoin	dm^2	$\frac{d\|\mathbf{w}_0\|^4}{\epsilon^4}$
SVM-Perf	$\frac{dm}{\lambda\epsilon_{\text{acc}}}$	$\frac{d\|\mathbf{w}_0\|^4}{\epsilon^4}$
PEGASOS	$\frac{d}{\lambda\epsilon_{\text{acc}}}$	$\frac{d\|\mathbf{w}_0\|^2}{\epsilon^2}$

5. The Intermediate Regime

We have so far considered two extreme regimes: one in which learning is bounded only by available data, but not by computational resources, and another where it is bounded only by computational resources, but unlimited data is available. These two analyzes tell us how many samples are needed in order to guarantee some target error rate (regardless of computational resources), and how much computation is needed to guarantee this target error rate (regardless of available data). However, if we have just enough samples to allow a certain error guarantee, the runtime needed in order to obtain such an error rate might be much higher than the runtime given unlimited samples. In terms of the error decomposition, the approximation and estimation errors together would already account for the target error rate, requiring the optimization error to be extremely small. Only when more and more samples are available might the required runtime decrease down to that obtained in the data-laden regime.

Accordingly, we study the runtime of a training method as a decreasing function of the available training set size. As argued earlier, studied this way, the required runtime should never increase as more data is available. We would like to understand how the excess data can be used to decrease the runtime.

In many optimization methods, including dual decomposition methods and SVM-Perf discussed earlier, the computational cost of each basic step increases, sometimes sharply, with the size of the data set considered. In such algorithms, increasing the working data set size in the hope of being able to optimize to within a lower optimization accuracy is a double-edged sword. Although we can reduce the required optimization accuracy, and doing so reduces the required runtime, we also increase the computational cost of each basic step, which sharply increases the runtime.

However, in the case of a stochastic gradient descent approach, the runtime to get some desired optimization accuracy does not increase as the sample size increases. In this case, increasing the sample size is a pure win: The desired optimization accuracy decreases, with no counter effect, yielding a net decrease in the runtime.

In the following sections, we present a detailed theoretical analysis based on performance guarantees, as well as an empirical investigation, demonstrating a decrease in PEGASOS runtime as more data is available.

5.1. Theoretical Analysis

Returning to the "oracle" analysis of Section 4 and substituting into equation (4) our bound on the optimization accuracy of PEGASOS after running for time T, we obtain:

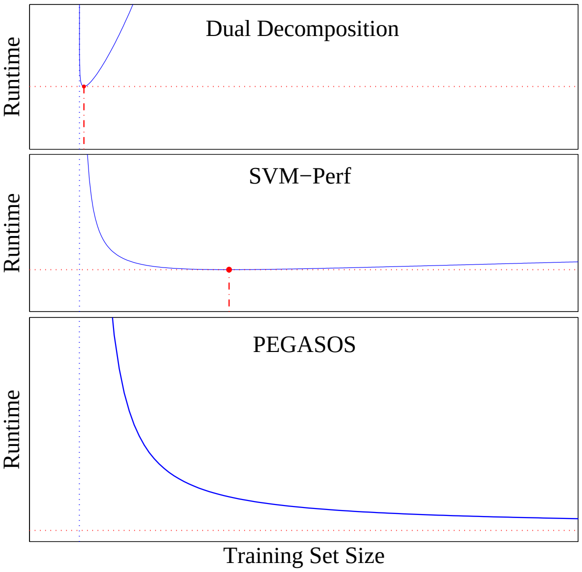

Figure 2. Descriptive behavior of the runtime needed to achieve some fixed error guarantee based on upper bounds for different optimization approaches (solid curves). The dotted lines are the sample-size requirement in the data-bounded regime (vertical) and the runtime requirement in the data-laden regime (horizontal). In the top two panels (dual decomposition and SVM-Perf), the minimum runtime is achieved for some finite training set size, indicated by a dash-dotted line.

$$\ell(\tilde{\mathbf{w}}) \leq \ell(\mathbf{w}_0) + \tilde{O}(\frac{d}{\lambda T}) + \frac{\lambda}{2}\|\mathbf{w}_0\|^2 + O(\frac{\|\mathbf{w}_0\|}{\sqrt{m}}) \quad (8)$$

The above bound is minimized when $\lambda = \tilde{\Theta}(\sqrt{\frac{d}{\|\mathbf{w}_0\|^2 T}})$, yielding $\ell(\tilde{\mathbf{w}}) \leq \ell(\mathbf{w}_0) + \epsilon(T,m)$ with

$$\epsilon(T,m) = \tilde{O}\left(\|\mathbf{w}_0\|\sqrt{\tfrac{d}{T}}\right) + O\left(\tfrac{\|\mathbf{w}_0\|}{\sqrt{m}}\right). \quad (9)$$

Inverting the above expression, we get the following bound on the runtime required to attain generalization error $\ell(\tilde{\mathbf{w}}) \leq \ell(\mathbf{w}_0) + \epsilon$ using a training set of size m:

$$T(m;\epsilon) = \tilde{O}\left(\frac{d}{\left(\frac{\epsilon}{\|\mathbf{w}_0\|} - O(\frac{1}{\sqrt{m}})\right)^2}\right). \quad (10)$$

This runtime analysis, which monotonically decreases with the available data set size, is depicted in the bottom panel of Figure 2. The data-bounded (statistical learning theory) analysis describes the vertical asymptote of $T(\cdot;\epsilon)$—at what sample size is it at all possible to achieve the desired error. The analysis of the data-laden regime of Section 4 described the minimal runtime using any amount of data, and thus specifies the horizontal asymptote $\inf T(m;\epsilon) =$

$\lim_{m\to\infty} T(m;\epsilon)$. The more detailed analysis carried out here bridges between these two extreme regimes.

Before moving on to empirically observing this behavior, let us contrast this behavior with that displayed by learning methods whose runtime required for obtaining a fixed optimization accuracy does increase with data set size. We can repeat the analysis above, replacing the first term on the right hand side of (8) with the guarantee on the optimization accuracy at runtime of T, for different algorithms.

For SVM-Perf, we have $\epsilon_{\text{acc}} \leq O\left(dm/(\lambda T)\right)$. The optimal choice of λ is then $\lambda = \Theta\left(\sqrt{\frac{dm}{T\|\mathbf{w}_0\|^2}}\right)$ and the runtime needed to guarantee generalization error $\ell(\mathbf{w}_0) + \epsilon$ when running SVM-Perf on m samples is $T(m;\epsilon) = O\left(dm \Big/ \left(\frac{\epsilon}{\|\mathbf{w}_0\|} - O(\frac{1}{\sqrt{m}})\right)^2\right)$. The behavior of this guarantee is depicted in the middle panel of Figure 2. As the sample size increases beyond the statistical limit $m_0 = \Theta(\|\mathbf{w}_0\|^2/\epsilon^2)$, the runtime indeed decreases sharply, until it reaches a minimum, corresponding to the data laden bound, precisely at $4m_0$, i.e. when the sample size is four times larger than the minimum required to be able to reach the desired target generalization error. Beyond this point, the other edge of the sword comes into play, and the runtime (according to the performance guarantees) increases as more samples are included.

The behavior of a dual decomposition method with runtime $\Theta(m^2 d \log \frac{1}{\epsilon_{\text{acc}}})$ is given by $T(m;\epsilon) = m^2 d \log(1/(\epsilon - \Theta\frac{\|\mathbf{w}_0\|}{\sqrt{m}}))$ and depicted in the top panel of Figure 2. Here, the optimal sample size is extremely close to the statistical limit, and increasing the sample size beyond the minimum increases the runtime quadratically.

5.2. Empirical Analysis

The above analysis is based on upper bounds, and is only descriptive, in that it ignores various constants and even certain logarithmic factors. We now show that this type of behavior can be observed empirically for the stochastic subgradient optimizer PEGASOS.

We trained PEGASOS[2] on training sets of increasing size taken from two large data sets, the Reuters CCAT and the CoverType datasets[3]. We measured the average hinge loss of the learned predictor on a (fixed) held-out test set. For each training set size, we found the median number of iterations (over multiple runs with multiple training sets) for achieving some target average hinge loss, which was very slightly above the best "test" hinge loss that could be reliably obtained by training on the entire available training set. For each training set size we used the optimal λ for achieving the desired target hinge loss[4]. The (median) required number of iterations is displayed in Figure 3. For easier interpretability and reproducibility, we report the number of iterations. Since each PEGASOS iteration takes constant time, the actual runtime is proportional to the number of iterations.

So far we have measured the generalization error only in terms of the average hinge loss $\ell(\tilde{\mathbf{w}})$. However, our true goal is usually to attain low misclassification error, $P(Y \neq \operatorname{sign}\langle\tilde{\mathbf{w}}, \mathbf{X}\rangle)$. The dashed lines in Figure 3 indicate the (median) number of iterations required to achieve a target misclassification error, which again is very slightly above the best that can be hoped for with the entire data set.

These empirical results demonstrate that the runtime of SVM training using PEGASOS indeed *decreases* as the size of the training set increases. It is important to note that PEGASOS is the fastest published method for these datasets (Shalev-Shwartz et al., 2007; Bottou, Web Page), and so we are indeed investigating the best possible runtimes. To gain an appreciation of this, as well as to observe the runtime dependence on the training set size for other methods, we repeated a limited version of the experiments using SVM-Perf and the dual decomposition method SVM-Light (Joachims, 1998). Figure 4 and its caption report the runtimes required by SVM-Perf and SVM-Light to achieve the same fixed misclassification error using varying data set sizes. We can indeed verify that PEGASOS's

[2] We used a variant of the method described by Shalev-Shwartz et al. (2007), with a single example used in each update: Following Bottou (Web Page), instead of sampling an example independently at each iteration, a random permutation over the training set is used. When the permutation is exhausted, a new, independent, random permutation is drawn. Although this variation does not match the theoretical analysis, it performs slightly better in practice. Additionally, the PEGASOS projection step is skipped, as it can be shown that even without it, $\|\mathbf{w}\|^2 \leq 4/\lambda$ is maintained.

[3] The binary text classification task CCAT from the Reuters RCV1 collection and Class 1 in the CoverType dataset of Blackard, Jock & Dean. CCAT consists of 804,414 examples with 47,236 features of which 0.16% are non-zero. CoverType has 581,012 examples with 54 features of which 22% are non-zero. We used 23,149 CCAT examples and 58,101 CoverType examples as test sets and sampled training sets from the remainder.

[4] Selecting λ based on results on the test set seems like cheating, and is indeed slightly cheating. However, the same λ was chosen for multiple random training sets of the same size, and represents the optimal λ for the *learning problem*, not for a specific training set (i.e. we are not gaining here from random fluctuations in learning). The setup in which the optimal λ is "known" is common in evaluation of SVM runtime. Choosing λ by proper validation involves many implementation choices that affect runtime, such as the size of the holdout and/or number of rounds of cross-validation, the range of λs considered, and the search strategy over λs. We therefore preferred a "known λ" setup, where we could obtain results that are cleaner, more interpretable, and less affected by implementation details. The behavior displayed by our results is still indicative of a realistic situation where λ must be selected.

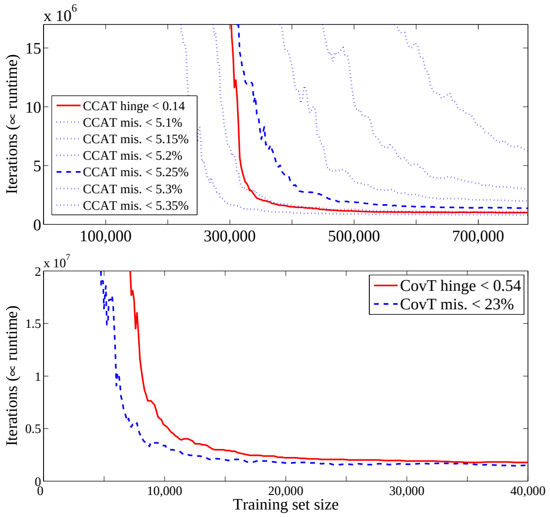

Figure 3. Number of PEGASOS iterations required to achieve the desired hinge loss (solid lines) or misclassification error (dashed and dotted lines) on the test set. Top: CCAT. The minimum achievable hinge loss and misclassification error are 0.132 and 5.05%. Bottom: CoverType. The minimum achievable hinge loss and misclassification error are 0.536 and 22.3%.

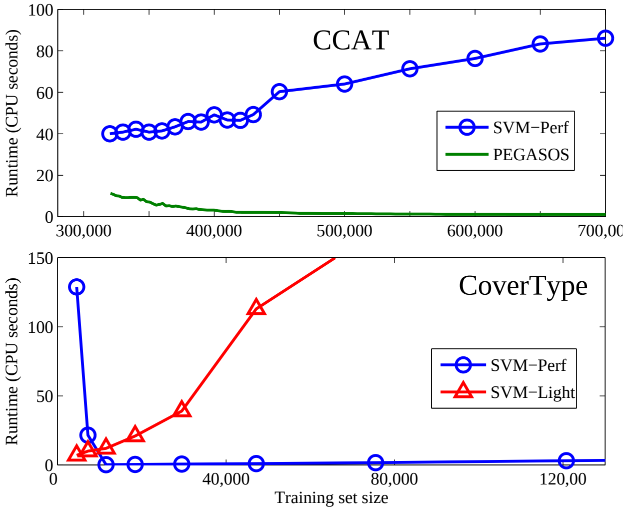

Figure 4. Runtime required to achieve average misclassification error of 5.25% on CCAT (top) and 23% on CoverType (bottom) on a 2.4 GHz Intel Core2, using optimal λ settings. SVM-Light runtimes for CCAT increased from 1371 seconds using 330k examples to 4.4 hours using 700k examples. SVM-Light runtimes for CoverType increased to 552 seconds using 120k examples.

runtime is significantly lower than the optimal SVM-Perf and SVM-Light runtimes on the CCAT dataset. On the CoverType data set, PEGASOS and SVM-Perf have similar optimal runtimes (both optimal runtimes were under a second, and depending on the machine used, each method was up to 50% faster or slower than the other), while SVM-Light's runtime is significantly higher (about 7 seconds). We also clearly see the increase in runtime for large training set sizes for both SVM-Light and SVM-Perf. On the CoverType dataset, we were able to experimentally observe the initial decrease in SVM-Perf runtime, when we are just past the statistical limit, and up to some optimal training set size. On CCAT, and on both data sets for SVM-Light, the optimal data set size is the minimal size statistically required and any increase in data set size increases runtime (since the theoretical analysis is just an upper bound, it is possible that there is no initial decrease, or that it is very narrow and hard to detect experimentally).

In order to gain a better understanding of the reduction in PEGASOS's runtime, we show in Figure 5 the average (over multiple training sets) generalization error achieved by PEGASOS over time, for various data set sizes. It should not be surprising that the generalization error decreases with the number of iterations, nor that it is lower for larger data sets. The important observation is that for smaller data sets the error decreases more slowly, even before the statistical limit for that data set is reached, as opposed to the hypothetical behavior depicted in the insert of Figure 5. This can also be seen in the dotted plots of Figure 3, which are essentially contour lines of the generalization error as a function of runtime and training set size—the error decreases when *either* runtime or training set size increase. And so, fixing the error, we can trade off between the runtime and data set size, decreasing one of them when the other is increased.

The hypothetical situation depicted in the insert occurs when runtime and dataset size each limit the attainable error independently. This corresponds to “L”-shaped contours: both a minimum runtime and a minimum dataset size are required to attain each error level, and once both requirements are met, the error is attainable. In such a situation, the runtime does *not* decrease as data set size increases, but rather, as in the “L”-shaped graph, remains constant once the statistical limit is passed. This happens, e.g., if the optimization can be carried out with a single pass over the data (or at least, if one pass is enough for getting very close to $\ell(\hat{\mathbf{w}})$). Although behavior such as this has been reported using *second-order* stochastic gradient de-

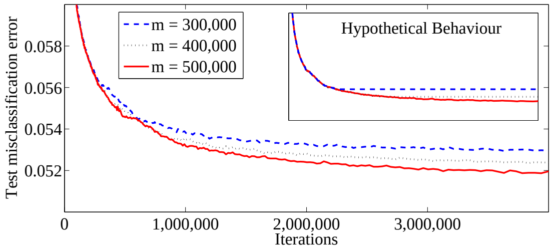

Figure 5. Average misclassification error achieved by PEGASOS on the CCAT test set as a function of runtime (#iterations), for various training set sizes. The insert is a cartoon depicting a hypothetical situation discussed in the text.

scent for *unregularized* linear learning (Bottou & LeCun, 2004), this is not the case here. Unfortunately we are not aware of an efficient one-pass optimizer for SVMs.

6. Discussion

We suggest here a new way of studying and understanding the runtime of training: Instead of viewing additional training data as a computational burden, we view it as an asset that can be used to our benefit. We already have a fairly good understanding, backed by substantial theory, on how additional training data can be used to lower the generalization error of a learned predictor. Here, we consider the situation in which we are satisfied with the error, and study how additional data can be used to decrease training runtime. To do so, we study runtime as an explicit function of the acceptable predictive performance.

Specifically, we show that a state-of-the-art stochastic gradient descent optimizer, PEGASOS, indeed requires training runtime that monotonically decreases as a function of the sample size. We show this both theoretically, by analyzing the behavior of upper bounds on the runtime, and empirically on two standard datasets where PEGASOS is the fastest known SVM optimizer. To the best of our knowledge, this is the first demonstration of a SVM optimizer that displays this natural behavior.

The reason PEGASOS's runtime decreases with increased data is that its runtime to get a fixed optimization accuracy does not depend on the training set size. This enables us to leverage a decreased estimation error, without paying a computational penalty for working with more data.

The theoretical analysis presented in Section 5.1, and we believe also the empirical reduction in PEGASOS's runtime, indeed relies on this decrease in estimation error. This decrease is significant close to the statistical limit on the sample size, as is evident in the results of Figure 3—a roughly 10–20% increase in sample size reduces the runtime by about a factor of five. However, the decrease diminishes for larger sample sizes. This can also be seen from the theoretical analysis—having a sample size which is greater than the statistical limit by a constant factor enables us to achieve a runtime which is greater than the theoretical (data-laden) limit by a constant factor (in fact, as the careful reader probably noticed, since our data-laden theoretical analysis ignores constant factors on ϵ and m, it seems that the training set size needed to be within the data-laden regime, as specified in equation (7), is the same as the minimum data set size required statistically). Such "constant factor" effects should not be discounted—having four times as much data (as is roughly the factor for CoverType) is often quite desirable, as is reducing the runtime by a factor of ten (as this four-fold increase achieves).

We are looking forward to seeing methods that more explicitly leverage large data sets in order to reduce runtime, achieving stronger decreases in practice, and being able to better leverage very large data sets. Although it seems that not much better can be done theoretically given only the simple oracle assumption of Section 4, a better theoretical analysis of such methods might be possible using richer assumptions. We would also like to see practical methods for non-linear (kernelized) SVMs that display similar behavior. Beyond SVMs, we believe that many other problems in machine learning, usually studied computationally as optimization problems, can and should be studied using the type of analysis presented here.

References

Bartlett, P. L., & Mendelson, S. (2003). Rademacher and gaussian complexities: risk bounds and structural results. *J. Mach. Learn. Res.*, *3*, 463–482.

Bottou, L. (Web Page). Stochastic gradient descent examples. `http://leon.bottou.org/projects/sgd`.

Bottou, L., & Bousquet, O. (2008). The tradeoffs of large scale learning. *Advances in Neural Information Processing Systems 20*.

Bottou, L., & LeCun, Y. (2004). Large scale online learning. *Advances in Neural Information Processing Systems 16*.

Bottou, L., & Lin, C.-J. (2007). Support vector machine solvers. In L. Bottou, O. Chapelle, D. DeCoste and J. Weston (Eds.), *Large scale kernel machines*. MIT Press.

Joachims, T. (1998). Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods—Support Vector learning*. MIT Press.

Joachims, T. (2006). Training linear svms in linear time. *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*.

Lin, C.-J. (2002). A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, *13*, 1045–1052.

Platt, J. C. (1998). Fast training of Support Vector Machines using sequential minimal optimization. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods–Support Vector learning*. MIT Press.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. *Proceedings of the 24th International Conference on Machine Learning*.

Smola, A., Vishwanathan, S., & Le, Q. (2008). Bundle methods for machine learning. *Advances in Neural Information Processing Systems 20*.

Sridharan, K. (2008). Fast convergence rates for excess regularized risk with application to SVM. `http://ttic.uchicago.edu/~karthik/con.pdf`.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.