UNIVERSITY OF CALIFORNIA

SANTA CRUZ

# Bad Optimizations Make Good Learning

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

**Ziqi Chen**

March 2012

# Table of Contents

# List of Figures

# List of Tables

**Bad Optimizations Make Good Learning**

*Ziqi Chen*

## ABSTRACT

This thesis reports on experiments aimed at explaining why machine learning algorithms using the greedy stochastic gradient descent (SGD) algorithm sometimes generalize better than algorithms using other optimization techniques. We propose two hypothesis, namely the "canyon effect" and the "classification insensitivity", and illustrate them with two data sources. On these data sources, SGD generalizes more accurately than SVM$^{perf}$, which performs more intensive optimization, over a wide variety of choices of the regularization parameters. Finally, we report on some similar, but predictably less dramatic, effects on natural data.

**Acknowledgments**

The text of this thesis extends the following previously submitted material to *International Conference on Machine Learning 2012, Edinburgh, Scotland.* This material is coauthored by Mr. Ziqi Chen, Professor David P. Helmbold and Dr. Philip M. Long. Professor David P. Helmbold and Dr. Philip M. Long directed and supervised the research in this publication which forms the basis for the thesis. Thanks to David P. Helmbold and Philip M. Long for focus, encouragement, patience, constructive comments, helpful criticism, and great ideas. Professor Manfred K. Warmuth helped out insights in an early material for optimizations for boosting and thesis refinement. The original title of the earlier submitted material was inspired by a comment along similar lines during a talk by Kristin Bennett. I also want to thank my family supporting my work and study at University of California, Santa Cruz.

## 1 Introduction

Many machine learning algorithms work by solving an optimization problem [16, 8, 7, 17, 1, 15]. Typically, the function to be optimized has two parts, a loss function that penalizes training error, and a regularizer that penalizes complexity. One very popular concrete example is the SVM objective function, which, for examples $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m) \in \mathbb{R}^n \times \{-1, 1\}$, is

$$\phi(\mathbf{w}) = ||\mathbf{w}||^2 + C \sum_{t=1}^{m} \ell(y_t(\mathbf{w} \cdot \mathbf{x}_t)),$$

where $\ell$ is the hinge loss, defined by $\ell(z) = \max\{1 - z, 0\}$. Note that the smaller $C$ is, the more important $||w||$ is to the objective. Therefore smaller $C$ has the effect of greater regularization.

It is generally acknowledged that, past a certain point, further optimization of the objective function is often not rewarded with improvement in generalization accuracy (see [1, 13]). In fact, many have found that very simple and greedy optimization techniques, such as Stochastic Gradient Descent (SGD), provide excellent generalization (see [6, 5, 2]). Early stopping of an SGD algorithm has been viewed as an alternative to a penalty function as a means of regularization. However, SGD generalizes relatively well even when compared with regularized methods [5, 14] (see also Figure 8 of this thesis).

In this thesis we hypothesize that two effects help simple SGD achieve better generalization than methods that more effectively minimize the objective function. We present experiments on data from mixtures of Gaussians supporting these hypotheses.

We call the first effect the *canyon effect* (see Figure 1). This occurs when two examples, $(\mathbf{x}_s, y_s)$ and $(\mathbf{x}_t, y_t)$, are nearly contradictory – often this is because both examples are part of a cluster of related cases that is corrupted by label noise, so that $\mathbf{x}_s \approx \mathbf{x}_t$ but $y_s = -y_t$. If example $s$ is encountered first, and is classified incorrectly, a stochastic gradient algorithm will update $\mathbf{w}$ in the direction of $y_s\mathbf{x}_s$. This may cause it to misclassify $y_t\mathbf{x}_t$, leading to an update back in the direction $y_t\mathbf{x}_t$. Since

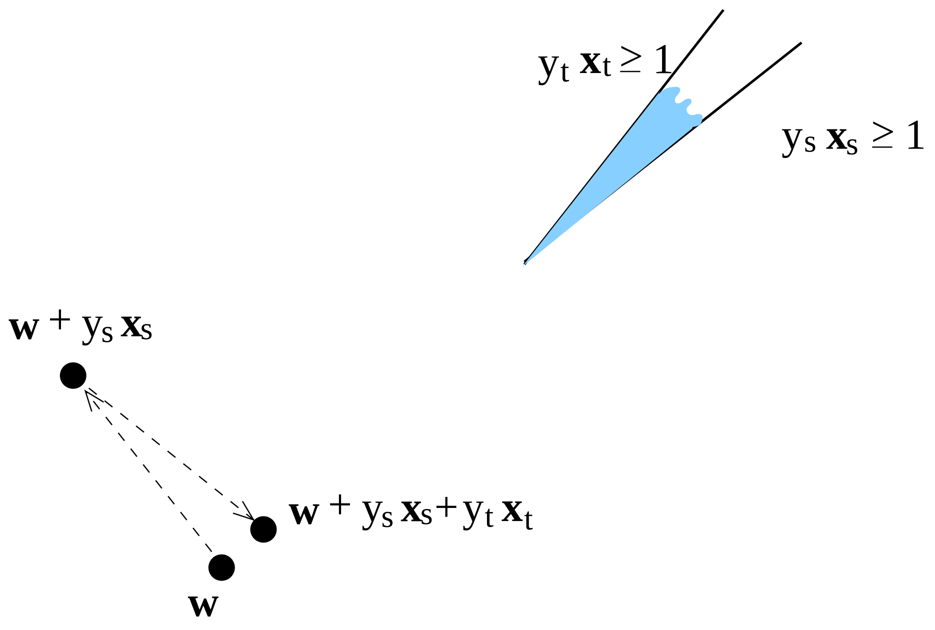

$y_t \mathbf{x}_t \geq 1$
$y_s \mathbf{x}_s \geq 1$
$\mathbf{w} + y_s \mathbf{x}_s$
$\mathbf{w} + y_s \mathbf{x}_s + y_t \mathbf{x}_t$
$\mathbf{w}$

Figure 1: The projection of weight space onto the span of two nearly contradictory examples $(\mathbf{x}_s, y_s)$ and $(\mathbf{x}_t, y_t)$. The shaded region corresponds to those weights that achieve zero hinge loss on these two examples. Starting from some weight vector $\mathbf{w}$, and performing stochastic gradient descent on the hinge loss for one example, followed by the other, results in nearly canceling updates. More efficient optimization more rapidly adds length of $\mathbf{w}$ in the direction $y_s\mathbf{x}_s + y_t\mathbf{x}_t$.

$y_s\mathbf{x}_s \approx -y_t\mathbf{x}_t$, the effect of these two updates nearly cancel. If one of the examples was corrupted with label noise, then $y_s\mathbf{x}_s + y_t\mathbf{x}_t$, the difference between $\mathbf{x}_s$ and $\mathbf{x}_t$, is likely to be an irrelevant direction. If an algorithm that more effectively optimizes the SVM objective function is applied, its hypothesis could include a significant amount of its weight vector in the $y_s\mathbf{x}_s + y_t\mathbf{x}_t$ direction in order to "handle" both of these examples. Even when the algorithm has a "budget" on (or pays for) the length of its weight vector, it still may find it worthwhile to expend a significant amount of its budget on this irrelevant direction. If SGD is run for *many* epochs, it is also likely to put a lot of weight in this direction. However, if SGD is stopped before running *too* long, the damage can be limited. Thus, SGD run for few epochs will be relatively immune to this ping-pong effect compared to algorithms that more accurately optimize the objective.

The canyon effect is strongest when $C$ is relatively large, since large $C$ places a premium on classifying the training examples correctly. When $C$ is small, the canyon effect is greatly reduced, but SGD can still outperform more accurate optimizers.

When $C$ is *very* small, the optimal solution will have very small weights. If all examples have margin $y_s(\mathbf{w}^* \cdot \mathbf{x}_s) < 1$ then, near the optimum $\mathbf{w}^*$, the rewards for increasing the margin of examples that already have a relatively large margin are the same as those for improving the margin of examples that are currently misclassified. A global optimizer may respond to these pressures by achieving a relatively large margin on some examples at the expense of small negative margins on others. Since it is easier to get large margin on large instances, this situation usually occurs when the examples have different lengths. We call this second effect the *classification-insensitivity effect.* SGD algorithms update their weight vector every time they see a misclassified example, so they do not ignore these smaller magnitude misclassified examples. Since SGD confers protection against the canyon effect, using SGD reduces the need to use a small value of $C$, indirectly protecting against the classification-insensitivity effect.

In this thesis, we perform experiments with data sources that give rise to the canyon effect and the classification-insensitivity effect. We compare the generalization error of an SGD algorithm against hypotheses produced by SVM$^{perf}$, which does a good job of optimizing $\phi$. We designed our synthetic sources to have some of the characteristics of web spam. Web spammers make money by placing advertisements on low-quality pages that match keywords. These spammy pages are are often generated automatically by a script, and are often closely related in many respects. Training examples for identifying web spam may be obtained using human raters. However, web spam does not have a precise, objective definition, and human raters may vary in the effort that they expend to look for signs of spam. (for example they may perform searches for scraped content only in some cases, or some may read more or less of a page to see whether the language "feels natural"). Consequently, training data for web spam will tend to have a lot of label noise. Our synthetic sources are mixtures of spherically symmetrical Gaussians. Each Gaussian is meant to roughly correspond to a spam campaign. The variance of the Gaussians is small compared with their separation. Adding label noise gives rise to the canyon effect. The classification-insensitivity effect can be created by placing the Gaussians so that those far from the origin are best separated in one direction, but all of the Gaussians can be separated using a different direction.

For this source and a moderately small number of training examples, five rounds of SGD with $C$ fixed at $10^5$ generalizes significantly better than using SVM$^{perf}$with a wide range of values of $C$. The fact that it improves on the generalization of SVM$^{perf}$for small $C$ shows that this improvement is not just due to the fact that SGD promotes short weight vectors. Specific aspects of the way that SGD performs its optimization must be important.

The importance of this effect depends on how often something like it is seen with natural data. We also give an example of similar behavior using the natural RCV1-V2 dataset [11, 3]. As expected, the effect is smaller – after all, we constructed the artificial datasets to bring out these effects – but a similar comparative performance is seen on the RCV1-V2 data. Furthermore, SGD with its default $\lambda$ (corresponding to $C = 10^5$) results in better generalization than SVM$^{perf}$using a wide range of $C$-values after modest numbers of training examples.

## 2 Related Work

Earlier papers have explained the relative strength of SGD by comparing upper bounds on accuracy obtained through SGD and second-order algorithms [5, 4, 14]. The idea is that adding new examples using a stochastic gradient algorithm yields generalization dividends at a faster rate than more intensive processing of old examples. Thus one pass using stochastic gradient produces good hypotheses more quickly. The classification-insensitivity effect was exploited in a theoretical analysis of boosting by [12].

It is generally known that both SGD and averaged SGD (ASGD) will converge to the theoretical value of the objective function.[18]. The accuracies reach the peak and then will eventually decay. Experiment result in this thesis verify this phenomenon.

## 3 Method and Experiments

All the experiments conducted in this thesis used version 2.0 of the implementation of SGD by [3]. For a more intensive optimization method, we used version 3.0 of SVM$^{perf}$[10]. SVM$^{perf}$is a descendant of SVM$^{light}$[9] that is optimized for the linear kernel.

For running time comparisons of RCV1-V2(Reuters Corpus Volume 1-2) dataset, which was also used for empirical study in our experiment, a detailed table shows SVM$^{perf}$ran in 66 seconds whereas SVM-Light needed 23642 seconds[2].

Although SGD and SVM$^{perf}$are optimizing two different objective functions in the actual implementation, we fixed $\lambda$ as default value $10^{-5}$ such that for prototypes' demonstration, the SGD will beat whatever SVM$^{perf}C$'s variations.

SVM$^{perf}$optimizes the following objective function:

$$\phi(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{m}\sum_{t=1}^{m}\ell(y_t(\mathbf{w}\cdot\mathbf{x}_t)),$$

where $\ell$ is the hinge loss, $\max\{0, 1 - y_t(\mathbf{w}\cdot\mathbf{x}_t)\}$.

SGD optimizes the related criterion

$$\phi'(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m}\sum_{t=1}^{m}\ell(y_t(\mathbf{w}\cdot\mathbf{x}_t))$$

which, after rescaling, is equivalent to

$$\frac{1}{2}||\mathbf{w}||^2 + \frac{1}{\lambda}\frac{1}{m}\sum_{t=1}^{m}\ell(y_t(\mathbf{w}\cdot\mathbf{x}_t)),$$

i.e., the same objective function as SVM$^{perf}$, with $C = \frac{1}{\lambda}$. For most of our experiments, we use the default value $\lambda = 10^{-5}$, though some later experiments will explore the impact of varying $\lambda$ in SGD as well as $C$ in SVM$^{perf}$. Based on the results of [2], we use the default number 5 of passes over the data for SGD.

In the SGD program, hinge loss penalty can be available if specify the type of loss functions during code compilations. The SVM$^{perf}$, by default, will use hinge loss for optimization.

## 3.1 Synthetic Data Generation

We experimented with data generated i.i.d., where each example $(\mathbf{x}, y) \in \mathbb{R}^n \times \{-1, 1\}$ was generated from one of the following two sources. (We experimented with various values of $n$.)

### 3.1.1 Random-8 and the canyon effect

The Random-8 source was designed to promote the canyon effect while including some of the characteristics of web spam. Data from this source was generated from a uniform mixture of eight spherically symmetrical Gaussians. Since the mean of each Gaussian was randomly chosen, we call this source the *random-8* source. This source was simple enough that, rather than estimating test error using test data, we could calculate it analytically.

The parameters of the eight Gaussians were chosen randomly. Each of the eight centers $\boldsymbol{\mu}_j$ was chosen uniformly at random from $[-1, 1]^n$, and each covariance matrix $\mathbf{C}_j$ was $\sigma_j^2\mathbf{I}$, for a $\sigma_j$ chosen uniformly at random from $(0, 0.1]$. The typical label associated with each Gaussian, $y_{G_j}$ was also chosen uniformly at random from $\{-1, 1\}$.

Each example $(\mathbf{x}, y)$ was generated by first choosing one of the eight Gaussians (from the uniform distribution), and generating a feature vector $\mathbf{x}$ from the chosen Gaussian. The label $y$ starts as the typical label for the chosen Gaussian, but is then flipped with noise probability $\eta$, e.g. $\eta = 10\%$.

When the dimensionality $n$ is large enough, the positive and negative centers are likely to be linearly separable. Furthermore, the $\sigma_j$ values are small, so the error rates of good hypotheses usually approach the noise rate $\eta$.

The process for generating the parameters of the canyon effect source is summarized in Algorithm 1, and the training set generation is summarized in Algorithm 2.

Both SGD and SVM$^{perf}$learn a hypothesis weight vector $\mathbf{w}$. Instead of using test data, we can analytically calculate the generalization error of a linear threshold hypothesis for the Random-8 source. (The analytical expression could play a role in

**Algorithm 1** Random-8 Gaussian generation

```
1: Input: Number of group $N$; Dimension $n$; Max variance radius $r = 0.1$
2: for $j = 1$ to $N$ do
3:     generate $\boldsymbol{\mu}_j \leftarrow \vec{0}$
4:     for $k = 1$ to $n$ do
5:         update $\mu_{jk} \leftarrow rand \in [-1, 1]$
6:     end for
7:     generate $\sigma_j \leftarrow rand \in (0, r]$
8:     if $rand < 1/2$ then
9:         $y_{G_j} = +1$
10:    else
11:        $y_{G_j} = -1$
12:    end if
13: end for
```

**Algorithm 2** Random-8 Training data generation

```
1: Input: Number of training set $T$;Dimension $n$; Noise level $\eta$
2: for $i = 1$ to $T$ do
3:     $j \leftarrow$ integer $rand \in [1, N]$
4:     set $\vec{\mathbf{x}}_i \leftarrow G_j \sim (\boldsymbol{\mu}_j, \sigma_j^2\mathbf{I})$
5:     if $rand < \eta$ then
6:         $y_i \leftarrow -y_{G_j}$
7:     else
8:         $y_i \leftarrow y_{G_j}$
9:     end if
10: end for
```

a future theoretical analysis of this source.)

The probability of a half-space with respect to spherical Gaussian is easy to compute analytically. By translating the center to the origin, projecting onto the direction of the normal vector, and exploiting the fact that the result of such a projection is a one-dimensional Gaussian.

Consider an arbitrary spherical Gaussian $G$ with mean $\boldsymbol{\mu}$ together with covariance matrix $\sigma^2\mathbf{I}$, and an arbitrary half-space $\mathbf{w} \cdot \mathbf{x} + b \leq 0$. WLOG assume that $\boldsymbol{\mu}$ is in the half-space (if not, take one minus the probability below). Let $d \geq 0$ be the distance between $\boldsymbol{\mu}$ and the half-space boundary, and $\mathbf{z}$ be the point on the boundary closest to $\boldsymbol{\mu}$. Now translate the space by $-\boldsymbol{\mu}$ to center the Gaussian, and then rotate the space so that the translated $\mathbf{z}$ gets mapped to $(d, 0, 0, \ldots, 0)$. Note that the original half-space has been transformed to the halfspace $x_1 \leq d$. Now the probability that a point $\mathbf{x}$ drawn from $G$ is in the original half-space is equal to the probability that a point $\mathbf{x}$ drawn from the transformed Gaussian has a first coordinate at most $d$. Since spherical Gaussians are preserved under marginals, translation and rotation, the probability that the transformed Gaussian generates a point with first coordinate at most $d$ is $\int_{-\infty}^{d} \mathcal{N}(x \mid 0, \sigma^2)dx = \Phi(d/\sigma)$.

Once we have the probabilities of the positive and negative prediction regions under each of the Gaussians, a straightforward calculation using the mixture weights and noise probability gives the generalization error rate of the half-space hypothesis.

The detailed derivations are in the *Analytical Error Probability* section:

**Analytical Error Probability:** From the setup above, we know the probability density of the multi Gaussian distribution, particularly for $\mathbf{x}_i$ generated from $G_j$, will be: $f(\mathbf{x}_i) = f(x_1, x_2, ..., x_n) =$

$$\frac{1}{(2\pi)^{n/2} \left|\mathbf{C_j}\right|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)'\mathbf{C_j}^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\}.$$

Now assuming the cluster $G_j$ is pure without noise examples, which indicates $\eta = 0$, the expected probability of having wrong predictions on $G_j$ with learned $\mathbf{w}$, is identical to wrong predictions of a rotated hypothesis $\mathbf{w}'$ which could be orthogonal to any base direction in the $\mathbb{R}^n$, around the mean $\boldsymbol{\mu}_j$ of $G_j$. Without loss of generality, we assume the right side of the $\mathbf{w}$ get wrong prediction, and rotation will stop when the norm of $\mathbf{w}'$ is aligned to $x_1$ direction. The interception of $\mathbf{w}'$ on axis $x_1$ is $d$, whose distance to $\boldsymbol{\mu}_j$ equals to the distance from $\mathbf{w}$ to $\boldsymbol{\mu}_j$. For opposite correct/wrong prediction scenario, the derivations only need changing the lower limit and upper limit of the integral. That means:

$$
\begin{aligned}
\widehat{error}_j^{\eta=0} &= P(\text{wrong side}|\mathbf{w}, G_j) \\
&= P(\text{wrong side}|\mathbf{w}', G_j, \text{with the same distance } d) \\
&= \underbrace{\int_{-\infty}^{+\infty} ... \int_{-\infty}^{+\infty} \int_{d+\mu_{j1}}^{+\infty}}_{n\ layers} f(x_1, x_2, ..., x_n) dx_1 dx_2 ... dx_n \\
&= \int_{d+\mu_{j1}}^{+\infty} (.. \int_{-\infty}^{+\infty} (\int_{-\infty}^{+\infty} f(x_1, x_2, ..., x_n) dx_n) dx_{n-1} ..) dx_1 \\
&= \int_{d+\mu_{j1}}^{+\infty} (... \int_{-\infty}^{+\infty} f(x_1, x_2, ..., x_{n-1}) \cdot 1 \cdot dx_{n-1} ...) dx_1 \\
&= ... \\
&= \int_{d+\mu_{j1}}^{+\infty} f(x_1) \underbrace{\cdot 1 \cdot 1 ... \cdot 1}_{n-1} dx_1 \\
&= \int_{d+\mu_{j1}}^{+\infty} f(x_1) dx_1 \\
&= \int_{d+\mu_{j1}}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_1-\mu_{j1})^2}{2\sigma_j^2}} dx_1 \\
&= \int_{d}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{u^2}{2\sigma_j^2}} du \\
&= 1 - P(x | x \leqslant d, X \sim (0, \sigma_j^2)) \\
&= 1 - \Phi_{0,\sigma_j}(d)
\end{aligned}
$$

where $d = \frac{\mathbf{w}\cdot\boldsymbol{\mu}_j + b}{\|\mathbf{w}\|_2}$ is the distance from Gaussian center $\boldsymbol{\mu}_j$ to the hyperplane $\mathbf{w}$ or rotated $\mathbf{w}'$. Note $b$ is the threshold from the dual form of the objective function that would be also learned by both SVM$^{perf}$and SGD.

$$\mathbf{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{l} \boldsymbol{\alpha}_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1],$$

where $\boldsymbol{\alpha}_i \geqslant 0$ are the Lagrangian multipliers for constrains.

Figure 2 shows the rotation of the $\mathbf{w}$ and $d$ based on a single 2D Gaussian group computation.

The derivation above shows that the estimated error on each $G_j$ for each leaned hypothesis $\mathbf{w}$ can be simplified to a calculation of one dimensional PDF(Probability Density Function) $\Phi_{0,\sigma_j}(d)$ due to the fact that the initial settings for symmetrically spherical Gaussian would simplify the covariance matrix such that the integral will be beneficial from this simple diagonal matrix. The final estimated error need to

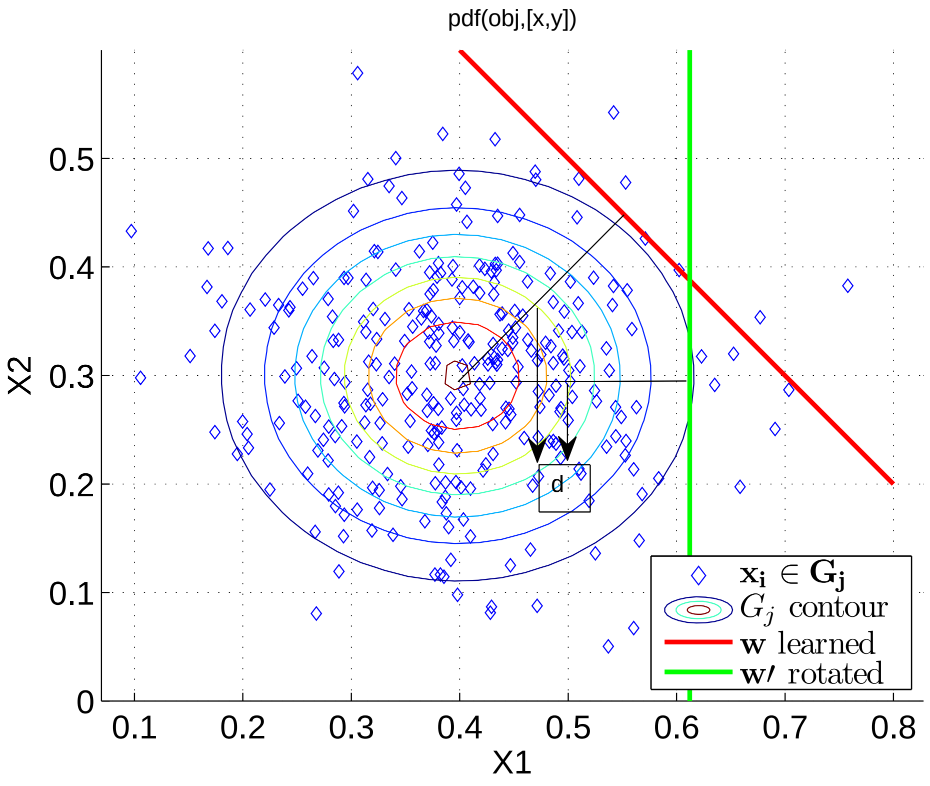


Figure 2: A 2D demo for one $G_j$ with mean as $\boldsymbol{\mu}_j$: rotating $\mathbf{w}$ centering around $\boldsymbol{\mu}_j$ to $\mathbf{w}'$, which is orthogonal to $x_1$, will have the same value of the expected error for learned hypothesis on $G_j$.

adjusted based on the noise level (with probability $\eta$, examples will have the negated label assignment). The wrong side in pure case would actually have portion $1 - \eta$ predict wrong whereas the correct side of the pure case would have $\eta$ portion actually predict wrong. From Bayes' formula, the final error is:

$$\begin{aligned}\widehat{error}_j &= P(\text{wrong side}|\mathbf{w}, G_j) \cdot (1-\eta) + P(\text{right side}|\mathbf{w}, G_j) \cdot \eta \\ &= \widehat{error}_j^{\eta=0} \cdot (1-\eta) + (1 - \widehat{error}_j^{\eta=0}) \cdot \eta \\ &= (1 - \Phi_{0,\sigma_j}(d)) \cdot (1-\eta) + \Phi_{0,\sigma_j}(d) \cdot \eta\end{aligned}$$

The total error for all $G_j$ ($N = 8$ for experiment) would be:

$$\widehat{error} = \frac{1}{N}\sum_{j=1}^{N} \widehat{error}_j \tag{1}$$

Since the one dimensional Gaussian calculation is very fast and there are existing codes to do that, the running time of the test error estimation is dramatically reduced during the experiments. There is even no need to generated the test data set, which speed up the experiments by reducing big block data writing operations to the hard drives. Concretely, the correctness of derivations can be verified by comparing the calculated estimated errors versus a large test set generated. The difference between theoretical calculation and estimate test error using 30000 test data was always less than 0.2%, in terms of accuracies, for any data generation experiments that we tried.

When the data is generated using the random-8 source, SGD beats SVM$^{perf}$with large $C$ (see Section 4.1). However, there is no clear difference between SGD and SVM$^{perf}$with small $C$. This leaves open the possibility that SGD's advantage is due to its small weight vector, i.e. that it effectively performs regularization akin to a small-$C$ SVM algorithm.

### 3.1.2 Fixed-4 source adding the classification-insensitivity effect

Now we describe the *fixed-4* source which retains the canyon effect while promoting the classification-insensitivity effect. In other words, it encourages the small-$C$ optimization to focus on large magnitude examples while essentially ignoring incorrect examples close to the origin. The fixed-4 source is a mixture of 4 Gaussians, located as in Figure 3. The two blue (+) Gaussians generate mostly positive exam-

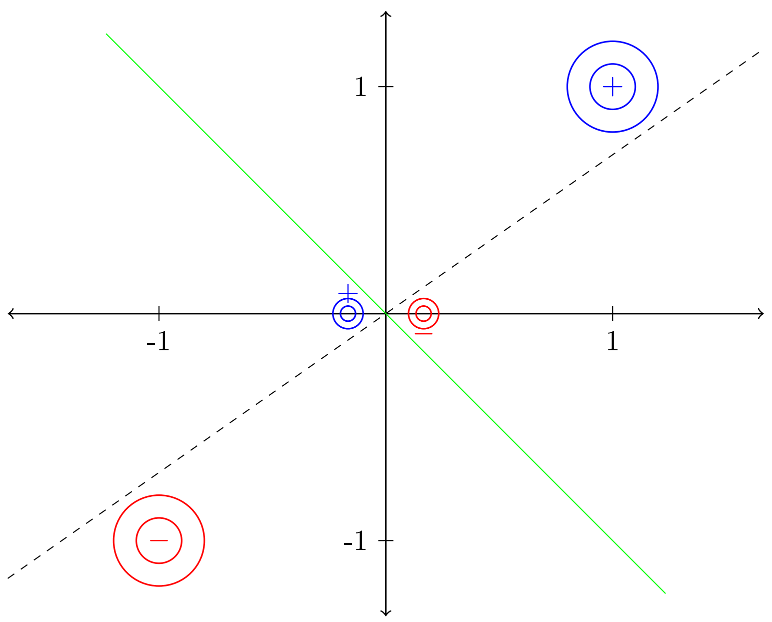


Figure 3: Fixed-4 source: a mixture of 4 Gaussians. The Gaussians located further from the origin have more weight in the mixture and are more diffuse. Linear separators tend to have small margin on all examples (dashed line), but the green hypothesis obtains large margin on most of the points at the cost of prediction errors.

ples while the red $(-)$ Gaussians generate mostly negative examples.

The fixed-4 source has two kinds of examples; small magnitude examples from the Gaussians at $(\pm\epsilon, 0)$, and relatively large magnitude examples from the Gaussians at $(+1, +1)$ or $(-1, -1)$. For a weight vector of a given size, the margins on the "large magnitude" examples will tend to be bigger than the margins on the "small magnitude" examples.

The centers of the clouds are positioned so that they are linearly separable (dashed line). However, if the far clouds are classified correctly by a large margin, then most examples from the close clouds will be misclassified (green line). Since SGD (with a large $C$) tends not to make updates on correctly classified examples, it is less likely to over-emphasize the large-magnitude examples, and thus obtain better hypotheses.

To preserve the canyon effect, we continue to keep the variances of the Gaussians small and flip a fraction $\eta = 5\%$ of the labels to generate similar examples with contradictory labels.

The fixed-4 source in higher dimensions splits the $n$ dimensions between $k$

Table 1: Fixed-4 Gaussian Means.

| $G'_j$ | $y_{G'_j}$ | WEIGHT | $\sigma'_j$ | $\boldsymbol{\mu}'_j$ $(\epsilon = 0.1)$ |
|---|---|---|---|---|
| $G'_1$ | +1 | 40% | 0.1 | $\vec{1} : (\underbrace{1, 1, ..., 1}_{n})$ |
| $G'_2$ | +1 | 10% | 0.032 | $(\underbrace{-\epsilon, .., -\epsilon}_{k}, \underbrace{0, ..., 0}_{n-k})$ |
| $G'_3$ | -1 | 10% | 0.032 | $(\underbrace{\epsilon, .., \epsilon}_{k}, \underbrace{0, ..., 0}_{n-k})$ |
| $G'_4$ | -1 | 40% | 0.1 | $-\vec{1} : (\underbrace{-1, -1, ..., -1}_{n})$ |

**Algorithm 3** Training data generation - fixed-4 source

```
1: As algorithm 2 ...
2: $j \leftarrow$ rand integer $j \in [1, N]$ with prob. of $wt_j$
3: As algorithm 2 ...
```

"key" dimensions where the means of the nearby Gaussians are non-zero and the $n-k$ "non-key" dimensions where these means are zero. (Note that calling these "key" dimensions is somewhat subjective as the $k$ "key" dimensions have contradictory signs in the Gaussians with the same typical labels.) Table 1 shows the typical label, mixture weight, standard deviation, and mean for each Gaussian $G'_j$ in the fixed-4 source. The mixture coefficients weight the "large magnitude" Gaussians more heavily, so instead of picking a Gaussian from a uniform distribution when generating examples, we now use a weighted mixture. The selected Gaussian's typical label is flipped with probability $\eta = 0.05$.

Once the learned $\mathbf{w}$ is obtained, the generalization error for this source is easily computed along similar lines as the Random-8 source. For data generated from this source, SGD with the default $\lambda = 10^{-5}$ generalized better than SVM$^{perf}$with both small and large $C$ values, especially when the number $k$ of key dimensions is small (See Section 4.2).

## 3.2 Natural data

The original RCV1-V2 data set [11] consists of 23149 training documents and 781265 testing documents with 47152 features. We processed the data using software provided for that purpose from the SGD package [3]. Inside the package, there is a program to unzip and shuffle the original data, and then format to a database that can be read by both SVM$^{perf}$and SGD package. The default data set generation swaps between the original training and testing set in order to do the running time measurements in [2]. However, we switch back to the larger test set to more accurately estimate the generalization error rate.

## 4 Results

For various sources and algorithms, we plot the generalization error as examples are added to the training set. Because the most significant differences between algorithms are seen over the first few thousand examples, we concentrate our graphs on this part of the curve.

### 4.1 The canyon effect alone

The results from the Random-8 source are shown in Figure 4 and Figure 5.

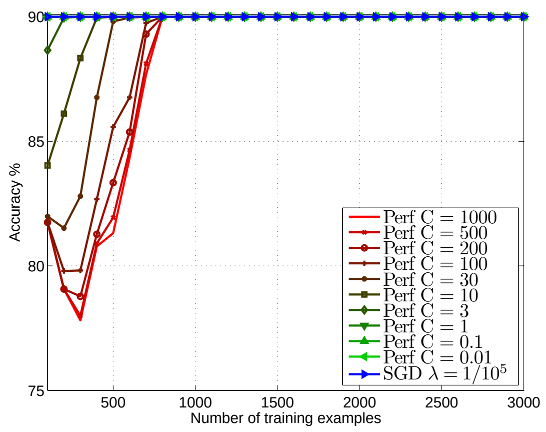
They show that, over a variety of dimensions $n$ and noise rates $\eta$, when the number of examples is only moderately large, the SGD hypothesis is much more accurate than the SVM hypothesis trained with large values of $C$ ($C > 30$).

When $C$ is small enough ($C < 30$), the accuracy of the SVM hypothesis quickly (in terms of number of training examples) overlaps the accuracy of the SGD hypothesis. Again, this holds for a variety of noise rates and numbers of variables. The Bayes-optimal accuracy is around 90%, reflecting the $\eta = 0.1$ noise rate.

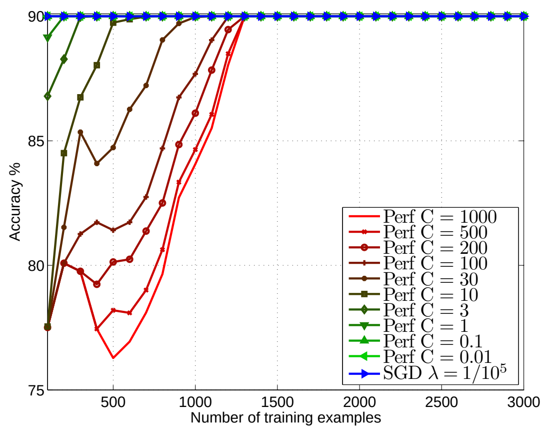
Before conducting the experiments, we expected the canyon effect to result in an advantage of SGD over large-$C$ SVM$^{perf}$.

An intriguing phenomenon is the early dip in accuracy of large-$C$ SVM$^{perf}$as more training examples are added. Figures 4(a), 4(b) and 4(c) show that the number of examples to reach the bottom of the dip increases with the dimensionality for large-$C$ SVM$^{perf}$.

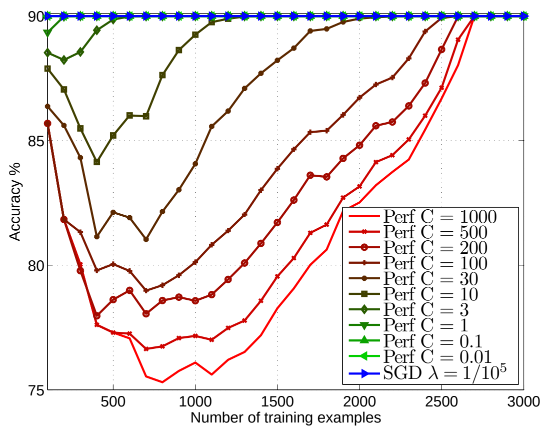
The depths and locations (in terms of number of training examples) of this valley for $C = 1000$ are given in Table 2.



(a) $n = 100$, $\eta = 0.1$

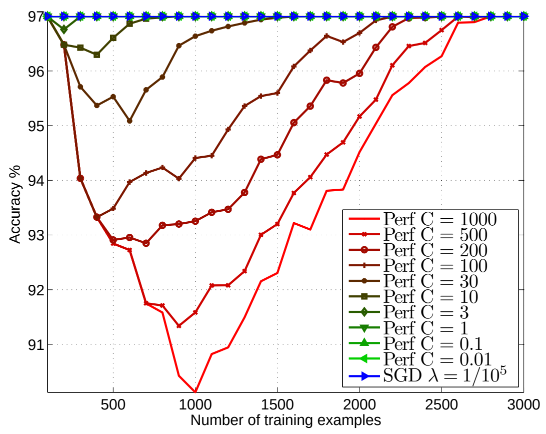


(b) $n = 200$, $\eta = 0.1$

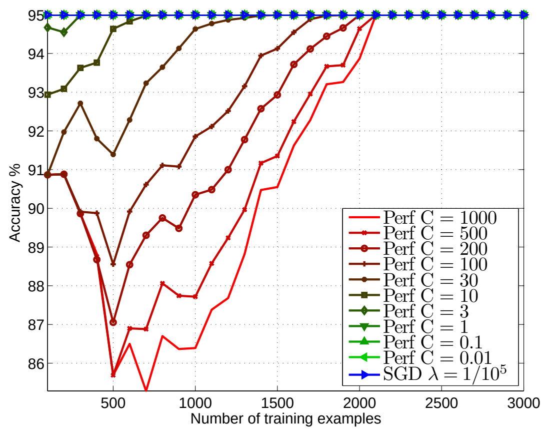


(c) $n = 400$, $\eta = 0.1$

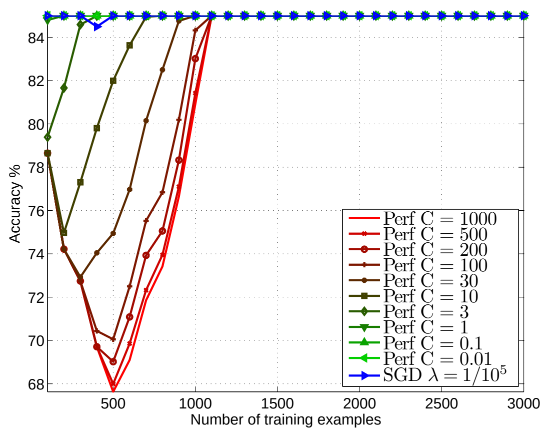
Figure 4: Random-8 accuracies as a function of the training set size for SVM$^{perf}$ with a variety of $C$ values and SGD - vary the dimensionality. Note: The curves for small $C$ (i.e. $C \leqslant 1$) are covered by the curve for SGD.



(a) $n = 200$, $\eta = 0.03$



(b) $n = 200$, $\eta = 0.05$



(c) $n = 200$, $\eta = 0.15$

Figure 5: Random-8 accuracies as a function of the training set size for SVM$^{perf}$with a variety of $C$ values and SGD - vary the noise rate. Note: The curves for small $C$ (i.e. $C \leqslant 1$) are covered by the curve for SGD.

Table 2: Observations of Dips for c = 1000

(a) Random-8 source with $\eta = 0.1$

| dimension | $\#_{worst}$ | *best* | *worst* | *best* $-$ *worst* |
|---|---|---|---|---|
| $n = 100$ | 250 | 90% | 77.8% | 12.2% |
| $n = 200$ | 500 | 90% | 76.3% | 13.7% |
| $n = 400$ | 1000 | 90% | 75% | 15% |

(b) Random-8 source in dimension $n = 200$

| noise rate | $\#_{worst}$ | *best* | *worst* | *best* $-$ *worst* |
|---|---|---|---|---|
| $\eta = 3\%$ | 1000 | 97% | 90.1% | 6.9% |
| $\eta = 5\%$ | 700 | 95% | 85.3% | 9.7% |
| $\eta = 10\%$ | 500 | 90% | 76.3% | 13.7% |
| $\eta = 15\%$ | 500 | 85% | 67.6% | 17.4% |

Figures 5(a), 5(b) and 5(c) show that the dip bottom occurs after fewer examples as the noise level increases. As might be expected, for a fixed dimensionality (i.e. $n = 200$) the more noise the greater the gap between the dip bottom and the Bayes optimal accuracy.

The number of examples at the dip bottom appears to be decreasing as the noise-rate increases, though the bottom is at the same place for $\eta = 10\%$ and $\eta = 15\%$. For SVM$^{perf}$with large $C$, the amount of decrease in the accuracy also appears to be increasing with $\eta$, but perhaps starting to asymptote around 14-15%.

The reason for large $C$ SVM$^{perf}$classifiers to perform badly is that the large penalty weighting on the loss function $\ell(y_t(\mathbf{w} \cdot \mathbf{x}_t))$ will severely affect the $\mathbf{w}$ for each examples coming into the training set. At the early stage of the training, because the noisy data point has the equal opportunity as the normal data to be chosen in the training set, the large $C$ learners will attempt to accommodate them most to reduce the loss. At this stage, a couple of more examples came in training set could result in radically changing the direction and length of vector $\mathbf{w}$ such that reducing even one loss classification will make $\phi$ smaller. Until to the worst valley point, the bad noisy data dominate the updating of hypothesis $\mathbf{w}$. As the number of training examples increase to very large, there would be no more linear separable solution for any learned hypothesis, no matter how the $C$ is, the $\mathbf{w}$ has to make lots of wrong classifications anyway since the VC dimensionality for $\mathbb{R}^n$ with hyperplane is only $n+1$, which is $\ll$ the number in the training set. In this situation, $\mathbf{w}$ will globally balance the total loss on all data points. Consequently, Bayes-optimal prediction rate is acquired. On the contrast, small $C$ is almost totally ignoring which ever noisy data point it sees. It would focus on global data in the view to minimize the objective function. Once new noisy data come in, instead of twisting in the hyperspace a lot as large $C$ does, these learners will just change a little bit to accommodate those points. The SGD has large penalty in the sense of optimizing the objective function, but it is not a so nice optimizer as SVM$^{perf}$. Although bad approximating to the QP programing solution make it mathematically worse towards the analytical form of the objective function, the SGD would not really updating $\mathbf{w}$ too much after encountering a canyon pair such that it will not better approach $\phi_{min}$ mathematically.

As a result, here is a potential explanation of these dips. When there are relatively few noisy examples, it is less likely that pairs of very similar examples with opposite labels will be seen. When there are very many examples, then the effects of irrelevant directions on $\mathbf{w}$ will cancel each other out. The fact that this takes longer for larger values of $n$ is consistent with the intuition that more directions need to be canceled in that case.

When dimensionality $n$ of the data increases, the large-$C$ SVM algorithm needs more training examples to reach the valley performance, because the VC dimension $n+1$ get larger such that more training data are needed to make $\mathbf{w}$ in the situation that cannot compromise those noisy points anyway. As a result, the valley point increase as the $n$ get larger. Similarly, when noise level are getting larger, the balanced situation, in which even large $C$ learners should have a global consideration since no way to reduce loss on bad noisy points, would come earlier because there are so much noise to quickly make $\mathbf{w}$ no way to linearly separate those points, where the worst performance occurs. To quantify the form of the shifted valley points and the depth of the valley, which is $best - worst$ accuracies, more theoretical analysis work needs to be done. Nevertheless, the first prototype gave us strong hints that noise in the clouds would help bad optimization beat good optimizers with large penalty on loss function.

In fact, most of the curves in Figure 4 and Figure 5 indicate a relationship between the dimensionality, the expected number of the noisy examples, and the number of examples at the dip bottom when $C = 1000$. In particular, the dip bottom tends to occur when the expected number of noisy examples is about $n/4$, one-quarter of the example dimensionality.

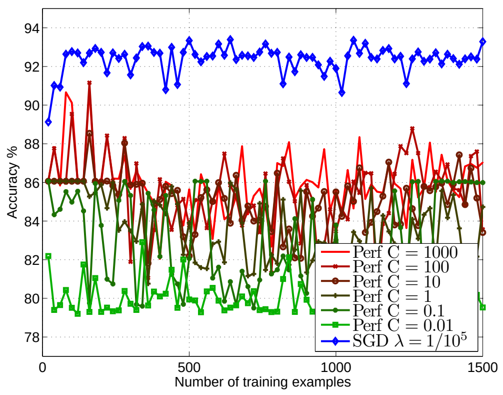
## 4.2 Adding the classification-insensitivity effect

The Fixed-4 source exhibits the canyon effect by having tight clouds of examples with label noise. In addition, it also exhibits the classification-insensitivity effect, which causes trouble for algorithms that aggressively set $C$ to a low value. Figure 6 shows the results of these experiments. The behavior of the large-$C$ SVM algorithm is qualitatively similar to the Random-8 results, including the dip in accuracy. The dips follow the same trends as in the Random-8 source: larger values of $n$ make the valley point occur with more examples for a fixed noise level.

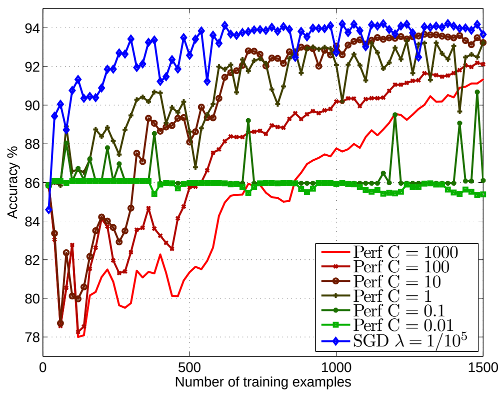
The result shows the valley point of large $C$ learners still around 250 training examples for $n = 100, \eta = 10\%$, which is pretty consistent with Figure 4(a) and Table 2 (a).

However, the small $C$ learners also fail to do well. When $C$ is small enough, the optimization essentially maximizes the total margin and "gives up" on the smaller clusters near the origin. The benefit of reducing the loss on the close points is too small relative to the benefit of increasing the margin on the larger clouds. As shown in Figure 3, there is a hyperplane that obtains large margin on most of the points, but has a significant number of classification errors. For example, the uniform vector $(\ell/\sqrt{n}, \ldots, \ell/\sqrt{n})$ achieves margins like $\ell\sqrt{n}$ on points from the larger clusters while losing only about $-\epsilon\ell k/\sqrt{n}$ margin on the points in the smaller clusters.

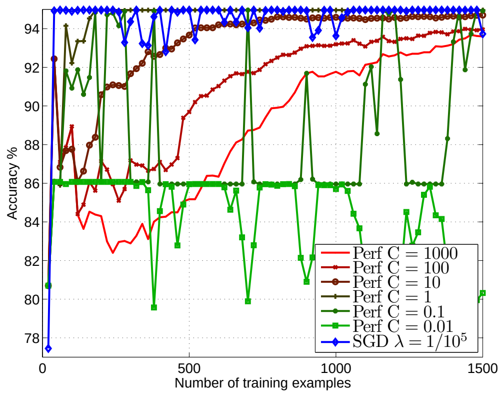
When $k$, the number of key (non-zero) dimensions in the means of the small clouds increases, SGD gets worse and eventually may become worse than the large-$C$ SVM. As $k$ increases, the angle between the centers of $G_1'$ and $G_2'$ (also with the angle between the centers of $G_3'$ and $G_4'$) becomes more obtuse. (This can be



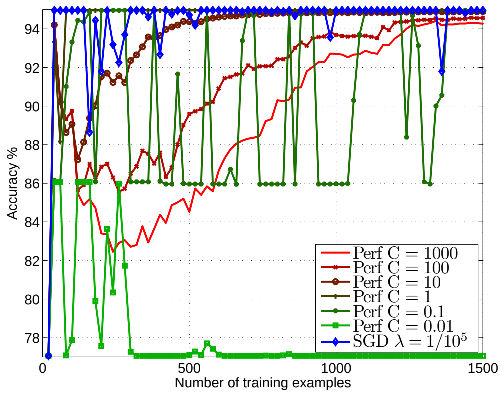
(a) $n = 3,\ k = 1$



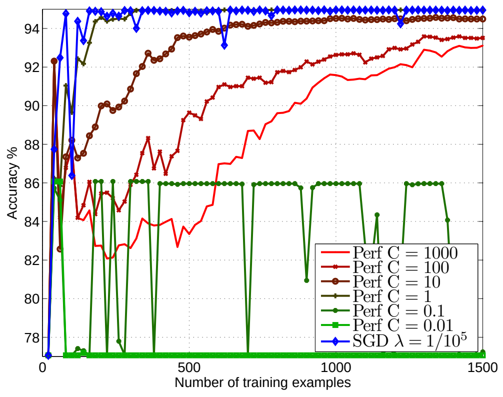
(b) $n = 100,\ k = 1$



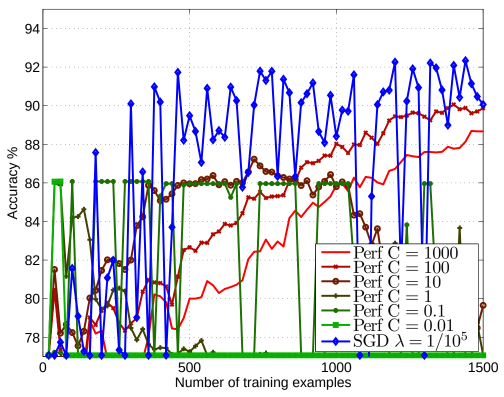
(c) $n = 100,\ k = 10$



(d) $n = 100,\ k = 50$



(e) $n = 100,\ k = 90$



(f) $n = 100,\ k = 99$

Figure 6: Fixed-4 accuracies as a function of training set size for SVM$^{perf}$with a variety of $C$-values and SGD. (a) and (b) show that, when $k = 1$, SGD is generally more accurate. Figure (a) and (c)- (f) show fluctuating accuracy for SVM$^{perf}$when $C = 0.1$. (e) and (f) show the advantage of SGD is less clear for larger values of $k$, where the classification-insensitivity effect is weaker.

verified by computing the cosines of those angles using the dot products between the centers.) This makes SGD converge more slowly, for a similar reason that it is protected against the canyon effect. At the same time, the Gaussians that are close to the origin are becoming less so, weakening the classification-insensitivity effect. The $k$ parameter impacts the maximum margin. Smaller $k$ values allow the data to be separated with a larger margin while $k$ values approaching $n$ (the dimensionality of the instances) lead to very small margins. When $k \approx n$, SVM$^{perf}$with super large $C$ can find the resulting very small gap and separate the clusters while SGD has trouble finding this gap and performs more like the low-$C$ SVM. In the case $k = 1$, where the classification-insensitivity effect is strongest, the advantage of SGD is clear.

**Explanation of Fluctuations:** The SVMs with small-$C$ have very unstable accuracies on the Fixed-4 source. When $C = 0.1$ in particular, we see the accuracy often oscillating between two values as we add more examples. To explain this phenomenon, we examined the simple 3D case of Figure 6(a). We took all hypothesis learned during the oscillations by small $C$ SVMs and calculated the angles between them. Surprisingly, they were all almost parallel. We then found two nearby points where the accuracies on the $C = 0.1$ curve jumped by a large amount and plotted the two planes corresponding to their hypotheses (see Figure 7). This plot reveals that the accuracies oscillate because the hypotheses shift across the smaller clusters. Some hypotheses get one of the two smaller clusters wrong, while others get both wrong.

Define the most frequently appearing higher accuracies' group for small $C$ learners, e.g. $C = 0.1$, is $\mathbf{w}^{up}$ and the flip side, $\mathbf{w}^{down}$ represents the group of the most frequently observed lower accuracies. For most $\mathbf{w}$ from different groups, $cos(\frac{\mathbf{w}_i^{up} \cdot \mathbf{w}_j^{down}}{\|\mathbf{w}_i^{up}\|_2 \cdot \|\mathbf{w}_j^{down}\|_2}) \approx 1$

Figure 7 is from the $C = 0.1$ curve in the 3D space of Figure 6(a). The hyperplanes in Figure 7 represent two consecutive hypotheses, from 160 and 180 training examples respectively. The learner switched from the purple (accuracy 79%) to the green (accuracy 86%) hypothesis. The 20 newly added training points came

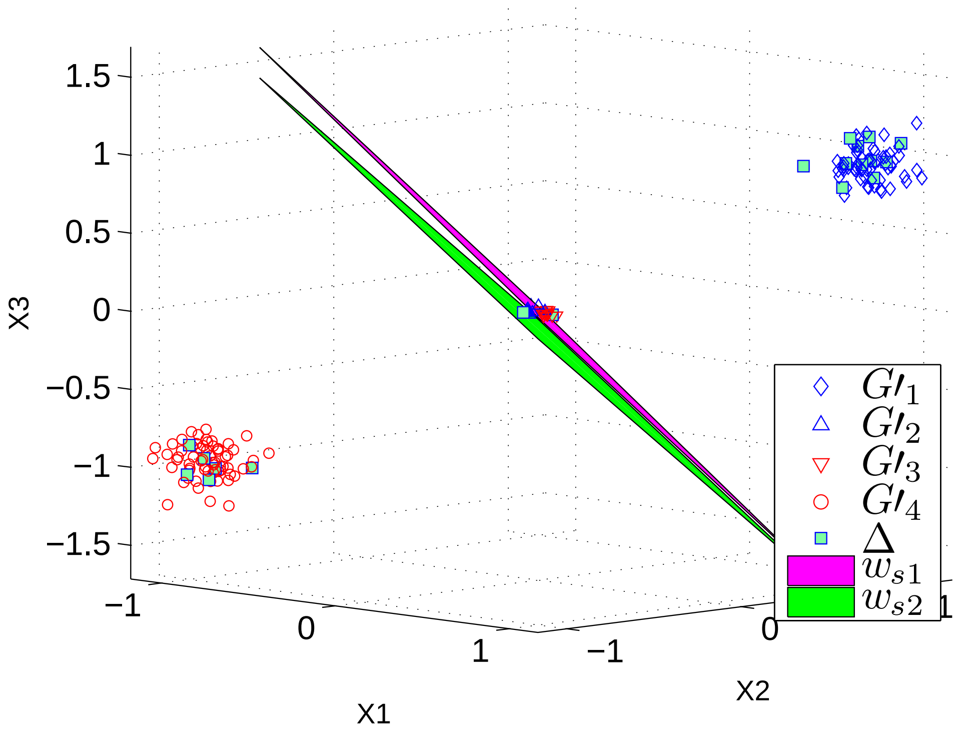


Figure 7: Change in hypothesis $\mathbf{w}$ after seeing $|\Delta| = 20$ more training examples: $\mathbf{w}_{s1}$ and $\mathbf{w}_{s2}$ are almost parallel. The actual number of added points from Gaussians $G'_1 : G'_2 : G'_3 : G'_4$ is $11 : 2 : 1 : 6$.

mostly from the upper (blue) cluster $G'_1$, causing the margin-optimizing hyperplane to be pushed down towards the lower (red) cluster. In this case, the push happened to shift the hyperplane across most of the smaller red cluster, and increased the accuracy by 7% (with the label noise, shifting across an entire small cluster changes the accuracy by 9%).

This is effect can be calculated from the distribution since $Acc_{\mathbf{w}_{s1}} = 40\% \times 2 \times (1-\eta) + 10\% \times 2 \times \eta = 0.76 + 0.01 = 77\%$ and $Acc_{\mathbf{w}_{s2}} = (40\% \times 2 + 10\%) \times (1-\eta) + 10\% \times \eta = 0.855 + 0.005 = 86\%$. Most of the switched accuracies for small $C$ learners for 3D in Figure 6 are all back and forth between the range of $[\mathbf{w}_i^{down}, \mathbf{w}_i^{up}]$. One more added point in the furthest two clouds could make small $C$ learners totally come across the small clouds region completely due to the large margin they would gain.

It is also observed that the value of $\mathbf{w}_i^{down}$ and $\mathbf{w}_i^{up}$ for $C = 0.1$ in Figure 6(c) and Figure 6(d) are increased. Due to the fact that it is not observable in 3D, there is no plot for the learned $\mathbf{w}_i^{down}$ and $\mathbf{w}_i^{up}$. For the observed n = 100 dimensionality and $C = 0.1$, the value of cosine similarity are still close to 1 but the angle between them are relatively bigger comparing with $C = 0.01$ scenario. For example, in Figure 6(c), the value of cosine similarity for fluctuated consecutive hypothesis (e.g. $\mathbf{w}_{360}$ with 86.00% accuracy vs $\mathbf{w}_{380}$ with 94.96%) is $\frac{\mathbf{w}_{360} \cdot \mathbf{w}_{380}}{\|\mathbf{w}_{360}\|_2 \cdot \|\mathbf{w}_{380}\|_2} = 0.9897$ indicating 8.22 degree while for $C = 0.01$ is only 1.72 degree. Better explanations leave open for future investigation on this phenomenon.

## 4.3 Stability of SGD

In our many run experiments for each setup above, we did see SGD occasionally perform badly due to the “unlucky” shuffled sequence for training set, which contains the noise data point at the end of the sequence resulting fatal updating on the already-good hypothesis. It is also a general issue for online algorithm when dealing with the last training example who is noise point. To avoid this, ASGD was used to achieve the similar performance as SGD but with better stability. Another way is to average, or to use median accuracies, from many runs with different shuffling for each training step.

## 4.4 Irrelevant Feature Evaluations

We also considered the two sources modified with an additional $n$ irrelevant features (making $2n$ features total). We run part of experiments above with irrelevant features. There are no observed significant effects on the algorithm's learning curves.

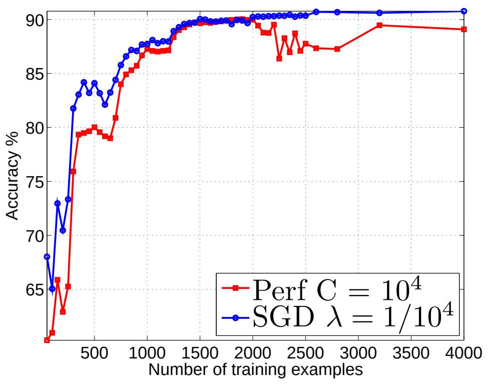
## 4.5 Study using natural data: RCV1-V2

For the RCV1-V2, we added examples to the training data one at a time, in random order, and plotted the test set error. The results show that SGD algorithm can get better generalization than SVM with any $C$ for up to 2000 training examples. Figure 8(a) shows apple to apple comparisons for both large penalty learners and 8(b) for small ones. We set $\lambda$ to $1/C$ so that the two algorithms, SVM$^{perf}$and SVM-SGD packages are using the same objective function. In both cases SGD beats the more effective optimizer SVM$^{perf}$.

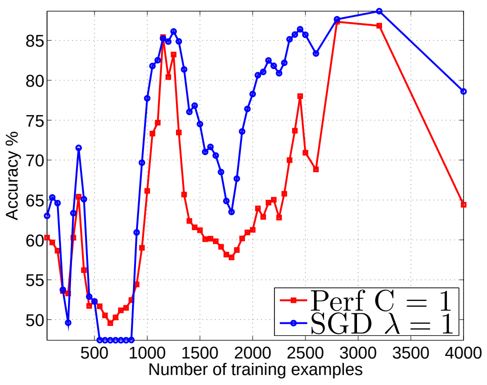
Figure 8(c) shows the performance of all learners we used on synthetic sources. The default $\lambda = 10^{-5}$ for SGD has consistently better generalization than the SVM hypotheses.

In Figure 8(a), the difference between SGD and SVM$^{perf}$is large at 400 examples. We did two things to verify that the canyon effect is involved in this difference. First, we found the closest pairs $\mathbf{x}_s, \mathbf{x}_t$ of points whose respective labels $y_s = 1, y_t = -1$ were different (within these 400 examples). We then scatter-plotted the cosine similarities between the values of $\mathbf{x}_s - \mathbf{x}_t$ for these examples and the $\mathbf{w}$ vectors (after the 400 examples) produced by the SVM and SGD algorithms in Figure 9(a). This shows that the SVM algorithm has much more "canyon activity".

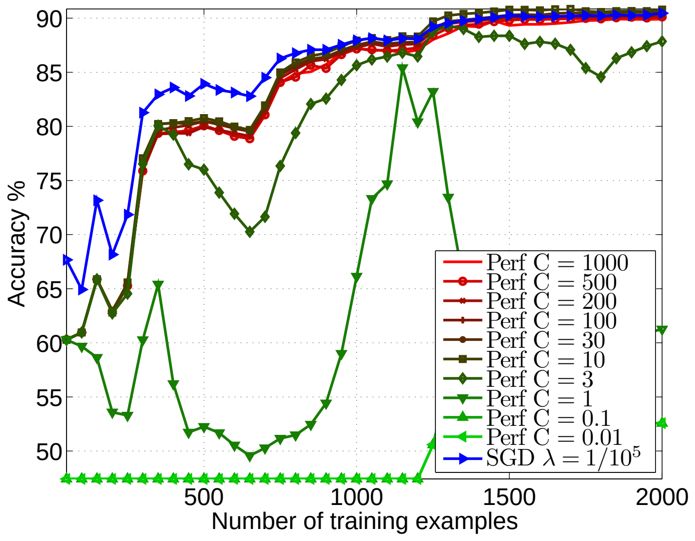
Second, we continued to run SGD on these first 400 examples for 50,000 epochs. The accuracy of SGD decayed from 83.81% to 79.76%, very close to the 79.17% accuracy of SVM$^{perf}$(see Figure 8(d)). The Figure 9(b) concretely shows that as large amount of number of epochs runs for SGD, the learned hypothesis will no



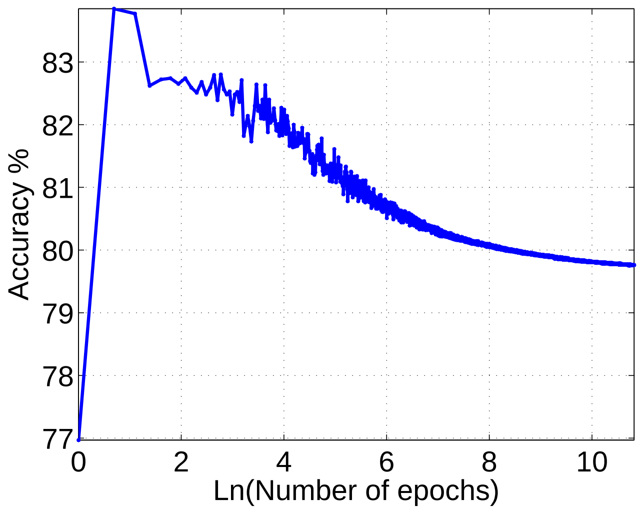
(a) $\lambda = 1/C = 1/10^4$



(b) $\lambda = 1/C = 1$

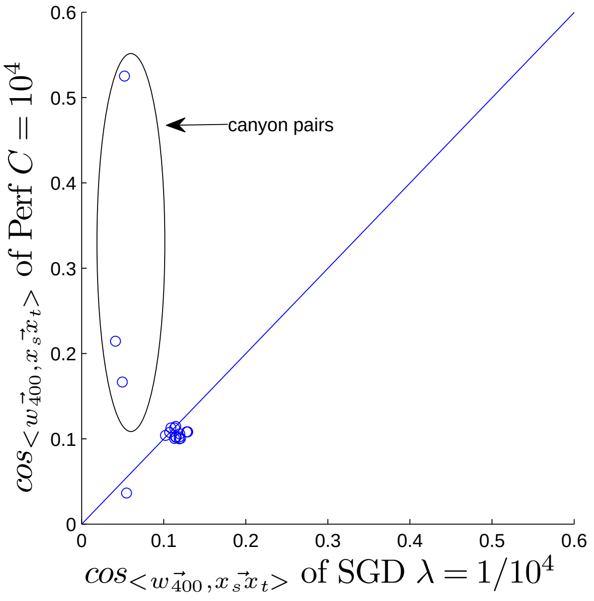


(c) RCV All Curves

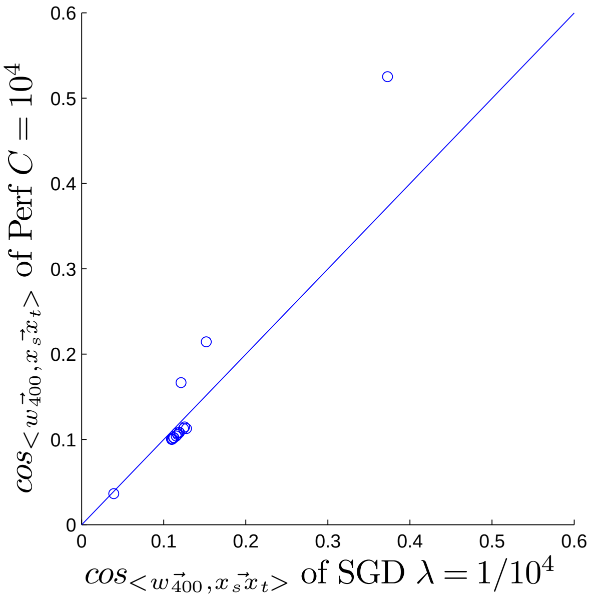


(d) Accuracy decay

Figure 8: (a), (b) and (c) shows accuracy comparison based on one shuffle. (d) shows SGD's accuracy decay over many epochs on the initial 400 examples from (a).



(a) $\lambda = 1/C = 1/10^4$, SGD-5 epochs



(b) $\lambda = 1/C = 1/10^4$, SGD-50000 epochs

Figure 9: The cosine similarity for different hypothesis with top 20 closest but differently labeled canyon-pairs.

longer immune to canyon effect.

We think that the canyon and classification insensitivity effects give SGD its advantage on the RCV1-V2 data, but we cannot rule out the possibility of other undiscovered effects also benefiting SGD.

## 5 Discussion

In this thesis, we have begun to explore the possibility that SGD algorithms generalize well in part because they are less sensitive to noise. Put another way, while they do not do a good job of optimizing the stated objective function, they effectively optimize a different function, which accords less importance to groups of examples that nearly contradict one another.

While we have concentrated on the SVM objective in this thesis, much of the discussion also applies for related approaches, especially regularized logistic regression, where the hinge-loss $\ell(z)$ is replaced with the logistic loss $\log(1 + e^{-z})$.

The SGD software [3] used in this work includes code for "Averaged Stochastic Gradient Descent" (ASGD) which outputs the average of all the weight vectors encountered during the SGD training process, instead of the last one. Roughly, the advantages of SGD should also hold for ASGD, for more-or-less the same reasons. We did the same suite of experiments using ASGD instead of SGD, with qualitatively similar results (but with the additional stability of ASGD in evidence). Some comparison results are provided in the appendix.

Our comparison with low-$C$ optimization confirms the hypothesis that the advantage of SGD is not simply due to its early stopping limiting the length of the weight vector. In particular, we propose two effects causing margin optimization to result in poor generalization. The canyon effect encourages large-$C$ optimizations to focus on nearly contradictory examples. In contrast, the classification-insensitivity effect causes low-$C$ optimizations to give up on classifying examples close to the origin in order to obtain good margins on examples further away. SGD is relatively immune to both effects, and our experiments with mixtures of Gaussians provide concrete evidence that a few epochs of SGD can outperform SVMs regardless of the choice of soft-margin penalty $C$.

We also showed that on natural data SGD can outperform SVM applied with a variety of values of $C$. Further examination of this natural data could confirm the extent to which it has characteristics in common with our synthetic datasets, such as tight clusters with conflicting labels.

These experiments raise a number of theoretical questions. For example, can it be shown analytically that for some source and training set size, the accuracy of SGD is higher than that of SVMs, for any setting of the regularization parameter? If so, how big can the difference in accuracies be? Can we characterize when SGD generalizes better, or at least provide sufficient conditions?

## References

[1] K. P. Bennett and E. Parrado-Hernandez. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7:1265–1281, 2006.

[2] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, 2010.

[3] L. Bottou. SGD 2.0, 2012. `http://leon.bottou.org/projects/sgd`.

[4] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. *NIPS*, 2007.

[5] L. Bottou and Y. LeCun. Large scale online learning. In *NIPS*, 2003.

[6] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. *EMNLP*, 2002.

[7] J. A. Hertz, A. Krogh, and R. Palmer. *Introduction to the theory of neural computation.* Addison-Wesley, 1991.

[8] G. E. Hinton. *Learning translation invariant recognition in a massively parallel network*, pages 1–13. Springer-Verlag, 1987.

[9] T. Joachims. Making large-scale support vector machines learning practical. In B. Sch olkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support vector machines*, pages 169–184, 1998.

[10] T. Joachims. Svm$^{perf}$ 3.00, 2009. `http://svmlight.joachims.org/svm_perf.html`.

[11] D. D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

[12] P. M. Long and R. A. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78(3):287–304, 2010.

[13] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming, Series B*, 127(1):3–30, 2011.

[14] S. Shalev-Shwartz and N. Srebro. Svm optimization: inverse dependence on training set size. *ICML*, pages 928–935, 2008.

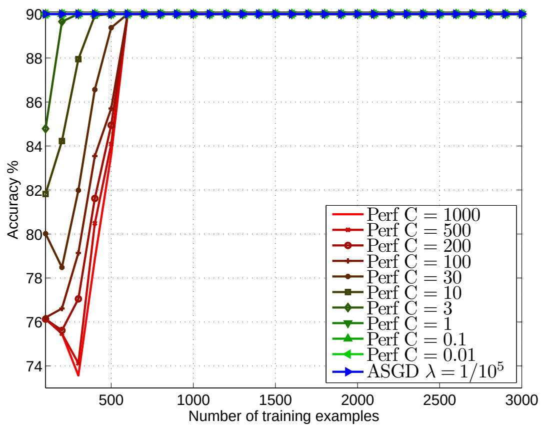
[15] S. Sra, S. Nowozin, and S. J. Wright, editors. *Optimization for Machine Learning.* MIT Press, 2011.

[16] A. N. Tychonoff and V. Y. Arsenin. *Solution of Ill-posed Problems.* Winston and Sons, 1977.

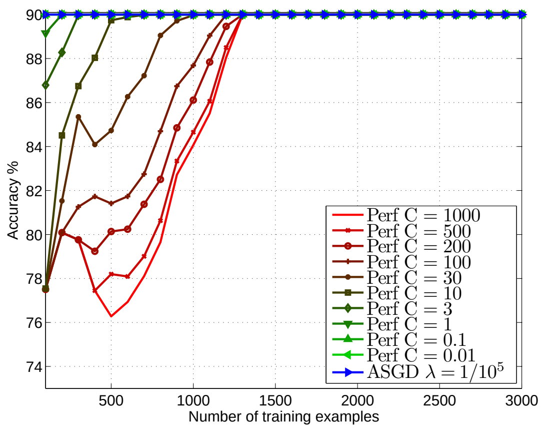
[17] V. N. Vapnik. *Statistical Learning Theory.* New York, 1998.

[18] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 116–, New York, NY, USA, 2004. ACM.

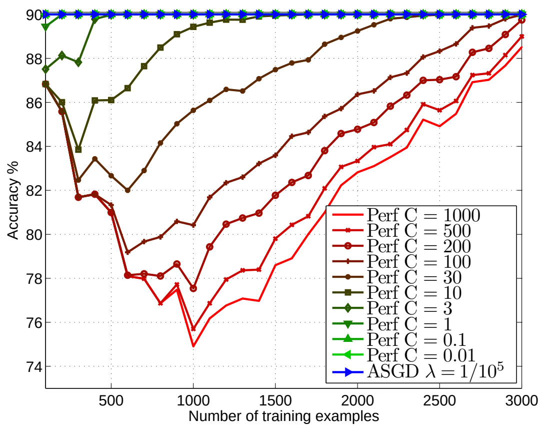
## A Random-8 ASGD Result



(a) $n = 100$, $\eta = 0.1$

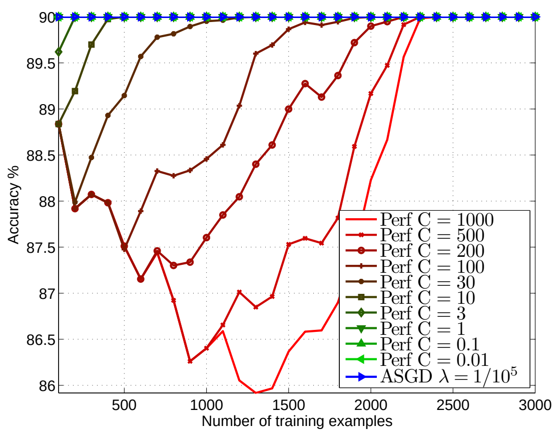


(b) $n = 200, \eta = 0.1$

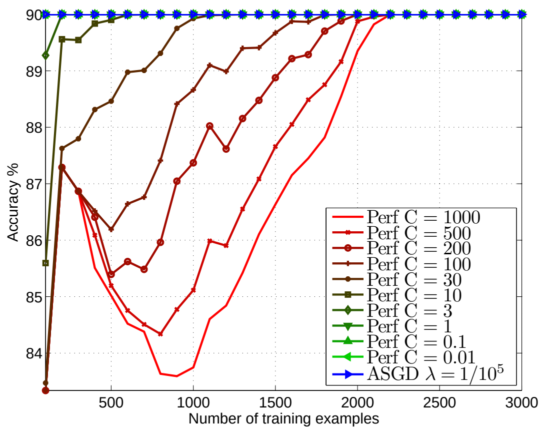


(c) $n = 400$, $\eta = 0.1$

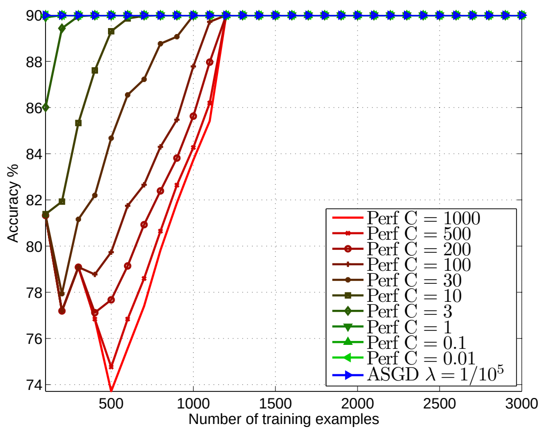
Figure 10: Random-8 accuracies as a function of the training set size for SVM$^{perf}$with a variety of $C$ values and ASGD - vary the dimensionality. Note: The curves for small $C$ (i.e. $C \leqslant 1$) are covered by the curve for ASGD.



(a) $n = 200$, $\eta = 0.03$



(b) $n = 200$, $\eta = 0.05$



(c) $n = 200$, $\eta = 0.15$

Figure 11: Random-8 accuracies as a function of the training set size for SVM$^{perf}$with a variety of $C$ values and ASGD - vary the noise rate. Note: The curves for small $C$ (i.e. $C \leqslant 1$) are covered by the curve for ASGD.

## B RCV1-V2 ASGD Result

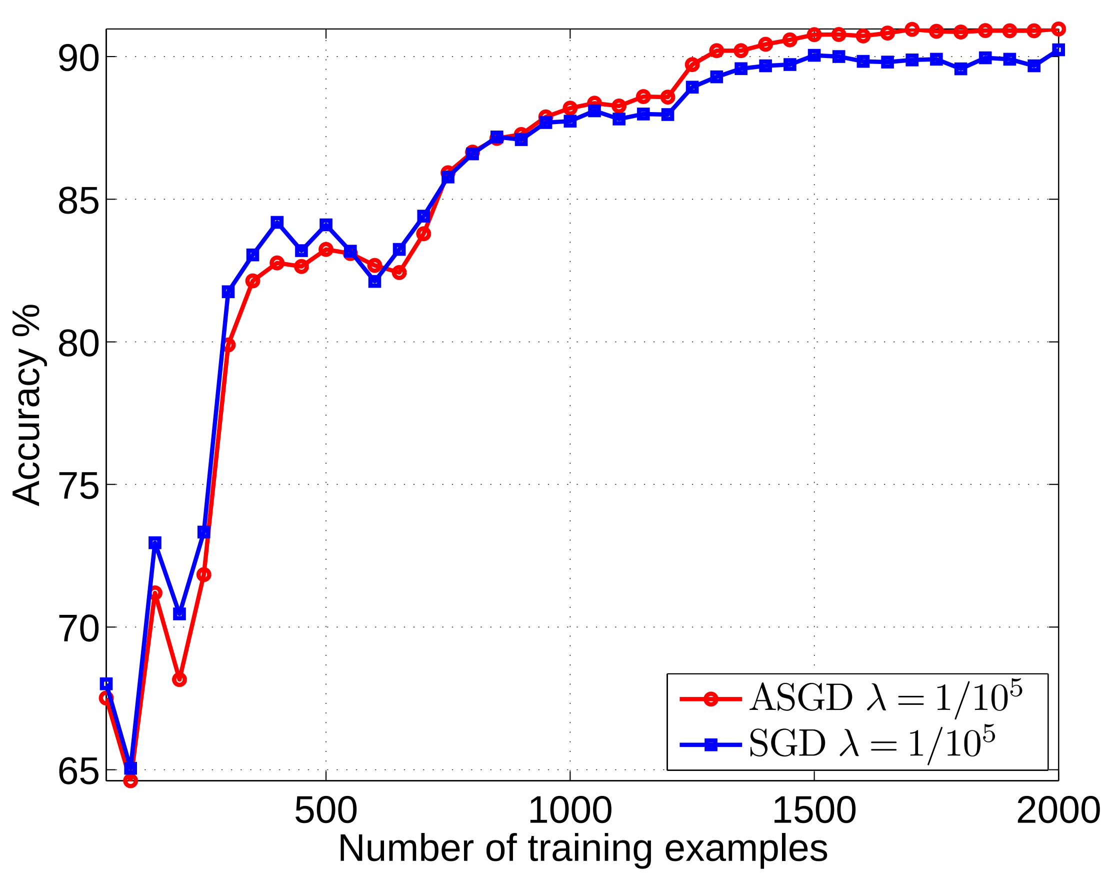


Figure 12: Results from RCV1-V2 data. ASGD vs SGD curves.