Learning on the Border: Active Learning in Imbalanced Data Classification

Şeyda Ertekin[1], Jian Huang[2], Léon Bottou[3], C. Lee Giles[2,1]

[1]Department of Computer Science and Engineering
[2]College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802, USA
[3]NEC Laboratories America
4 Independence Way, Princeton, NJ 08540, USA
sertekin@cse.psu.edu, {jhuang, giles}@ist.psu.edu, leon@bottou.org

ABSTRACT

This paper is concerned with the class imbalance problem which has been known to hinder the learning performance of classification algorithms. The problem occurs when there are significantly less number of observations of the target concept. Various real-world classification tasks, such as medical diagnosis, text categorization and fraud detection suffer from this phenomenon. The standard machine learning algorithms yield better prediction performance with balanced datasets. In this paper, we demonstrate that active learning is capable of solving the class imbalance problem by providing the learner more balanced classes. We also propose an efficient way of selecting informative instances from a smaller pool of samples for active learning which does not necessitate a search through the entire dataset. The proposed method yields an efficient querying system and allows active learning to be applied to very large datasets. Our experimental results show that with an early stopping criteria, active learning achieves a fast solution with competitive prediction performance in imbalanced data classification.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning, induction*

General Terms

Algorithms, experimentation

Keywords

Active learning, imbalanced data, support vector machines

1. INTRODUCTION

Classification is a supervised learning method which acquires a training dataset to form its model for classifying unseen examples. A training dataset is called imbalanced if at least one of the classes are represented by significantly less number of instances (i.e. observations, examples, cases) than the others. Real world applications often face this problem because naturally normal examples which constitute the majority class in classification problems are generally abundant; on the other hand the examples of interest are generally rare and form the minority class. Another reason for class imbalance problem is the limitations (e.g., cost, difficulty or privacy) on collecting instances of some classes. Examples of applications which may have class imbalance problem include, but are not limited to, predicting pre-term births [8], identifying fraudulent credit card transactions [4], text categorization [7], classification of protein databases [19] and detecting certain objects from satellite images [13]. Despite that they are difficult to identify, rare instances generally constitute the target concept in classification tasks. However, in imbalanced data classification, the class boundary learned by standard machine learning algorithms can be severely skewed toward the target class. As a result, the false-negative rate can be excessively high.

In classification tasks, it is generally more important to correctly classify the minority class instances. In real-world applications mispredicting a rare event can result in more serious consequences than mispredicting a common event. For example in the case of cancerous cell detection, misclassifying non-cancerous cells leads to additional clinical testing but misclassifying cancerous cells leads to very serious health risks. Similar problem might occur in detection of a threatening surveillance event from video streams, where misclassifying a normal event may only result in increased security but misclassifying a life threatening event may lead to disastrous consequences. However in classification problems with imbalanced data, the minority class examples are more likely to be misclassified than the majority class examples. Due to their design principles, most of the machine learning algorithms optimize the overall classification accuracy hence sacrifice the prediction performance on the minority classes. This paper proposes an efficient active learning framework which has high prediction performance to overcome this serious data mining problem.

In addition to the naturally occurring class imbalance problem, the imbalanced data situation may also occur in one-against-rest schema in multiclass classification. Assuming there are N different classes, one of the simplest multiclass classification schemes built on top of binary classifiers is to train N different binary classifiers. Each classifier is trained to distinguish the examples in a single class from the examples in all remaining classes. When it is desired to classify a new example, the N classifiers are run, and the

CIKM'07, November 6–8, 2007, Lisboa, Portugal.

classifier which has the highest classification confidence is chosen. Therefore, even though the training data is balanced, issues related to the class imbalance problem can frequently surface.

In this paper we propose an alternative to the existing methods: using active learning strategy to deal with the class imbalance problem. Active learning has been pronounced by some researchers [18, 1] as a sampling method but no systematic study has been done to show that it works well with imbalanced data. We demonstrate that by selecting informative instances for training, active learning can indeed be a useful technique to address the class imbalance problem. We constrain our discussion to a standard two-class classification problem with Support Vector Machines (SVMs). In the rest of the paper, we refer to the minority and majority classes as ”positive” and ”negative” respectively.

In this paper, we propose an efficient SVM based active learning selection strategy which queries small pool of data at each iterative step instead of querying the entire dataset. The proposed method brings the advantage of efficient querying in search of the most informative instances, thus enabling active learning strategy to be applied to large datasets without high computational costs. Rather than using a traditional batch SVM, we use an *online* SVM algorithm [3] which suits better to the nature of active learning due to its incremental learning steps. We present that active learning's querying strategy yields a balanced training set in the early stages of the learning without any requirement of preprocessing of the data. Major research direction in recent literature to overcome the class imbalance problem is to resample the original training dataset to create more balanced classes. This is done either by oversampling the minority class and/or undersampling the majority class until the classes are approximately equally represented. Our empirical results show that active learning can be a more efficient alternative to resampling methods in creating balanced training set for the learner. AL does not risk losing information as in undersampling and does not bring an extra burden of data as in oversampling. With early stopping, active learning can achieve faster and scalable solution without sacrificing prediction performance.

2. RELATED WORK

Recent research on class imbalance problem has focused on several major groups of techniques. One is to assign distinct costs to the classification errors [6, 17]. In this method, the misclassification penalty for the positive class is assigned a higher value than that of the negative class. This method requires tuning to come up with good penalty parameters for the misclassified examples. The second is to resample the original training dataset, either by over-sampling the minority class and/or under-sampling the majority class until the classes are approximately equally represented [5, 11, 14, 15]. Both resampling methods introduce additional computational costs of data preprocessing and oversampling can be overwhelming in the case of very large scale training data. Undersampling has been proposed as a good means of increasing the sensitivity of a classifier. However this method may discard potentially useful data that could be important for the learning process therefore significant decrease in the prediction performance may be observed. Discarding the redundant examples in undersampling has been discussed in [16] but since it is an adaptive method for ensemble learning and does not involve an external preprocessing step it can not be applied to other types of algorithms. Oversampling has been proposed to create synthetic positive instances from the existing positive samples to increase the representation of the class. Nevertheless, oversampling may suffer from overfitting and due to the increase in the number of samples, the training time of the learning process gets longer. If a complex oversampling method is used, it also suffers from high computational costs during preprocessing data. In addition to those, oversampling methods demand more memory space for the storage of newly created instances and the data structures based on the learning algorithm (i.e., extended kernel matrix in kernel classification algorithms). Deciding on the oversampling and undersampling rate is also another issue of those methods. Another technique suggested for class imbalance problem is to use a recognition-based, instead of discrimination-based inductive learning [10, 20]. These methods attempt to measure the amount of similarity between a query object and the target class, where classification is accomplished by imposing a threshold on the similarity measure. The major drawback of those methods is the need for tuning the similarity threshold of which the success of the method mostly relies on. On the other hand, discrimination-based learning algorithms have been proved to give better prediction performance in most domains.

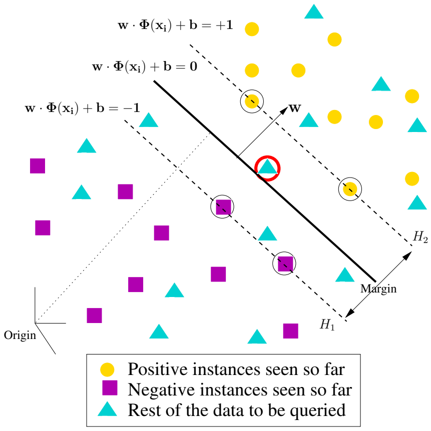

$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x_i}) + \mathbf{b} = +\mathbf{1}$
$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x_i}) + \mathbf{b} = \mathbf{0}$
$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x_i}) + \mathbf{b} = -\mathbf{1}$
w
H_2
Margin
H_1
Origin
● Positive instances seen so far
■ Negative instances seen so far
▲ Rest of the data to be queried

Figure 1: Active Learning with SVM (separable case). The most informative sample among the unseen training samples is the one (in bold circle) closest to the hyperplane (solid line). The circled samples on the dashed lines are support vectors.

In [2] the behavior of Support Vector Machines (SVM) with imbalanced data is investigated. They applied [5]'s SMOTE algorithm to oversample the data and trained SVM with different error costs. SMOTE is an oversampling approach in which the minority class is oversampled by creating synthetic examples rather than with replacement. The k nearest positive neighbors of all positive instances are identified and synthetic positive examples are created and placed randomly along the line segments joining the k minority class nearest neighbors. Preprocessing the data with SMOTE may lead to improved prediction performance at the classifiers, however it also brings more computational cost to the system for preprocessing and yet the increased number of training data makes the SVM training very costly since the training time at SVMs scales quadratically with the number of training instances. In order to cope with today's tremendously growing dataset sizes, we believe that there is a need for more computationally efficient and scalable algorithms. We show that such a solution can be achieved by using active learning strategy.

3. METHODOLOGY

Active learning is mostly regarded as a technique that addresses the unlabeled training instance problem. The learner has access to a

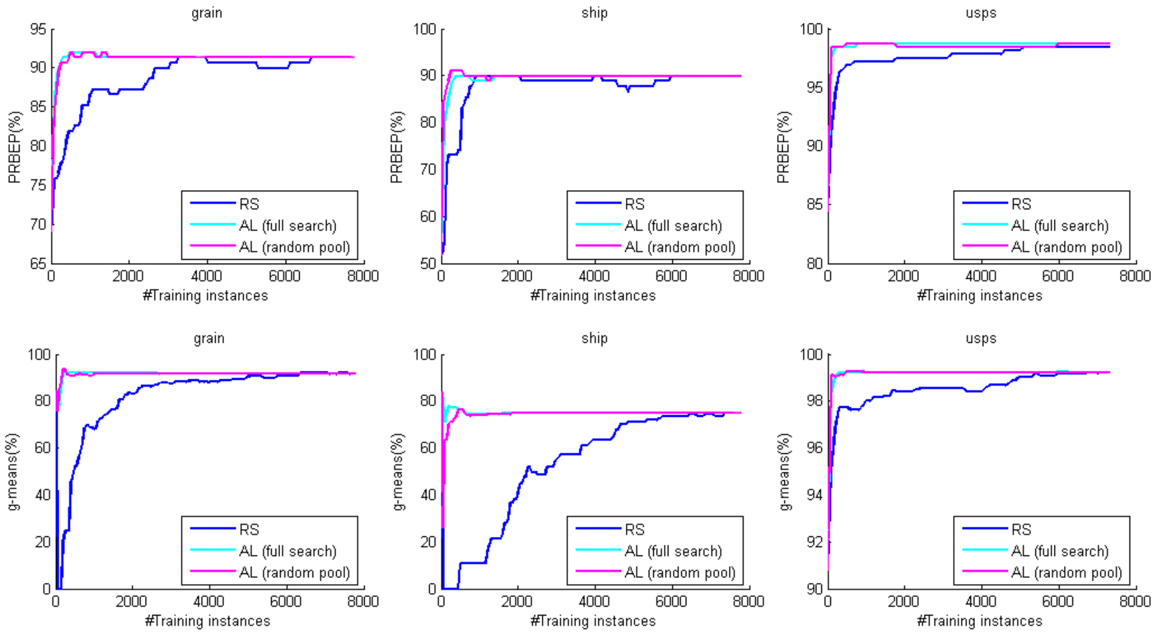


Figure 2: Comparison of PRBEP and g-means of RS, AL(full search) and AL(random pool). The training times of AL(full search) vs. AL(random pool) until saturation in seconds are: 272 vs. 50 (grain), 142 vs. 32 (ship) and 126 vs. 13 (USPS). AL(random pool) is 4 to 10 times faster than AL(full search) with similar prediction performance.

vast pool of unlabeled examples, and it tries to make a clever choice to select the most informative example to obtain its label. However, in the cases where all the labels are available beforehand, active learning can still be leveraged to obtain the informative instances through the training sets [21, 3, 9]. In SVMs, *informativeness* of an instance is synonymous with its distance to the hyperplane. The farther an instance is to the hyperplane, the more the learner is confident about its true class label, hence it does not bring much (or any) information to the system. On the other hand, the instances close to the hyperplane are informative for learning. SVM based active learning can pick up the informative instances by checking their distances to the hyperplane. The closest instances to the hyperplane are considered to be the most informative instances.

The strategy of selecting instances within the margin addresses the imbalanced dataset classification very well. Suppose that the class distributions of an imbalanced dataset is given in Figure 3. The shaded region corresponds to the class distribution of the data within the margin. As it can be observed, the imbalance ratio of the classes within the margin is much smaller than the class imbalance ratio of the entire dataset. Any selection strategy which focuses on the instances in the margin most likely ends up with a more balanced class distribution than that of the entire dataset. Our empirical findings with various type of real-world data confirm that the imbalance ratios of the classes within the margin in real-world data are generally much lower than that of the entire data as shown in Figure 3.

A brief explanation of the SVMs is given in Section 3.1 followed by the working principles of the efficient active learning algorithm in Section 3.2. We explain the advantage of using online SVMs with the active sample selection in Section 3.3. In Section 3.4, we then describe an early stopping heuristics for active learning.

3.1 Support Vector Machines

Support Vector Machines [24] are well known for their strong theoretical foundations, generalization performance and ability to handle high dimensional data. In the binary classification setting, let $((x_1, y_1) \cdots (x_n, y_n))$ be the training dataset where x_i are the feature vectors representing the instances and $y_i \in (-1, +1)$ be the labels of the instances. Using the training set, SVM builds an optimum hyperplane – a linear discriminant in a higher dimensional feature space – that separates the two classes by the largest margin (see Figure 1). This hyperplane can be obtained by minimizing the following objective function:

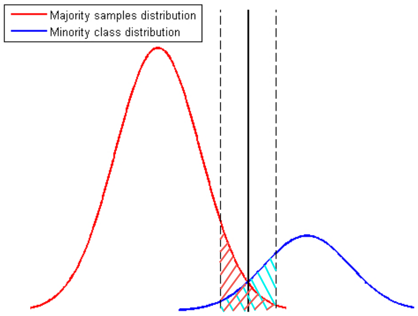


Figure 3: Data within the margin is less imbalanced than the entire data.

$$\min_{\mathbf{w},\mathbf{b},\xi_i} \frac{1}{2}\mathbf{w}\cdot\mathbf{w}^T + C\sum_{i=1}^{N}\xi_i \tag{1}$$

$$\text{subject to} \begin{cases} \forall i \; y_i(\mathbf{w}^T\Phi(x_i) - b) \geq 1 - \xi_i \\ \forall i \; \xi_i \geq 0 \end{cases} \tag{2}$$

where $\mathbf{w}$ is the norm of the hyperplane, b is the offset, y_i are the labels, $\Phi(\cdot)$ is the mapping from input space to feature space,

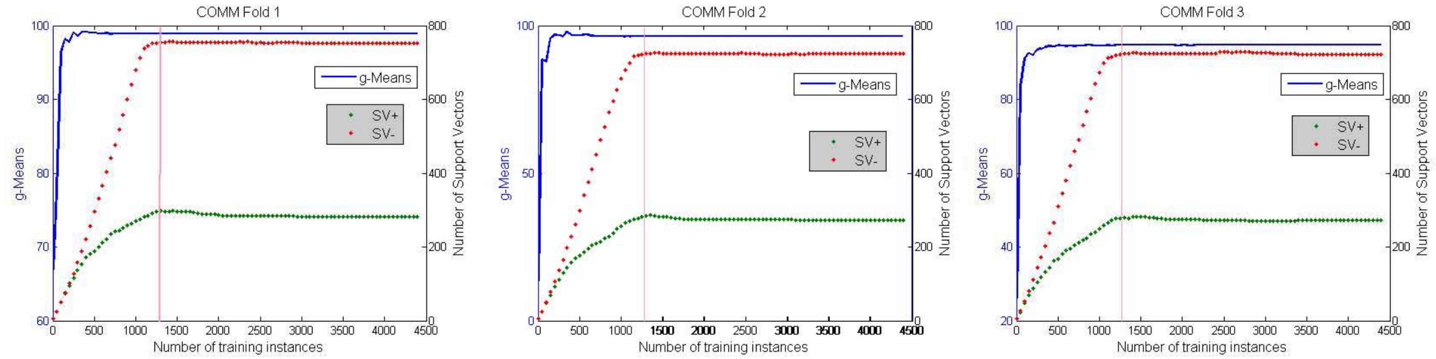


Figure 4: 3-fold cross-validation results for the training set of the category COMM in CiteSeer dataset. Vertical lines correspond to early stopping points.

and ξ_i are the slack variables that permit the non-separable case by allowing misclassification of training instances. In practice the convex quadratic programming (QP) problem in Equation 1 is solved by optimizing the dual cost function. The dual representation of Equation 1 is given as

$$\max W(\alpha) \equiv \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j}) \qquad (3)$$

$$\text{subject to} \begin{cases} \forall i \; 0 \leq \alpha_i \leq C \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \end{cases} \qquad (4)$$

where y_i are the labels, $\Phi(\cdot)$ is the mapping from the input space to the feature space, $K(\mathbf{x_i}, \mathbf{x_j}) = \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$ is the kernel matrix and the α_i's are the *Lagrange multipliers* which are non-zero only for the training instances which fall in the margin. Those training instances are called *support vectors* and they define the position of the hyperplane. After solving the QP problem, the norm of the hyperplane **w** can be represented as

$$w = \sum_{i=1}^{n} \alpha_i \Phi(x_i) \qquad (5)$$

3.2 Active Learning

Note that in equation 5, only the support vectors have an effect on the SVM solution. This means that if SVM is retrained with a new set of data which only consist of those support vectors, the learner will end up finding the same hyperplane. This fact leads us to the idea that not all the instances are equally important in the training sets. Then the question is how to select the most informative examples in the datasets. In this paper we will focus on a form of selection strategy called SVM based active learning. In SVMs, the most informative instance is believed to be the closest instance to the hyperplane since it divides the *version space* into two equal parts. The aim is to reduce the version space as fast as possible to reach the solution faster in order to avoid certain *costs* associated with the problem. For the possibility of a non-symmetric version space, there are more complex selection methods suggested by [23], but it has been observed that the advantage of those are not significant when compared to their high computational costs.

Active Learning with Small Pools: The basic working principle of SVM active learning is: i) learn an SVM on the existing training data, ii) select the closest instance to the hyperplane, and iii) add the new selected instance to the training set and train again. In classical active learning [23], the search for the most informative instance is performed over the entire dataset. Note that, each iteration of active learning involves the recomputation of each training example's distance to the new hyperplane. Therefore, for large datasets, searching the entire training set is a very time consuming and computationally expensive task. We believe that we do not have to search the entire set at each iteration.

By using the "59 trick" [22], we propose a selection method, which does not necessitate a full search through the entire dataset but locates an approximate most informative sample by examining a small constant number of randomly chosen samples. The method picks L (L ≪ # training instances) random training samples in each iteration and selects the best (closest to the hyperplane) among them. Suppose, instead of picking the closest instance among all the training samples $X_N = (x_1, x_2, \cdots, x_N)$ at each iteration, we first pick a random subset X_L, $L \ll N$ and select the closest sample x_i from X_L based on the condition that x_i is among the top $p\%$ closest instances in X_N with probability $(1 - \eta)$. Any numerical modification to these constraints can be met by varying the size of L, and is independent of N. To demonstrate, the probability that at least one of the L instances is among the closest $p\%$ is $1-(1-p\%)^L$. Due to the requirement of $(1-\eta)$ probability, we have

$$1 - (1 - p\%)^L = 1 - \eta \qquad (6)$$

which follows the solution of L in terms of η and p

$$L = \log \eta \; / \; \log(1 - p\%) \qquad (7)$$

For example, the active learner will pick one instance, with 95% probability, that is among the top 5% closest instances to the hyperplane, by randomly sampling only $\lceil \log(.05)/\log(.95) \rceil = 59$ instances regardless of the training set size. This approach scales well since the size of the subset L is independent of the training set size N, requires significantly less training time and does not have an adverse effect on the classification performance of the learner.

In our experiments, we set $L = 59$ which means we pick 59 random instances to form the query pool at each learning step and pick the closest instance to the hyperplane from this pool. Figure 2 shows the comparisons of PRBEP and g-means performances of the proposed method AL(random pool) and the traditional active learning method AL(full search) [23]. RS corresponds to random sampling where instances are selected randomly. As Figure 2 depicts, the proposed active learning method with small pools achieves as good prediction performance as the traditional active

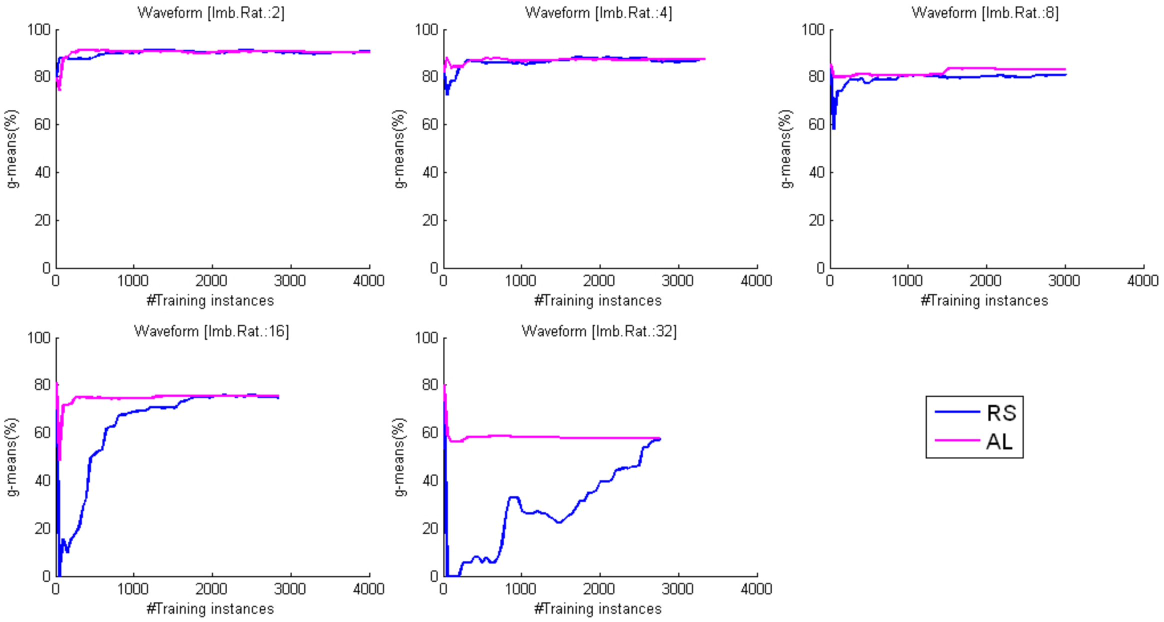


Figure 5: Comparison of g-means of AL and RS on the waveform datasets with different imbalance ratios (Imb.R.=2, 4, 8, 16, 32).

learning method. Moreover, the proposed strategy is 4 to 10 times faster than the traditional active learning for the given datasets.

3.3 Online SVM for Active Learning

Online learning algorithms are usually associated with problems where the complete training set is not available. However, in cases where the complete training set is available, their computational properties can be leveraged for faster classification and incremental learning. In our framework, we use an online SVM algorithm, LASVM [3] instead of a traditional batch SVM tool (e.g., libsvm, SVM^{light}). LASVM is an online kernel classifier which relies on the traditional soft margin SVM formulation. LASVM yields the classification accuracy rates of the state-of-the art traditional SVM solvers but requires less computational resources. Traditional SVM works in a batch setting where all the training instances are used to form the one and final model. LASVM, on the other hand, works in an online setting, where its model is continually modified as it processes training instances one by one. Each LASVM iteration receives a fresh training example and tries to optimize the dual cost function in Equation (3) using feasible direction searches.

Online learning algorithms can select the new data to process either by random or active selection.They can integrate the information of the new seen data to the system without training all the samples again, hence they can incrementally build a learner. This working principle of LASVM leads to speed improvement and less memory demand which makes the algorithm applicable to very large datasets. More importantly, this incremental working principle suits the nature of active learning in a much better way than the batch algorithms. The new informative instance selected by active learning can be integrated to the existing model without retraining all the samples repeatedly. Empirical evidence indicates that a single presentation of each training example to the algorithm is sufficient to achieve training errors comparable to those achieved by the SVM solution [3]. In section 3.4 we also show that if we use an early stopping criteria in active sample selection, we do not have to introduce all the training instances to the learner.

3.4 Active Learning with Early Stopping

Early stopping criteria is advantageous to the active learning method since it converges to the solution faster than the random sample selection method. A theoretically sound method to stop training is when the examples in the margin are exhausted. To check if there are still unseen training instances in the margin, the distance of the new selected instance is compared to the support vectors of the current model. If the new selected instance by active learning (closest to the hyperplane) is not closer than any of the support vectors, we conclude that the margin is exhausted. A practical implementation of this idea is to count the number of support vectors during the active learning training process. If the number of the support vectors stabilizes, it implies that all possible support vectors have been selected by the active learning method.

In order to analyze this method, we conducted a 3-fold cross-validation on one of the datasets (see Figure 4). In cross-validation, $2/3$ of the training set is used for training and the remaining $1/3$ is reserved as the hold-out dataset. Since the training set distribution is representative of the test set distribution, we believe that the algorithm's behavior would most likely be the same in the test set. As can be seen in Figure 4, in active learning setups, after using certain number of labeled training data, the number of support vectors saturates and g-means levels off as well. Those graphs support the idea that the model does not change after the system observes enough informative samples. Further, adding more training data after this point does not make a remarkable change in the model and consequently in prediction performance. Notice that in Figure 4 the vertical line indicates the suggested early stopping point and it is approximately equal in all three folds. As a result, we adopt the early stopping strategy of examining the number of support vectors in the entire training datasets without performing cross-validation.

4. PERFORMANCE METRICS

Classification accuracy is not a good metric to evaluate classifiers in applications with class imbalance problem. SVMs have to

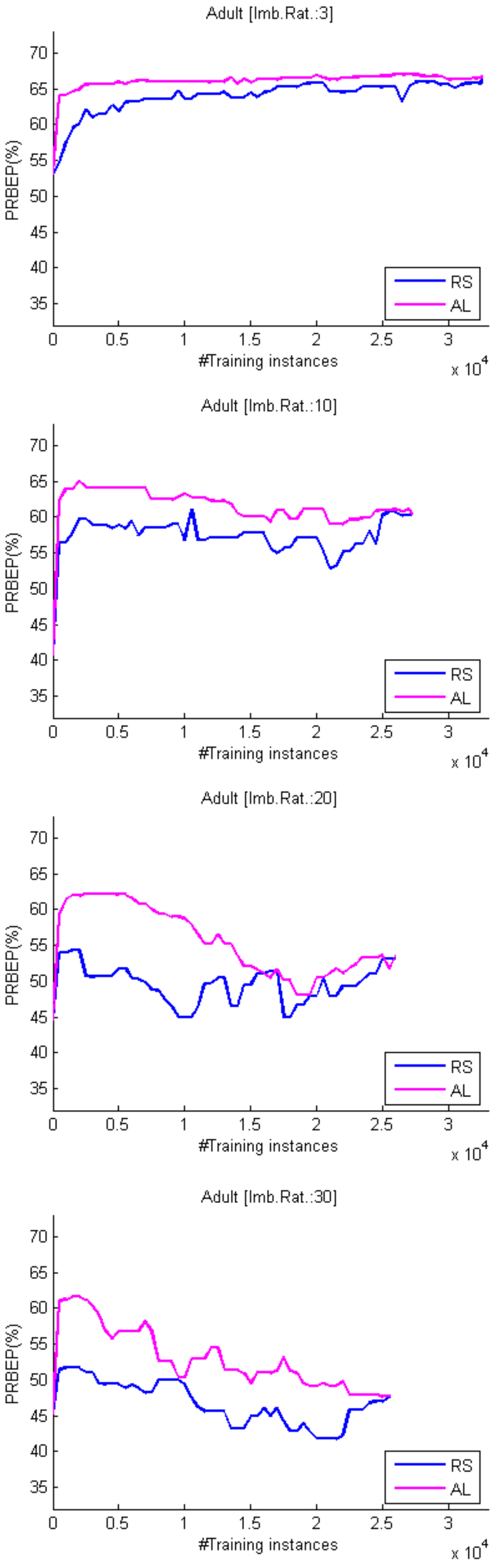


Figure 6: Comparison of PRBEP of AL and RS on the adult datasets with different imbalance ratios (Imb.R.=3, 10, 20, 30).

achieve a tradeoff between maximizing the margin and minimizing the empirical error. In the non-separable case, if the misclassification penalty C is very small, SVM learner simply tends to classify every example as negative. This extreme approach makes the *margin* the largest while making no classification errors on the negative instances. The only error is the cumulative error of the positive instances which are already few in numbers. Considering an imbalance ratio of 99 to 1, a classifier that classifies everything as negative, will be 99% accurate but it will not have any practical use as it can not identify the positive instances.

For evaluation of our results, we use several other prediction performance metrics such as g-means, AUC and PRBEP which are commonly used in imbalanced data classification. g-means [14] is denoted as $g = \sqrt{sensitivity \cdot specifity}$ where sensitivity is the accuracy on the positive instances given as $TruePos./(TruePos. + FalseNeg.)$ and specificity is the accuracy on the negative instances given as $TrueNeg./(TrueNeg. + FalsePos.)$.

The Receiver Operating Curve (ROC) displays the relationship between sensitivity and specificity at all possible thresholds for a binary classification scoring model, when applied to independent test data. In other words, ROC curve is a plot of the true positive rate against the false positive rate as the decision threshold is changed. The *area under the ROC curve* (AUC) is a numerical measure of a model's discrimination performance and shows how successfully and correctly the model separates the positive and negative observations and ranks them. Since AUC metric evaluates the classifier across the entire range of decision thresholds, it gives a good overview about the performance when the operating condition for the classifier is unknown or the classifier is expected to be used in situations with significantly different class distributions.

Precision Recall Break-Even Point (PRBEP) is another commonly used performance metric for imbalanced data classification. PRBEP is the accuracy of the positive class at the threshold where precision equals to recall. Precision is defined as $TruePos./(TruePos. + FalsePos.)$ and recall is defined as $TruePos./(TruePos. + FalseNeg.)$

5. DATASETS

We study the performance of the algorithm on various benchmark real-world datasets. The overview of the datasets are given in Table 2. The *Reuters-21578* is a popular text mining benchmark dataset. We test the algorithms with 8 of the top 10 most populated categories of *Reuters-21578*. We did not use categories 'earn' and 'acq' since their class imbalance ratios are not high enough. As a text dataset, we also used 5 categories from CiteSeer[1] data. We used 4 benchmark datasets from the popular UCI Machine Learning Repository as well. *Letter* and *satimage* are image datasets. The 'letter A' is used as the positive class in *letter* and 'class 4' (damp grey soil) is used as positive class in *satimage*. *Abalone* is a biology dataset. respectively. In *abalone*, instances labeled as 'class 7' are used to form the positive class. *MNIST* and *USPS* are OCR data of handwritten digits and 'digit 8' is used as a positive class in *Mnist*. *Adult* is a census dataset to predict if the income of a person is greater than 50K based on several census parameters, such as age, education, marital status etc. The training set consists of 32,562 instances and the class imbalance ratio is 3. *Waveform* is a popular artificial dataset used commonly in simulation studies. These datasets cover a wide range of data imbalance ratio.

[1] http://citeseer.ist.psu.edu

Table 1: Comparison of g-means and AUC for AL and RS with entire training data (Batch). Support vector ratios are given at the saturation point. Data efficiency corresponds to the percentage of training instances which AL processes to reach saturation.

Dataset		g-means (%)		AUC (%)		Imb.	SV- /	Data
		Batch	AL	Batch	AL	Rat.	SV+	Efficiency
Reuters	Corn	85.55	86.59	99.95	99.95	41.9	3.13	11.6%
	Crude	88.34	89.51	99.74	99.74	19.0	2.64	22.6%
	Grain	91.56	91.56	99.91	99.91	16.9	3.08	29.6%
	Interest	78.45	78.46	99.01	99.04	21.4	2.19	30.9%
	Money-fx	81.43	82.79	98.69	98.71	13.4	2.19	18.7%
	Ship	75.66	74.92	99.79	99.80	38.4	4.28	20.6%
	Trade	82.52	82.52	99.23	99.26	20.1	2.22	15.4%
	Wheat	89.54	89.55	99.64	99.69	35.7	3.38	11.6%
CiteSeer	AI	87.83	88.58	94.82	94.69	4.3	1.85	33.4%
	COMM	93.02	93.65	98.13	98.18	4.2	2.47	21.3%
	CRYPT	98.75	98.87	99.95	99.95	11.0	2.58	15.2%
	DB	92.39	92.39	98.28	98.46	7.1	2.50	18.2%
	OS	91.95	92.03	98.27	98.20	24.2	3.52	36.1%
UCI	Abalone-7	100.0	100.0	100.0	100.0	9.7	1.38	24.0%
	Letter-A	99.28	99.54	99.99	99.99	24.4	1.46	27.8%
	Satimage	82.41	83.30	95.13	95.75	9.7	2.62	41.7%
USPS		99.22	99.25	99.98	99.98	4.9	1.50	6.8%
MNIST-8		98.47	98.37	99.97	99.97	9.3	1.59	11.7%

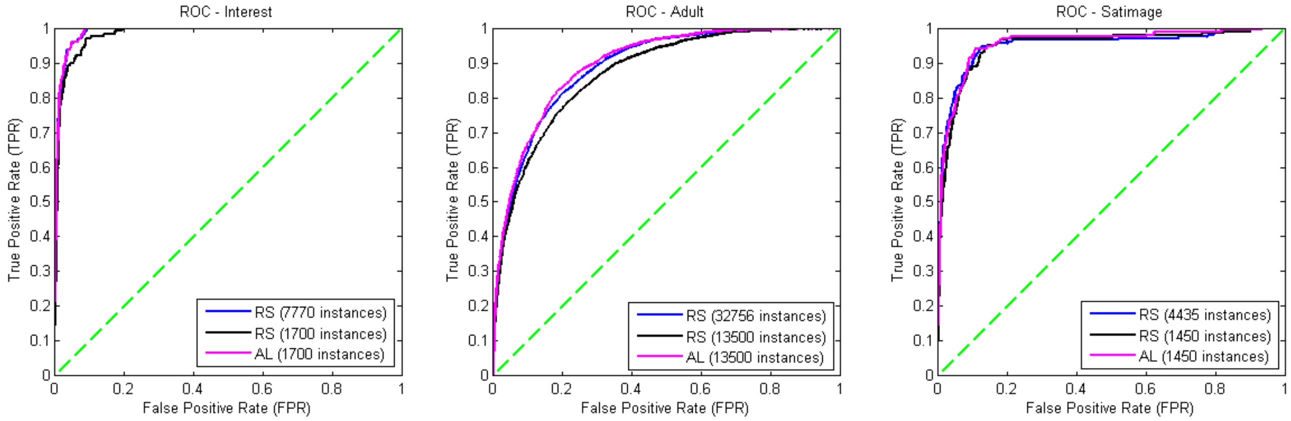


Figure 7: Comparison of ROC curves of AL, RS (early stopped at the same number of instances as AL) and RS (with all training data) in Interest, Adult and Satimage datasets.

6. EXPERIMENTS AND EMPIRICAL EVALUATION

We first conduct experiments to compare the performance of the proposed active learning strategy AL(random pool) with the traditional active learning method, AL(full search). The results show that with the proposed method, we can make faster active learning without sacrificing any prediction performance (see Figure 2). In the rest of the paper, we refer to our proposed method as AL since it is the only active learning method that we used afterwards.

In order to make a thorough analysis on the effect of AL to imbalanced data classification, we examine its performance by varying class imbalance ratios using two performance metrics. We randomly remove the instances from the minority class in *Waveform* and *Adult* datasets to achieve different data imbalance ratios. Comparisons of g-means of AL and RS in Figure 5 show that the prediction performance of AL is less sensitive to the class imbalance ratio changes than that of the RS. Comparisons of another performance metric PRBEP in Figure 6 give even more interesting results. As the class imbalance ratio is increased, AL curves display peaks in the early steps of the learning. This implies that by using an early stopping criteria AL can give higher prediction performance than RS can possibly achieve even after using all the training data. Figure 6 curves allow us to think that addition of any instances to the learning model after finding the informative instances can be detrimental to the prediction performance of the classifier. This finding strengthens the idea of applying an early stopping to the active learning algorithms.

We also compared the performance of early stopped AL with Batch algorithm. Table 1 presents the g-means and the AUC values of the two methods. Data efficiency column for AL indicates that by processing only a portion of the instances from the training set, AL can achieve similar or even higher prediction performance than that of Batch which sees all the training instances. Another important observation from Table 1 is that support vector imbalance ratios in the final models are much less than the class imbalance ratios of the datasets. This confirms our discussion of Figure 3 in section 3. The class imbalance ratio within the margins are much less than the class imbalance ratio of the entire data and active learning can be used to reach those informative instances which most likely become support vectors without seeing all the training instances.

In order to evaluate the methods at different thresholds, we also investigate the ROC curves as given in Figure 7. The ROC curves of AL are similar and sometimes better than of the Batch algorithm (RS, seeing all the training instances). The AUC of AL and Batch are 0.8980 and 0.8910 respectively in the *Adult* dataset. At the same number of training instances where AL is early stopped, AUC of RS can be substantially lower. As Figure 7 shows, the ROC curve of AL is markedly higher than that of RS (early stopping) and the AUC values are 0.8980 and 0.8725 respectively for *Adult* dataset. These results suggest that AL converges faster than RS using fewer and informative instances and AL can get even higher prediction performance than the Batch algorithm by processing only a portion of the training set.

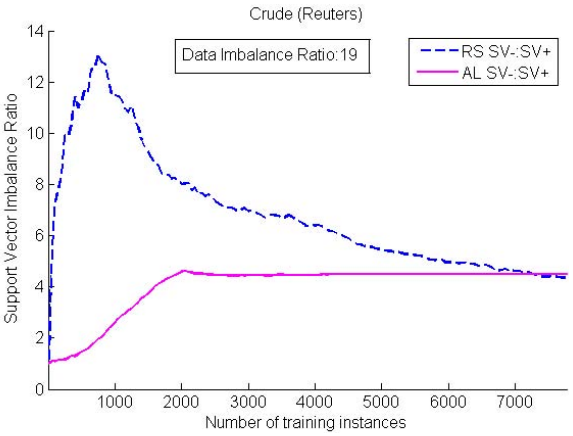


Figure 8: Support Vector ratios in AL and RS

In Figure 8, we investigate how the number of support vectors changes in AL and Random Sampling (RS). With random sampling, the instances are selected for the learner randomly from the entire pool of the training data. Therefore, the support vector imbalance ratio quickly approaches the data imbalance ratio. As learning continues, the learner should gradually see all the instances within the final margin and the support vector imbalance ratio decreases. When RS finishes learning, the support vector imbalance ratio is the data imbalance ratio within the margin. The support vector imbalance ratio curve of AL is drastically different than RS. AL intelligently picks the instances closest to the margin in each step. Since the data imbalance ratio within the margin is lower than data imbalance ratio, the support vectors in AL are more balanced than RS during learning. Using AL, the model saturates by seeing only 2000 (among 7770) training instances and reaches the final support vector imbalance ratio. Note that both methods achieve similar support vector imbalance ratios when learning finishes, but AL achieves this in the early steps of the learning.

We compare the AL method discussed in this paper with several other strategies as well. Among them, undersampling (US), and an oversampling method (SMOTE) are examples of resampling techniques which require preprocessing. Recent research showed that oversampling at random does not help to improve prediction performance [12] therefore we use a more complex oversampling method (SMOTE). As an algorithmic method to compare, we use the method of assigning different costs (DC) to the positive and negative classes as the misclassification penalty parameter. For instance, if the imbalance ratio of the data is 19:1 in favor of the negative class, the cost of misclassifying a positive instance is set to be 19 times greater than that of misclassifying a negative one. We use LASVM[2], an online SVM tool, in all experiments. Other than the results of the methods addressing class imbalance problem, we also give results of Batch algorithm with the original training set to form a baseline. LASVM is run in random sampling mode for US, SMOTE and DC.

Table 2: Overview of the datasets.

Dataset		#Feat.	#Pos	#Neg	Ratio	c	γ
Reuters	Crude	8315	389	7381	19.0	2	1
	Grain	8315	433	7337	16.9	2	1
	Interest	8315	347	7423	21.4	1	2
	Money-fx	8315	538	7232	13.4	1	0.5
	Ship	8315	197	7573	38.4	1	0.5
	Wheat	8315	212	7558	35.7	1	0.5
CiteSeer	AI	6946	1420	5353	4.3	50	0.1
	COMM	6946	1252	5341	4.2	50	0.1
	Crypt	6946	552	6041	11.0	50	0.1
	DB	6946	819	5775	7.1	50	0.1
	OS	6946	262	6331	24.2	50	0.1
UCI	Abalone-7	9	352	3407	9.7	100	0.01
	Letter-A	16	710	17290	24.4	10	0.01
	Satimage	36	415	4020	9.69	50	0.001
USPS		256	1232	6097	5.0	1000	2
MNIST-8		780	5851	54149	9.3	1000	0.02

We give the comparisons of the methods for g-means performance metric for several datasets in Figure 9. The right border of the shaded pink area is the place where the aforementioned early stopping strategy is applied. The curves in the graphs are averages of 10 runs. For completeness we did not stop the AL experiments at the early stopping point but allow them to run on the entire training set. We present the PRBEP of the methods and the total running times of the SMOTE and AL on 18 benchmark and real-world datasets in Table 3. The results for active learning in Table 3 depict the results in the early stopping points. The results for the other methods in Table 3 depict the values at the end of the curves –when trained with the entire dataset– since those methods do not employ any early stopping criteria. We did not apply early stopping criteria to the other methods because as observed from Figure 9, no early stopping criteria would achieve a comparable training time with of AL's training time without a significant loss in their prediction performance based on convergence time. The other methods converge to similar levels of g-means when nearly all training instances are used, and applying an early stopping criteria would have little, if any, effect on their training times.

Since AL involves discarding some instances from the training set, it can be perceived as a type of undersampling method. Unlike US which discards majority samples randomly, AL performs an intelligent search for the most informative ones adaptively in each iteration according to the current hyperplane. In datasets where class imbalance ratio is high such as *corn*, *wheat*, *letter* and *satimage*, we observe significant decrease in PRBEP of US (see Table 3). Note that US's undersampling rate for the majority class in each category is set to the same value as the final support vector ratio which AL reaches in the early stopping point and RS reaches when it sees the entire training data. Although the class imbalance ratio provided to the learner in AL and US are the same,

[2] Available at http://leon.bottou.org/projects/lasvm

Table 3: Comparison of PRBEP and training time.

Metric		PRBEP					Training time (sec.)	
Dataset		Batch	US	SMOTE	DC	AL	SMOTE	AL
Reuters	Corn	91.07	78.57	91.07	89.28	89.29	87	16
	Crude	87.83	85.70	87.83	87.83	87.83	129	41
	Grain	92.62	89.93	91.44	91.94	91.94	205	50
	Interest	76.33	74.04	77.86	75.57	75.57	116	42
	Money-fx	73.74	74.30	75.42	75.42	76.54	331	35
	Ship	86.52	86.50	88.76	89.89	89.89	49	32
	Trade	77.77	76.92	77.77	77.78	78.63	215	38
	Wheat	84.51	81.61	84.51	84.51	85.92	54	25
CiteSeer	AI	78.80	80.68	78.99	78.79	79.17	1402	125
	COMM	86.59	86.76	86.59	86.59	86.77	1707	75
	CRYPT	97.89	97.47	97.89	97.89	97.89	310	19
	DB	86.36	86.61	86.98	86.36	86.36	526	41
	OS	84.07	83.19	84.07	84.07	84.07	93	23
UCI	Abalone-7	100.0	100.0	100.0	100.0	100.0	16	4
	Letter-A	99.48	96.45	99.24	99.35	99.35	86	3
	Satimage	73.46	68.72	73.46	73.93	73.93	63	21
USPS		98.44	98.44	98.13	98.44	98.75	4328	13
MNIST-8		97.63	97.02	97.74	97.63	97.74	83,339	1,048

AL achieves significantly better PRBEP performance metric than US. The Wilcoxon signed-rank test (2-tailed) reveals that the zero median hypothesis can be rejected at the significance level 1% (p=0.0015), implying that AL performs statistically better than US in these 18 datasets. These results reveal the importance of using the informative instances for learning.

Table 4 presents the rank of PRBEP prediction performance of the five approaches in a variety of datasets. The values in bold correspond to the cases where AL wins and it's clear that winning cases are very frequent for AL (12 out of 18 cases). The average rank also indicates that AL achieves the best PRBEP among the five methods. SMOTE and DC achieve higher PRBEP than the Batch algorithm. The loss of information when undersampling the majority class affects US's prediction performance. Table 3 also gives the comparison of the computation times of the AL and SMOTE. Note that SMOTE requires significantly long preprocessing time which dominates the training time in large datasets, e.g., MNIST-8 dataset. The low computation cost, scalability and high prediction performance of AL suggest that AL can efficiently handle the class imbalance problem.

Table 4: Comparison of ranks of different methods in PRBEP. The values in bold correspond to the cases where AL win. AL wins in 12 out of 18 cases in PRBEP.

Metric		Rank				
Dataset		Batch	US	SMOTE	DC	AL
Reuters	Corn	1	5	1	4	3
	Crude	1	5	1	1	**1**
	Grain	1	5	4	2	2
	Interest	2	5	1	3	3
	Money-fx	5	4	2	2	**1**
	Ship	4	5	3	1	**1**
	Trade	3	5	3	2	**1**
	Wheat	2	5	2	2	**1**
CiteSeer	AI	4	1	3	5	2
	COMM	3	2	3	3	**1**
	CRYPT	1	5	1	1	**1**
	DB	3	2	1	3	3
	OS	1	5	1	1	**1**
UCI	Abalone-7	1	1	1	1	**1**
	Letter-A	1	5	4	2	2
	Satimage	3	5	3	1	**1**
USPS		2	2	5	2	**1**
MNIST-8		3	5	1	3	**1**
Avg. Rank		2.28	4.00	2.22	2.17	**1.50**

7. CONCLUSIONS

The class imbalance problem has been known to impact the prediction performance of classification algorithms. The results of this paper offer a better understanding of the effect of the active learning on imbalanced datasets. We first propose an efficient active learning method which selects informative instances from a randomly picked small pool of examples rather than making a full search in the entire training set. This strategy renders active learning to be applicable to very large datasets which otherwise would be computationally very expensive. Combined with the early stopping heuristics, active learning achieves a fast and scalable solution without sacrificing prediction performance. We then show that the proposed active learning strategy can be used to address the class imbalance problem. In simulation studies, we demonstrate that as the imbalance ratio increases, active learning can achieve better prediction performance than random sampling by only using the informative portion of the training set. By focusing the learning on the instances around the classification boundary, more balanced class distributions can be provided to the learner in the earlier steps of the learning. Our empirical results on a variety of real-world datasets allow us to conclude that active learning is comparable or even better than other popular resampling methods in dealing with imbalanced data classification.

8. REFERENCES

[1] N. Abe. Invited talk: Sampling approaches to learning from imbalanced datasets: Active learning, cost sensitive learning and beyond. *Proc. of ICML Workshop: Learning from Imbalanced Data Sets*, 2003.

[2] R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. *Proc. of European Conference on Machine Learning*, pages 39–50, 2004.

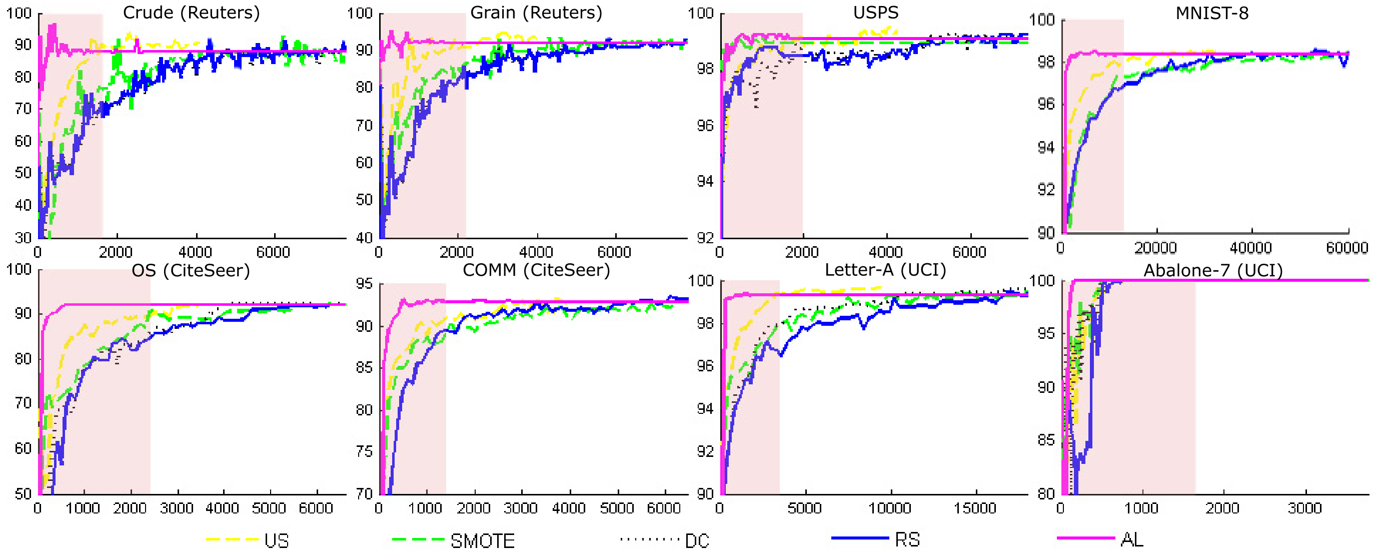


Figure 9: Comparisons of g-means. The right border of the shaded area corresponds to the early stopping point.

[3] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research (JMLR)*, 6:1579–1619, 2005.

[4] P. K. Chan and S. J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 1998.

[5] N. V. Chawla, K. W. Bowyer., L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research (JAIR)*, 16:321–357, 2002.

[6] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proc. of the ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 1999.

[7] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proc. of Int. Conference on Information and Knowledge Management (CIKM)*, 1998.

[8] J. W. Grzymala-Busse, Z. Zheng, L. K. Goodwin, and W. J. Grzymala-Busse. An approach to imbalanced datasets based on changing rule strength. In *Proc. of In Learning from Imbalanced Datasets, AAAI Workshop*, 2000.

[9] J. Huang, S. Ertekin, and C. L. Giles. Efficient name disambiguation for large scale datasets. In *Proc. of European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, 2006.

[10] N. Japkowicz. A novelty detection approach to classification. In *Proc. of the Int. Joint Conference on Artificial Intelligence (IJCAI)*, pages 518–523, 1995.

[11] N. Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of 2000 Int. Conference on Artificial Intelligence (IC-AI'2000)*, volume 1, pages 111–117, 2000.

[12] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 2002.

[13] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2-3):195–215, 1998.

[14] M. Kubat and S. Matwin. Addressing the curse of imbalanced training datasets: One sided selection. *Proc. of Int. Conference on Machine Learning (ICML)*, 30(2-3), 1997.

[15] C. X. Ling and C. Li. Data mining for direct marketing: Problems and solutions. In *Knowledge Discovery and Data Mining*, pages 73–79, 1998.

[16] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory under-sampling for class-imbalance learning. In *Proc. of the International Conference on Data Mining (ICDM)*, 2006.

[17] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk. Reducing misclassification costs. In *Proc. of 11th Int. Conference on Machine Learning (ICML)*, 1994.

[18] F. Provost. Machine learning from imbalanced datasets 101. In *Proc. of AAAI Workshop on Imbalanced Data Sets*, 2000.

[19] P. Radivojac, N. V. Chawla, A. K. Dunker, and Z. Obradovic. Classification and knowledge discovery in protein databases. *Journal of Biomedical Informatics*, 37(4):224–239, 2004.

[20] B. Raskutti and A. Kowalczyk. Extreme re-balancing for svms: a case study. *SIGKDD Explorations Newsletter*, 6(1):60–69, 2004.

[21] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proc. of the 17th Int. Conference on Machine Learning (ICML)*, pages 839–846, 2000.

[22] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. of 17th Int. Conference on Machine Learning (ICML)*.

[23] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research (JMLR)*, 2:45–66, 2002.

[24] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

Journal of Machine Learning Research 6 (2005) 1579–1619 Submitted 3/05; Published 9/05

Fast Kernel Classifiers
with Online and Active Learning

Antoine Bordes ANTOINE.BORDES@BDE.ESPCI.FR
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540, USA, and
Ecole Supérieure de Physique et Chimie Industrielles
10 rue Vauquelin
75231 Paris CEDEX 05, France

Seyda Ertekin SEYDA@PSU.EDU
The Pennsylvania State University
University Park, PA 16802, USA

Jason Weston JASONW@NEC-LABS.COM
Léon Bottou LEON@BOTTOU.ORG
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540, USA

Editor: Nello Cristianini

Abstract

Very high dimensional learning systems become theoretically possible when training examples are abundant. The computing cost then becomes the limiting factor. Any efficient learning algorithm should at least take a brief look at each example. But should all examples be given equal attention?

This contribution proposes an empirical answer. We first present an online SVM algorithm based on this premise. LASVM yields competitive misclassification rates after a single pass over the training examples, outspeeding state-of-the-art SVM solvers. Then we show how active example selection can yield faster training, higher accuracies, and simpler models, using only a fraction of the training example labels.

1. Introduction

Electronic computers have vastly enhanced our ability to compute complicated statistical models. Both theory and practice have adapted to take into account the essential compromise between the number of examples and the model capacity (Vapnik, 1998). Cheap, pervasive and networked computers are now enhancing our ability to collect observations to an even greater extent. Data sizes outgrow computer speed. During the last decade, processors became 100 times faster, hard disks became 1000 times bigger.

Very high dimensional learning systems become theoretically possible when training examples are abundant. The computing cost then becomes the limiting factor. Any efficient learning algorithm should at least pay a brief look at each example. But should all training examples be given equal attention?

This contribution proposes an empirical answer:

- Section 2 presents kernel classifiers such as Support Vector Machines (SVM). Kernel classifiers are convenient for our purposes because they clearly express their internal states in terms of subsets of the training examples.

- Section 3 proposes a novel online algorithm, LASVM, which converges to the SVM solution. Experimental evidence on diverse data sets indicates that it reliably reaches competitive accuracies after performing a single pass over the training set. It uses less memory and trains significantly faster than state-of-the-art SVM solvers.

- Section 4 investigates two criteria to select informative training examples at each iteration instead of sequentially processing all examples. Empirical evidence shows that selecting informative examples without making use of the class labels can drastically reduce the training time and produce much more compact classifiers with equivalent or superior accuracy.

- Section 5 discusses the above results and formulates theoretical questions. The simplest question involves the convergence of these algorithms and is addressed by the appendix. Other questions of greater importance remain open.

2. Kernel Classifiers

Early linear classifiers associate classes $y = \pm 1$ to patterns x by first transforming the patterns into feature vectors $\Phi(x)$ and taking the sign of a linear discriminant function:

$$\hat{y}(x) = w'\Phi(x) + b. \tag{1}$$

The parameters w and b are determined by running some learning algorithm on a set of training examples $(x_1, y_1) \cdots (x_n, y_n)$. The feature function Φ is usually hand chosen for each particular problem (Nilsson, 1965).

Aizerman et al. (1964) transform such linear classifiers by leveraging two theorems of the *Reproducing Kernel* theory (Aronszajn, 1950).

The *Representation Theorem* states that many Φ-machine learning algorithms produce parameter vectors w that can be expressed as a linear combinations of the training patterns:

$$w = \sum_{i=1}^{n} \alpha_i \Phi(x_i).$$

The linear discriminant function (1) can then be written as a *kernel expansion*

$$\hat{y}(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i) + b, \tag{2}$$

where the *kernel* function $K(x,y)$ represents the dot products $\Phi(x)'\Phi(y)$ in feature space. This expression is most useful when a large fraction of the coefficients α_i are zero. Examples such that $\alpha_i \neq 0$ are then called *Support Vectors*.

Mercer's Theorem precisely states which kernel functions correspond to a dot product for some feature space. Kernel classifiers deal with the kernel function $K(x,y)$ without explicitly using the corresponding feature function $\Phi(x)$. For instance, the well known *RBF* kernel $K(x,y) = e^{-\gamma\|x-y\|^2}$ defines an implicit feature space of infinite dimension.

Kernel classifiers handle such large feature spaces with the comparatively modest computational costs of the kernel function. On the other hand, kernel classifiers must control the decision funcion complexity in order to avoid overfitting the training data in such large feature spaces. This can be achieved by keeping the number of support vectors as low as possible (Littlestone and Warmuth, 1986) or by searching decision boundaries that separate the examples with the largest margin (Vapnik and Lerner, 1963; Vapnik, 1998).

2.1 Support Vector Machines

Support Vector Machines were defined by three incremental steps. First, Vapnik and Lerner (1963) propose to construct the *Optimal Hyperplane*, that is, the linear classifier that separates the training examples with the widest margin. Then, Guyon, Boser, and Vapnik (1993) propose to construct the Optimal Hyperplane in the feature space induced by a kernel function. Finally, Cortes and Vapnik (1995) show that noisy problems are best addressed by allowing some examples to violate the margin condition.

Support Vector Machines minimize the following objective function in feature space:

$$\min_{w,b} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{with} \quad \begin{cases} \forall i \quad y_i\,\hat{y}(x_i) \geq 1-\xi_i \\ \forall i \quad \xi_i \geq 0. \end{cases} \tag{3}$$

For very large values of the hyper-parameter C, this expression minimizes $\|w\|^2$ under the constraint that all training examples are correctly classified with a margin $y_i\,\hat{y}(x_i)$ greater than 1. Smaller values of C relax this constraint and produce markedly better results on noisy problems (Cortes and Vapnik, 1995).

In practice this is achieved by solving the dual of this convex optimization problem. The coefficients α_i of the SVM kernel expansion (2) are found by defining the dual objective function

$$W(\alpha) = \sum_i \alpha_i y_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j K(x_i,x_j) \tag{4}$$

and solving the SVM *Quadratic Programming* (QP) problem:

$$\max_{\alpha} W(\alpha) \quad \text{with} \quad \begin{cases} \sum_i \alpha_i = 0 \\ A_i \leq \alpha_i \leq B_i \\ \quad A_i = \min(0, Cy_i) \\ \quad B_i = \max(0, Cy_i). \end{cases} \tag{5}$$

The above formulation slightly deviates from the standard formulation (Cortes and Vapnik, 1995) because it makes the α_i coefficients positive when $y_i = +1$ and negative when $y_i = -1$.

SVMs have been very successful and are very widely used because they reliably deliver state-of-the-art classifiers with minimal tweaking.

Computational Cost of SVMs There are two intuitive lower bounds on the computational cost of any algorithm able to solve the SVM QP problem for arbitrary matrices $K_{ij} = K(x_i,x_j)$.

1. Suppose that an oracle reveals whether $\alpha_i = 0$ or $\alpha_i = \pm C$ for all $i = 1 \ldots n$. Computing the remaining $0 < |\alpha_i| < C$ amounts to inverting a matrix of size $R \times R$ where R is the number of support vectors such that $0 < |\alpha_i| < C$. This typically requires a number of operations proportional to R^3.

2. Simply verifying that a vector α is a solution of the SVM QP problem involves computing the gradients of $W(\alpha)$ and checking the Karush-Kuhn-Tucker optimality conditions (Vapnik, 1998). With n examples and S support vectors, this requires a number of operations proportional to $n S$.

Few support vectors reach the upper bound C when it gets large. The cost is then dominated by the $R^3 \approx S^3$. Otherwise the term $n S$ is usually larger. The final number of support vectors therefore is the critical component of the computational cost of the SVM QP problem.

Assume that increasingly large sets of training examples are drawn from an unknown distribution $P(x,y)$. Let $\mathcal{B}$ be the error rate achieved by the best decision function (1) for that distribution. When $\mathcal{B} > 0$, Steinwart (2004) shows that the number of support vectors is asymptotically equivalent to $2n\mathcal{B}$. Therefore, regardless of the exact algorithm used, the asymptotical computational cost of solving the SVM QP problem grows at least like n^2 when C is small and n^3 when C gets large. Empirical evidence shows that modern SVM solvers (Chang and Lin, 2001-2004; Collobert and Bengio, 2001) come close to these scaling laws.

Practice however is dominated by the constant factors. When the number of examples grows, the kernel matrix $K_{ij} = K(x_i, x_j)$ becomes very large and cannot be stored in memory. Kernel values must be computed on the fly or retrieved from a cache of often accessed values. When the cost of computing each kernel value is relatively high, the kernel cache hit rate becomes a major component of the cost of solving the SVM QP problem (Joachims, 1999). Larger problems must be addressed by using algorithms that access kernel values with very consistent patterns.

Section 3 proposes an Online SVM algorithm that accesses kernel values very consistently. Because it computes the SVM optimum, this algorithm cannot improve on the n^2 lower bound. Because it is an online algorithm, early stopping strategies might give approximate solutions in much shorter times. Section 4 suggests that this can be achieved by carefully choosing which examples are processed at each iteration.

Before introducing the new Online SVM, let us briefly describe other existing online kernel methods, beginning with the kernel Perceptron.

2.2 Kernel Perceptrons

The earliest kernel classifiers (Aizerman et al., 1964) were derived from the Perceptron algorithm (Rosenblatt, 1958). The decision function (2) is represented by maintaining the set S of the indices i of the support vectors. The bias parameter b remains zero.

```
Kernel Perceptron
  1) $\mathcal{S} \leftarrow \emptyset$,   $b \leftarrow 0$.
  2) Pick a random example $(x_t, y_t)$
  3) Compute $\hat{y}(x_t) = \sum_{i \in \mathcal{S}} \alpha_i K(x_t, x_i) + b$
  4) If  $y_t \hat{y}(x_t) \leq 0$  then  $\mathcal{S} \leftarrow \mathcal{S} \cup \{t\}$,   $\alpha_t \leftarrow y_t$
  5) Return to step 2.
```

Such *Online Learning Algorithms* require very little memory because the examples are processed one by one and can be discarded after being examined.

Iterations such that $y_t\,\hat{y}(x_t) < 0$ are called *mistakes* because they correspond to patterns misclassified by the perceptron decision boundary. The algorithm then modifies the decision boundary by inserting the misclassified pattern into the kernel expansion. When a solution exists, Novikoff's Theorem (Novikoff, 1962) states that the algorithm converges after a finite number of mistakes, or equivalently after inserting a finite number of support vectors. Noisy data sets are more problematic.

Large Margin Kernel Perceptrons The success of Support Vector Machines has shown that large classification margins were desirable. On the other hand, the Kernel Perceptron (Section 2.2) makes no attempt to achieve large margins because it happily ignores training examples that are very close to being misclassified.

Many authors have proposed to close the gap with online kernel classifiers by providing larger margins. The Averaged Perceptron (Freund and Schapire, 1998) decision rule is the majority vote of all the decision rules obtained after each iteration of the Kernel Perceptron algorithm. This choice provides a bound comparable to those offered in support of SVMs. Other algorithms (Frieß et al., 1998; Gentile, 2001; Li and Long, 2002; Crammer and Singer, 2003) explicitely construct larger margins. These algorithms modify the decision boundary whenever a training example is either misclassified or classified with an insufficient margin. Such examples are then inserted into the kernel expansion with a suitable coefficient. Unfortunately, this change significantly increases the number of mistakes and therefore the number of support vectors. The increased computational cost and the potential overfitting undermines the positive effects of the increased margin.

Kernel Perceptrons with Removal Step This is why Crammer et al. (2004) suggest an additional step for *removing* support vectors from the kernel expansion (2). The Budget Perceptron performs very nicely on relatively clean data sets.

```
Budget Kernel Perceptron ($\beta, N$)
  1) $\mathcal{S} \leftarrow \emptyset, \quad b \leftarrow 0.$
  2) Pick a random example $(x_t, y_t)$
  3) Compute $\hat{y}(x_t) = \sum_{i\in\mathcal{S}} \alpha_i\, K(x_t, x_i) + b$
  4) If $\;y_t\,\hat{y}(x_t) \,\leq\, \beta\;$ then,
      4a) $\mathcal{S} \leftarrow \mathcal{S} \cup \{t\}, \quad \alpha_t \leftarrow y_t$
      4b) If $|\mathcal{S}| > N$ then $\mathcal{S} \leftarrow \mathcal{S} - \{\, \arg\max_{i\in\mathcal{S}}\; y_i\,(\hat{y}(x_i) - \alpha_i\, K(x_i, x_i))\,\}$
  5) Return to step 2.
```

Online kernel classifiers usually experience considerable problems with noisy data sets. Each iteration is likely to cause a mistake because the best achievable misclassification rate for such problems is high. The number of support vectors increases very rapidly and potentially causes overfitting and poor convergence. More sophisticated support vector removal criteria avoid this drawback (Weston et al., 2005). This modified algorithm outperforms all other *online* kernel classifiers on noisy data sets and matches the performance of Support Vector Machines with less support vectors.

3. Online Support Vector Machines

This section proposes a novel online algorithm named LASVM that converges to the SVM solution. This algorithm furthers ideas first presented by Bordes and Bottou (2005). Unlike this previous work, LASVM relies on the traditional "soft margin" SVM formulation, handles noisy data sets, and is nicely related to the SMO algorithm. Experimental evidence on multiple data sets indicates that it reliably reaches competitive test error rates after performing a single pass over the training set. It uses less memory and trains significantly faster than state-of-the-art SVM solvers.

3.1 Quadratic Programming Solvers for SVMs

Sequential Direction Search Efficient numerical algorithms have been developed to solve the SVM QP problem (5). The best known methods are the Conjugate Gradient method (Vapnik, 1982, pages 359–362) and the Sequential Minimal Optimization (Platt, 1999). Both methods work by making successive searches along well chosen directions.

Each direction search solves the restriction of the SVM problem to the half-line starting from the current vector α and extending along the specified direction u. Such a search yields a new feasible vector $\alpha + \lambda^* u$, where

$$\lambda^* = \arg\max W(\alpha + \lambda u) \quad \text{with} \quad 0 \leq \lambda \leq \phi(\alpha, u). \tag{6}$$

The upper bound $\phi(\alpha, u)$ ensures that $\alpha + \lambda u$ is feasible as well:

$$\phi(\alpha, u) \;=\; \min \left\{ \begin{array}{cl} 0 & \text{if } \;\sum_k u_k \neq 0 \\ (B_i - \alpha_i)/u_i & \text{for all } i \text{ such that } u_i > 0 \\ (A_j - \alpha_j)/u_j & \text{for all } j \text{ such that } u_j < 0. \end{array} \right\} \tag{7}$$

Calculus shows that the optimal value is achieved for

$$\lambda^* = \min \left\{ \phi(\alpha, u) \;,\; \frac{\sum_i g_i u_i}{\sum_{i,j} u_i u_j K_{ij}} \right\} \tag{8}$$

where $K_{ij} = K(x_i, x_j)$ and $g = (g_1 \ldots g_n)$ is the gradient of $W(\alpha)$, and

$$g_k \;=\; \frac{\partial W(\alpha)}{\partial \alpha_k} \;=\; y_k - \sum_i \alpha_i K(x_i, x_k) \;=\; y_k - \hat{y}(x_k) + b. \tag{9}$$

Sequential Minimal Optimization Platt (1999) observes that direction search computations are much faster when the search direction u mostly contains zero coefficients. At least two coefficients are needed to ensure that $\sum_k u_k = 0$. The *Sequential Minimal Optimization* (SMO) algorithm uses search directions whose coefficients are all zero except for a single $+1$ and a single -1.

Practical implementations of the SMO algorithm (Chang and Lin, 2001-2004; Collobert and Bengio, 2001) usually rely on a small positive tolerance $\tau > 0$. They only select directions u such that $\phi(\alpha, u) > 0$ and $u'g > \tau$. This means that we can move along direction u without immediately reaching a constraint and increase the value of $W(\alpha)$. Such directions are defined by the so-called τ-*violating pair* (i, j):

$$(i, j) \text{ is a } \tau\text{-}\textit{violating pair} \iff \left\{ \begin{array}{c} \alpha_i < B_i \\ \alpha_j > A_j \\ g_i - g_j > \tau. \end{array} \right.$$

```
SMO Algorithm
  1) Set $\alpha \leftarrow 0$ and compute the initial gradient $g$ (equation 9)
  2) Choose a $\tau$-violating pair $(i,j)$. Stop if no such pair exists.
  3) $\lambda \leftarrow \min\left\{ \frac{g_i - g_j}{K_{ii}+K_{jj}-2K_{ij}},\ B_i-\alpha_i,\ \alpha_j - A_j \right\}$
     $\alpha_i \leftarrow \alpha_i + \lambda\,,\ \ \alpha_j \leftarrow \alpha_j - \lambda$
     $g_s \leftarrow g_s - \lambda(K_{is}-K_{js}) \quad \forall s \in \{1\ldots n\}$
  4) Return to step (2)
```

The above algorithm does not specify how exactly the τ-violating pairs are chosen. Modern implementations of SMO select the τ-violating pair (i,j) that maximizes the directional gradient $u'g$. This choice was described in the context of Optimal Hyperplanes in both (Vapnik, 1982, pages 362–364) and (Vapnik et al., 1984).

Regardless of how exactly the τ-violating pairs are chosen, Keerthi and Gilbert (2002) assert that the SMO algorithm stops after a finite number of steps. This assertion is correct despite a slight flaw in their final argument (Takahashi and Nishi, 2003).

When SMO stops, no τ-violating pair remain. The corresponding α is called a τ-*approximate solution*. Proposition 13 in appendix A establishes that such approximate solutions indicate the location of the solution(s) of the SVM QP problem when the tolerance τ become close to zero.

3.2 Online LASVM

This section presents a novel online SVM algorithm named LASVM. There are two ways to view this algorithm. LASVM is an online kernel classifier sporting a support vector removal step: vectors collected in the current kernel expansion can be removed during the online process. LASVM also is a reorganization of the SMO sequential direction searches and, as such, converges to the solution of the SVM QP problem.

Compared to basic kernel perceptrons (Aizerman et al., 1964; Freund and Schapire, 1998), the LASVM algorithm features a removal step and gracefully handles noisy data. Compared to kernel perceptrons with removal steps (Crammer et al., 2004; Weston et al., 2005), LASVM converges to the known SVM solution. Compared to a traditional SVM solver (Platt, 1999; Chang and Lin, 2001-2004; Collobert and Bengio, 2001), LASVM brings the computational benefits and the flexibility of online learning algorithms. Experimental evidence indicates that LASVM matches the SVM accuracy after a single sequential pass over the training examples.

This is achieved by alternating two kinds of direction searches named PROCESS and REPROCESS. Each direction search involves a pair of examples. Direction searches of the PROCESS kind involve at least one example that is not a support vector of the current kernel expansion. They potentially can change the coefficient of this example and make it a support vector. Direction searches of the REPROCESS kind involve two examples that already are support vectors in the current kernel expansion. They potentially can zero the coefficient of one or both support vectors and thus remove them from the kernel expansion.

Building Blocks The LASVM algorithm maintains three essential pieces of information: the set $\mathcal{S}$ of potential support vector indices, the coefficients α_i of the current kernel expansion, and the partial derivatives g_i defined in (9). Variables α_i and g_i contain meaningful values when $i \in \mathcal{S}$ only.

The coefficient α_i are assumed to be null if $i \notin \mathcal{S}$. On the other hand, set $\mathcal{S}$ might contain a few indices i such that $\alpha_i = 0$.

The two basic operations of the Online LASVM algorithm correspond to steps 2 and 3 of the SMO algorithm. These two operations differ from each other because they have different ways to select τ-violating pairs.

The first operation, PROCESS, attempts to insert example $k \notin \mathcal{S}$ into the set of current support vectors. In the online setting this can be used to process a new example at time t. It first adds example $k \notin \mathcal{S}$ into $\mathcal{S}$ (step 1-2). Then it searches a second example in $\mathcal{S}$ to find the τ-violating pair with maximal gradient (steps 3-4) and performs a direction search (step 5).

```
LASVM PROCESS(k)
  1) Bail out if $k \in \mathcal{S}$.
  2) $\alpha_k \leftarrow 0\,,\quad g_k \leftarrow y_k - \sum_{s \in \mathcal{S}} \alpha_s K_{ks}\,,\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{k\}$
  3) If $y_k = +1$ then
         $i \leftarrow k\,,\quad j \leftarrow \arg\min_{s \in S} g_s$ with $\alpha_s > A_s$
     else
         $j \leftarrow k\,,\quad i \leftarrow \arg\max_{s \in S} g_s$ with $\alpha_s < B_s$
  4) Bail out if $(i, j)$ is not a $\tau$-violating pair.
  5) $\lambda \leftarrow \min \left\{ \frac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}},\ B_i - \alpha_i,\ \alpha_j - A_j \right\}$
     $\alpha_i \leftarrow \alpha_i + \lambda\,,\quad \alpha_j \leftarrow \alpha_j - \lambda$
     $g_s \leftarrow g_s - \lambda(K_{is} - K_{js}) \quad \forall s \in \mathcal{S}$
```

The second operation, REPROCESS, removes some elements from $\mathcal{S}$. It first searches the τ-violating pair of elements of $\mathcal{S}$ with maximal gradient (steps 1-2), and performs a direction search (step 3). Then it removes blatant non support vectors (step 4). Finally it computes two useful quantities: the bias term b of the decision function (2) and the gradient δ of the most τ-violating pair in $\mathcal{S}$.

```
LASVM REPROCESS
  1) $i \leftarrow \arg\max_{s \in \mathcal{S}} g_s$ with $\alpha_s < B_s$
     $j \leftarrow \arg\min_{s \in \mathcal{S}} g_s$ with $\alpha_s > A_s$
  2) Bail out if $(i, j)$ is not a $\tau$-violating pair.
  3) $\lambda \leftarrow \min \left\{ \frac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}},\ B_i - \alpha_i,\ \alpha_j - A_j \right\}$

     $\alpha_i \leftarrow \alpha_i + \lambda\,,\quad \alpha_j \leftarrow \alpha_j - \lambda$
     $g_s \leftarrow g_s - \lambda(K_{is} - K_{js}) \quad \forall s \in \mathcal{S}$
  4) $i \leftarrow \arg\max_{s \in \mathcal{S}} g_s$ with $\alpha_s < B_s$
     $j \leftarrow \arg\min_{s \in \mathcal{S}} g_s$ with $\alpha_s > A_s$
     For all $s \in \mathcal{S}$ such that $\alpha_s = 0$
         If $y_s = -1$ and $g_s \geq g_i$ then $\mathcal{S} = \mathcal{S} - \{s\}$
         If $y_s = +1$ and $g_s \leq g_j$ then $\mathcal{S} = \mathcal{S} - \{s\}$
  5) $b \leftarrow (g_i + g_j)/2\,,\quad \delta \leftarrow g_i - g_j$
```

Online LASVM After initializing the state variables (step 1), the Online LASVM algorithm alternates PROCESS and REPROCESS a predefined number of times (step 2). Then it simplifies the kernel expansion by running REPROCESS to remove all τ-violating pairs from the kernel expansion (step 3).

```
LASVM
  1) Initialization:
     Seed $\mathcal{S}$ with a few examples of each class.
     Set $\alpha \leftarrow 0$ and compute the initial gradient $g$ (equation 9)
  2) Online Iterations:
     Repeat a predefined number of times:
        - Pick an example $k_t$
        - Run PROCESS($k_t$).
        - Run REPROCESS once.
  3) Finishing:
     Repeat REPROCESS until $\delta \leq \tau$.
```

LASVM can be used in the online setup where one is given a continuous stream of fresh random examples. The online iterations process fresh training examples as they come. LASVM can also be used as a stochastic optimization algorithm in the offline setup where the complete training set is available before hand. Each iteration randomly picks an example from the training set.

In practice we run the LASVM online iterations in epochs. Each epoch sequentially visits all the randomly shuffled training examples. After a predefined number P of epochs, we perform the finishing step. A single epoch is consistent with the use of LASVM in the online setup. Multiple epochs are consistent with the use of LASVM as a stochastic optimization algorithm in the offline setup.

Convergence of the Online Iterations Let us first ignore the finishing step (step 3) and assume that online iterations (step 2) are repeated indefinitely. Suppose that there are remaining τ-violating pairs at iteration T.

a.) If there are τ-violating pairs (i, j) such that $i \in \mathcal{S}$ and $j \in \mathcal{S}$, one of them will be exploited by the next REPROCESS.

b.) Otherwise, if there are τ-violating pairs (i, j) such that $i \in \mathcal{S}$ or $j \in \mathcal{S}$, each subsequent PROCESS has a chance to exploit one of them. The intervening REPROCESS do nothing because they bail out at step 2.

c.) Otherwise, all τ-violating pairs involve indices outside $\mathcal{S}$. Subsequent calls to PROCESS and REPROCESS bail out until we reach a time $t > T$ such that $k_t = i$ and $k_{t+1} = j$ for some τ-violating pair (i, j). The first PROCESS then inserts i into $\mathcal{S}$ and bails out. The following REPROCESS bails out immediately. Finally the second PROCESS locates pair (i, j).

This case is not important in practice. There usually is a support vector $s \in \mathcal{S}$ such that $A_s < \alpha_s < B_s$. We can then write $g_i - g_j = (g_i - g_s) + (g_s - g_j) \leq 2\tau$ and conclude that we already have reached a 2τ-approximate solution.

The LASVM online iterations therefore work like the SMO algorithm. Remaining τ-violating pairs is sooner or later exploited by either PROCESS or REPROCESS. As soon as a τ-approximate solution is reached, the algorithm stops updating the coefficients α. Theorem 18 in the appendix gives more precise convergence results for this stochastic algorithm.

The finishing step (step 3) is only useful when one limits the number of online iterations. Running LASVM usually consists in performing a predefined number P of epochs and running the finishing step. Each epoch performs n online iterations by sequentially visiting the randomly shuffled training examples. Empirical evidence suggests indeed that a *single epoch* yields a classifier almost as good as the SVM solution.

Computational Cost of LASVM Both PROCESS and REPROCESS require a number of operations proportional to the number S of support vectors in set $\mathcal{S}$. Performing P epochs of online iterations requires a number of operations proportional to $n\,P\,\bar{S}$. The average number $\bar{S}$ of support vectors scales no more than linearly with n because each online iteration brings at most one new support vector. The asymptotic cost therefore grows like n^2 at most. The finishing step is similar to running a SMO solver on a SVM problem with only S training examples. We recover here the n^2 to n^3 behavior of standard SVM solvers.

Online algorithms access kernel values with a very specific pattern. Most of the kernel values accessed by PROCESS and REPROCESS involve only support vectors from set $\mathcal{S}$. Only PROCESS on a new example x_{k_t} accesses S fresh kernel values $K(x_{k_t}, x_i)$ for $i \in \mathcal{S}$.

Implementation Details Our LASVM implementation reorders the examples after every PROCESS or REPROCESS to ensure that the current support vectors come first in the reordered list of indices. The kernel cache records truncated rows of the reordered kernel matrix. SVMLight (Joachims, 1999) and LIBSVM (Chang and Lin, 2001-2004) also perform such reorderings, but do so rather infrequently (Joachims, 1999). The reordering overhead is acceptable during the online iterations because the computation of fresh kernel values takes much more time.

Reordering examples during the finishing step was more problematic. We eventually deployed an adaptation of the *shrinking* heuristic (Joachims, 1999) for the finishing step only. The set $\mathcal{S}$ of support vectors is split into an active set S_a and an inactive set S_i. All support vectors are initially active. The REPROCESS iterations are restricted to the active set S_a and do not perform any reordering. About every 1000 iterations, support vectors that hit the boundaries of the box constraints are either removed from the set $\mathcal{S}$ of support vectors or moved from the active set $\mathcal{S}_a$ to the inactive set $\mathcal{S}_i$. When all τ-violating pairs of the active set are exhausted, the inactive set examples are transferred back into the active set. The process continues as long as the merged set contains τ-violating pairs.

3.3 MNIST Experiments

The Online LASVM was first evaluated on the MNIST[1] handwritten digit data set (Bottou et al., 1994). Computing kernel values for this data set is relatively expensive because it involves dot products of 784 gray level pixel values. In the experiments reported below, all algorithms use the same code for computing kernel values. The ten binary classification tasks consist of separating each digit class from the nine remaining classes. All experiments use RBF kernels with $\gamma = 0.005$

[1] This data set is available at `http://yann.lecun.com/exdb/mnist`.

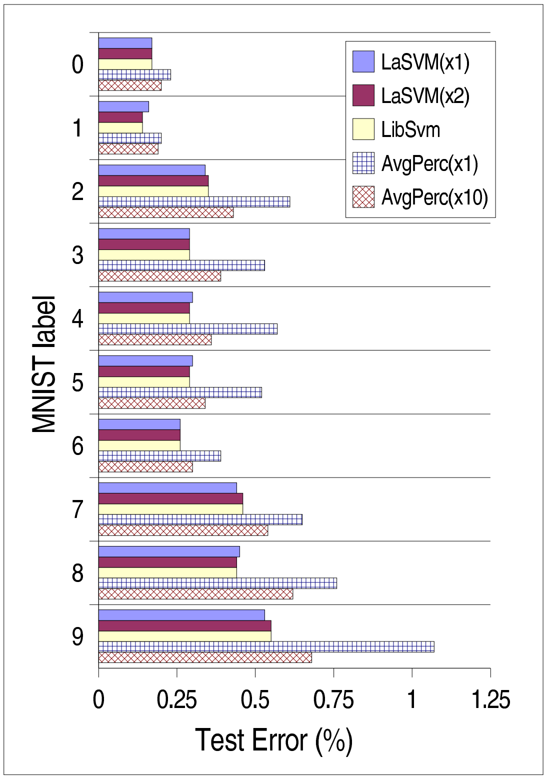


Figure 1: Compared test error rates for the ten MNIST binary classifiers.

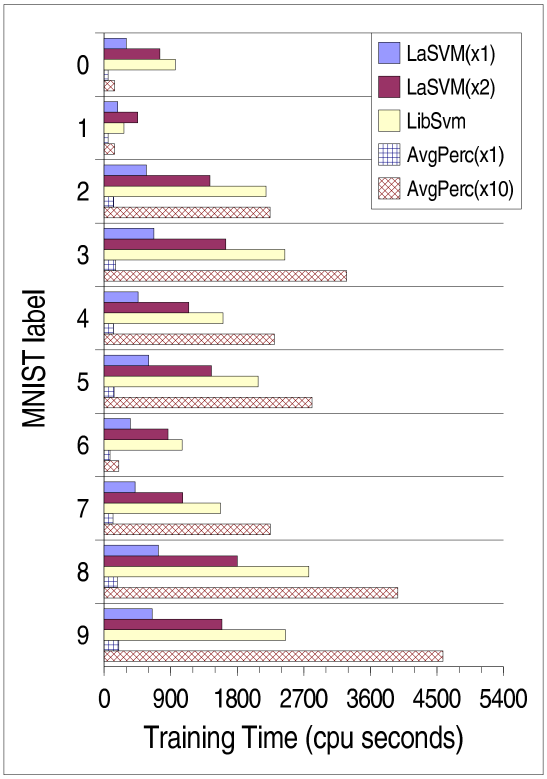


Figure 2: Compared training times for the ten MNIST binary classifiers.

and the same training parameters $C = 1000$ and $\tau = 0.001$. Unless indicated otherwise, the kernel cache size is 256MB.

LASVM versus Sequential Minimal Optimization Baseline results were obtained by running the state-of-the-art SMO solver LIBSVM (Chang and Lin, 2001-2004). The resulting classifier accurately represents the SVM solution.

Two sets of results are reported for LASVM. The LASVM×1 results were obtained by performing a single epoch of online iterations: each training example was processed exactly once during a single sequential sweep over the training set. The LASVM×2 results were obtained by performing two epochs of online iterations.

Figures 1 and 2 show the resulting test errors and training times. LASVM×1 runs about three times faster than LIBSVM and yields test error rates very close to the LIBSVM results. Standard paired significance tests indicate that these small differences are not significant. LASVM×2 usually runs faster than LIBSVM and very closely tracks the LIBSVM test errors.

Neither the LASVM×1 or LASVM×2 experiments yield the exact SVM solution. On this data set, LASVM reaches the exact SVM solution after about five epochs. The first two epochs represent the bulk of the computing time. The remaining epochs run faster when the kernel cache is large

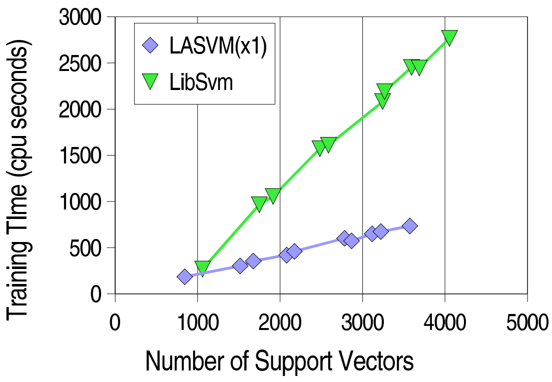


Figure 3: Training time as a function of the number of support vectors.

Algorithm	Error	Time
LIBSVM	**1.36%**	17400s
LASVM×1	1.42%	**4950s**
LASVM×2	**1.36%**	12210s

Figure 4: Multiclass errors and training times for the MNIST data set.

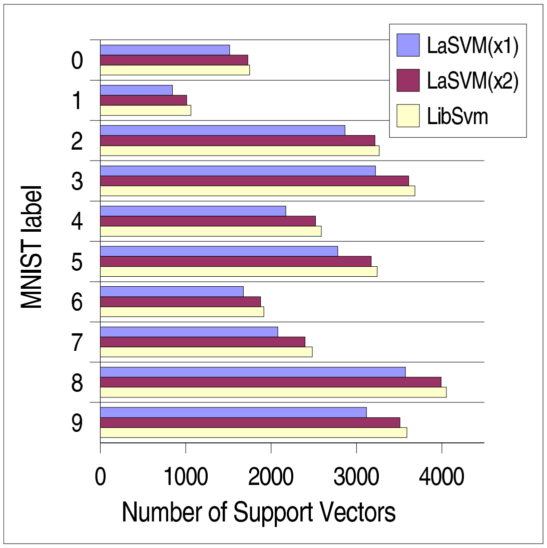


Figure 5: Compared numbers of support vectors for the ten MNIST binary classifiers.

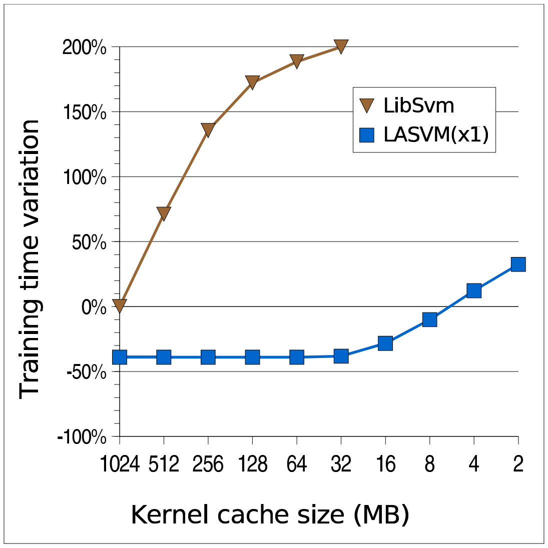


Figure 6: Training time variation as a function of the cache size. Relative changes with respect to the 1GB LIBSVM times are averaged over all ten MNIST classifiers.

enough to hold all the dot products involving support vectors. Yet the overall optimization times are not competitive with those achieved by LIBSVM.

Figure 3 shows the training time as a function of the final number of support vectors for the ten binary classification problems. Both LIBSVM and LASVM×1 show a linear dependency. The Online LASVM algorithm seems more efficient overall.

Figure 4 shows the multiclass error rates and training times obtained by combining the ten classifiers using the well known 1-versus-rest scheme (Schölkopf and Smola, 2002). LASVM×1 provides almost the same accuracy with much shorter training times. LASVM×2 reproduces the LIBSVM accuracy with slightly shorter training time.

Figure 5 shows the resulting number of support vectors. A single epoch of the Online LASVM algorithm gathers most of the support vectors of the SVM solution computed by LIBSVM. The first iterations of the Online LASVM might indeed ignore examples that later become support vectors. Performing a second epoch captures most of the missing support vectors.

LASVM versus the Averaged Perceptron The computational advantage of LASVM relies on its apparent ability to match the SVM accuracies after a single epoch. Therefore it must be compared with algorithms such as the Averaged Perceptron (Freund and Schapire, 1998) that provably match well known upper bounds on the SVM accuracies. The AVGPERC×1 results in Figures 1 and 2 were obtained after running a single epoch of the Averaged Perceptron. Although the computing times are very good, the corresponding test errors are not competitive with those achieved by either LIBSVM or LASVM. Freund and Schapire (1998) suggest that the Averaged Perceptron approaches the actual SVM accuracies after 10 to 30 epochs. Doing so no longer provides the theoretical guarantees. The AVGPERC×10 results in Figures 1 and 2 were obtained after ten epochs. Test error rates indeed approach the SVM results. The corresponding training times are no longer competitive.

Impact of the Kernel Cache Size These training times stress the importance of the kernel cache size. Figure 2 shows how the AVGPERC×10 runs much faster on problems 0, 1, and 6. This is happening because the cache is large enough to accomodate the dot products of all examples with all support vectors. Each repeated iteration of the Average Perceptron requires very few additional kernel evaluations. This is much less likely to happen when the training set size increases. Computing times then increase drastically because repeated kernel evaluations become necessary.

Figure 6 compares how the LIBSVM and LASVM×1 training times change with the kernel cache size. The vertical axis reports the relative changes with respect to LIBSVM with one gigabyte of kernel cache. These changes are averaged over the ten MNIST classifiers. The plot shows how LASVM tolerates much smaller caches. On this problem, *LASVM with a 8MB cache runs slightly faster than LIBSVM with a 1024MB cache.*

Useful orders of magnitude can be obtained by evaluating how large the kernel cache must be to avoid the systematic recomputation of dot-products. Following the notations of Section 2.1, let n be the number of examples, S be the number of support vectors, and $R \leq S$ the number of support vectors such that $0 < |\alpha_i| < C$.

- In the case of LIBSVM, the cache must accommodate about nR terms: the examples selected for the SMO iterations are usually chosen among the R free support vectors. Each SMO iteration needs n distinct dot-products for each selected example.

- To perform a *single* LASVM epoch, the cache must only accommodate about SR terms: since the examples are visited only once, the dot-products computed by a PROCESS operation can only be reutilized by subsequent REPROCESS operations. The examples selected by REPROCESS are usually chosen amont the R free support vectors; for each selected example, REPROCESS needs one distinct dot-product per support vector in set $\mathcal{S}$.

- To perform *multiple* LASVM epochs, the cache must accommodate about nS terms: the dot-products computed by processing a particular example are reused when processing the same example again in subsequent epochs. This also applies to multiple Averaged Perceptron epochs.

An efficient single epoch learning algorithm is therefore very desirable when one expects S to be much smaller than n. Unfortunately, this may not be the case when the data set is noisy. Section 3.4 presents results obtained in such less favorable conditions. Section 4 then proposes an active learning method to contain the growth of the number of support vectors, and recover the full benefits of the online approach.

3.4 Multiple Data Set Experiments

Further experiments were carried out with a collection of standard data sets representing diverse noise conditions, training set sizes, and input dimensionality. Figure 7 presents these data sets and the parameters used for the experiments.

Kernel computation times for these data sets are extremely fast. The data either has low dimensionality or can be represented with sparse vectors. For instance, computing kernel values for two Reuters documents only involves words common to both documents (excluding stop words). The Forest experiments use a kernel implemented with hand optimized assembly code (Graf et al., 2005).

Figure 8 compares the solutions returned by LASVM×1 and LIBSVM. The LASVM×1 experiments call the kernel function much less often, but do not always run faster. The fast kernel computation times expose the relative weakness of our kernel cache implementation. The LASVM×1 accuracies are very close to the LIBSVM accuracies. The number of support vectors is always slightly smaller.

LASVM×1 essentially achieves consistent results over very diverse data sets, after performing one single epoch over the training set only. In this situation, the LASVM PROCESS function gets only once chance to take a particular example into the kernel expansion and potentially make it a support vector. The conservative strategy would be to take all examples and sort them out during the finishing step. The resulting training times would always be worse than LIBSVM's because the finishing step is itself a simplified SMO solver. Therefore LASVM online iterations are able to very quickly discard a large number of examples with a high confidence. This process is not perfect because we can see that the LASVM×1 number of support vectors are smaller than LIBSVM's. Some good support vectors are discarded erroneously.

Figure 9 reports the relative variations of the test error, number of support vectors, and training time measured before and after the finishing step. The online iterations pretty much select the right support vectors on clean data sets such as “Waveform”, “Reuters” or “USPS”, and the finishing step does very little. On the other problems the online iterations keep much more examples as potential support vectors. The finishing step significantly improves the accuracy on noisy data sets such as “Banana”, “Adult” or “USPS+N”, and drastically increases the computation time on data sets with complicated decision boundaries such as “Banana” or “Forest”.

	Train Size	**Test Size**	γ	C	**Cache**	τ	**Notes**
Waveform[1]	4000	1000	0.05	1	40M	0.001	Artificial data, 21 dims.
Banana[1]	4000	1300	0.5	316	40M	0.001	Artificial data, 2 dims.
Reuters[2]	7700	3299	1	1	40M	0.001	Topic “moneyfx” vs. rest.
USPS[3]	7329	2000	2	1000	40M	0.001	Class “0” vs. rest.
USPS+N[3]	7329	2000	2	10	40M	0.001	10% training label noise.
Adult[3]	32562	16282	0.005	100	40M	0.001	As in (Platt, 1999).
Forest[3] (100k)	100000	50000	1	3	512M	0.001	As in (Collobert et al., 2002).
Forest[3] (521k)	521012	50000	1	3	1250M	0.01	As in (Collobert et al., 2002).

[1] `http://mlg.anu.edu.au/∼raetsch/data/index.html`

[2] `http://www.daviddlewis.com/resources/testcollections/reuters21578`

[3] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases`

Figure 7: Data Sets discussed in Section 3.4.

	LIBSVM				**LASVM×1**			
Data Set	**Error**	**SV**	**KCalc**	**Time**	**Error**	**SV**	**KCalc**	**Time**
Waveform	8.82%	1006	4.2M	3.2s	**8.68%**	**948**	**2.2M**	**2.7s**
Banana	9.96%	873	6.8M	9.9s	9.98%	869	6.7M	10.0s
Reuters	2.76%	1493	11.8M	**24s**	2.76%	1504	**9.2M**	31.4s
USPS	0.41%	236	1.97M	**13.5s**	0.43%	**201**	**1.08M**	15.9s
USPS+N	**0.41%**	2750	63M	305s	0.53%	**2572**	**20M**	**178s**
Adult	14.90%	11327	1760M	1079s	14.94%	11268	**626M**	**809s**
Forest (100k)	**8.03%**	43251	27569M	14598s	8.15%	**41750**	**18939M**	**10310s**
Forest (521k)	4.84%	124782	316750M	159443s	4.83%	122064	**188744M**	**137183s**

Figure 8: Comparison of LIBSVM versus LASVM×1: Test error rates (Error), number of support vectors (SV), number of kernel calls (KCalc), and training time (Time). Bold characters indicate significative differences.

	Relative Variation		
Data Set	**Error**	**SV**	**Time**
Waveform	-0%	-0%	+4%
Banana	-79%	-74%	+185%
Reuters	0%	-0%	+3%
USPS	0%	-2%	+0%
USPS+N%	-67%	-33%	+7%
Adult	-13%	-19%	+80%
Forest (100k)	-1%	-24%	+248%
Forest (521k)	-2%	-24%	+84%

Figure 9: Relative variations of test error, number of support vectors and training time measured before and after the finishing step.

3.5 The Collection of Potential Support Vectors

The final step of the REPROCESS operation computes the current value of the kernel expansion bias b and the stopping criterion δ:

$$\begin{aligned} g_{\max} &= \max_{s\in\mathcal{S}} g_s \ \text{ with } \alpha_s < B_s & \qquad b &= \frac{g_{\max}+g_{\min}}{2} \\ g_{\min} &= \min_{s\in\mathcal{S}} g_s \ \text{ with } \alpha_s > A_s & \qquad \delta &= g_{\max}-g_{\min}. \end{aligned} \tag{10}$$

The quantities $g_{\min}$ and $g_{\max}$ can be interpreted as bounds for the decision threshold b. The quantity δ then represents an uncertainty on the decision threshold b.

The quantity δ also controls how LASVM collects potential support vectors. The definition of PROCESS and the equality (9) indicate indeed that PROCESS(k) adds the support vector x_k to the kernel expansion if and only if

$$y_k\,\hat{y}(x_k) < 1 + \frac{\delta}{2} - \tau. \tag{11}$$

When α is optimal, the uncertainty δ is zero, and this condition matches the Karush-Kuhn-Tucker condition for support vectors $y_k\,\hat{y}(x_k) \leq 1$.

Intuitively, relation (11) describes how PROCESS collects potential support vectors that are compatible with the current uncertainty level δ on the threshold b. Simultaneously, the REPROCESS operations reduce δ and discard the support vectors that are no longer compatible with this reduced uncertainty.

The online iterations of the LASVM algorithm make equal numbers of PROCESS and REPROCESS for purely heuristic reasons. Nothing guarantees that this is the optimal proportion. The results reported in Figure 9 clearly suggest to investigate this arbitrage more closely.

Variations on REPROCESS Experiments were carried out with a slightly modified LASVM algorithm: instead of performing a single REPROCESS, the modified online iterations repeatedly run REPROCESS until the uncertainty δ becomes smaller than a predefined threshold $\delta_{\max}$.

Figure 10 reports comparative results for the “Banana” data set. Similar results were obtained with other data sets. The three plots report test error rates, training time, and number of support vectors as a function of $\delta_{\max}$. These measurements were performed after one epoch of online iterations without finishing step, and after one and two epochs followed by the finishing step. The corresponding LIBSVM figures are indicated by large triangles on the right side of the plot.

Regardless of $\delta_{\max}$, the SVM test error rate can be replicated by performing two epochs followed by a finishing step. However, this does not guarantee that the optimal SVM solution has been reached.

Large values of $\delta_{\max}$ essentially correspond to the unmodified LASVM algorithm. Small values of $\delta_{\max}$ considerably increases the computation time because each online iteration calls REPROCESS many times in order to sufficiently reduce δ. Small values of $\delta_{\max}$ also remove the LASVM ability to produce a competitive result after a single epoch followed by a finishing step. The additional optimization effort discards support vectors more aggressively. Additional epochs are necessary to recapture the support vectors that should have been kept.

There clearly is a sweet spot around $\delta_{\max} = 3$ when one epoch of online iterations alone almost match the SVM performance and also makes the finishing step very fast. This sweet spot is difficult to find in general. If $\delta_{\max}$ is a little bit too small, we must make one extra epoch. If $\delta_{\max}$ is a little

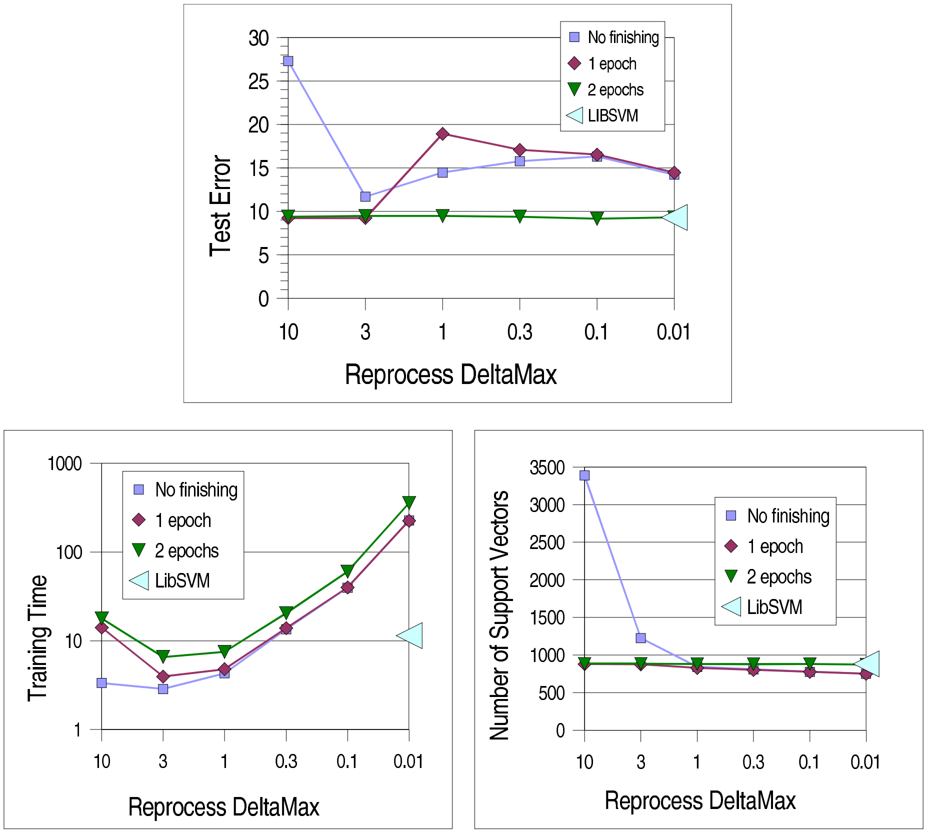


Figure 10: Impact of additional REPROCESS measured on “Banana” data set. During the LASVM online iterations, calls to REPROCESS are repeated until $\delta < \delta_{\max}$.

bit too large, the algorithm behaves like the unmodified LASVM. Short of a deeper understanding of these effects, the unmodified LASVM seems to be a robust compromise.

SimpleSVM The right side of each plot in Figure 10 corresponds to an algorithm that optimizes the coefficient of the current support vectors at each iteration. This is closely related to the SimpleSVM algorithm (Vishwanathan et al., 2003). Both LASVM and the SimpleSVM update a current kernel expansion by adding or removing one or two support vectors at each iteration. The two key differences are the numerical objective of these updates and their computational costs.

Whereas each SimpleSVM iteration seeks the optimal solution of the SVM QP problem restricted to the current set of support vectors, the LASVM online iterations merely attempt to improve the value of the dual objective function $W(\alpha)$. As a a consequence, LASVM needs a finishing step and the SimpleSVM does not. On the other hand, Figure 10 suggests that seeking the optimum at each iteration discards support vectors too aggressively to reach competitive accuracies after a single epoch.

Each SimpleSVM iteration updates the current kernel expansion using rank 1 matrix updates (Cauwenberghs and Poggio, 2001) whose computational cost grows as the square of the number of support vectors. LASVM performs these updates using SMO direction searches whose cost grows linearly with the number of examples. Rank 1 updates make good sense when one seeks the optimal coefficients. On the other hand, all the kernel values involving support vectors must be stored in memory. The LASVM direction searches are more amenable to caching strategies for kernel values.

4. Active Selection of Training Examples

The previous section presents LASVM as an Online Learning algorithm or as a Stochastic Optimization algorithm. In both cases, the LASVM online iterations pick random training examples. The current section departs from this framework and investigates more refined ways to select an informative example for each iteration.

This departure is justified in the offline setup because the complete training set is available beforehand and can be searched for informative examples. It is also justified in the online setup when the continuous stream of fresh training examples is too costly to process, either because the computational requirements are too high, or because it is inpractical to label all the potential training examples.

In particular, we show that selecting informative examples yields considerable speedups. Furthermore, training example selection can be achieved without the knowledge of the training example labels. In fact, excessive reliance on the training example labels can have very detrimental effects.

4.1 Gradient Selection

The most obvious approach consists in selecting an example k such that the PROCESS operation results in a large increase of the dual objective function. This can be approximated by choosing the example which yields the τ-violating pair with the largest gradient. Depending on the class y_k, the PROCESS(k) operation considers pair (k,j) or (i,k) where i and j are the indices of the examples in $\mathcal{S}$ with extreme gradients:

$$i = \arg\max_{s\in\mathcal{S}} g_s \text{ with } \alpha_s < B_s\,, \qquad j = \arg\min_{s\in\mathcal{S}} g_s \text{ with } \alpha_s > A_s.$$

The corresponding gradients are $g_k - g_j$ for positive examples and $g_i - g_k$ for negative examples. Using the expression (9) of the gradients and the value of b and δ computed during the previous REPROCESS (10), we can write:

$$\begin{aligned}
\text{when } y_k = +1, \quad & g_k - g_j = y_k g_k - \frac{g_i+g_j}{2} + \frac{g_i-g_j}{2} = 1 + \frac{\delta}{2} - y_k\hat{y}(x_k) \\
\text{when } y_k = -1, \quad & g_i - g_k = \frac{g_i+g_j}{2} + \frac{g_i-g_j}{2} + y_k g_k = 1 + \frac{\delta}{2} - y_k\hat{y}(x_k).
\end{aligned}$$

This expression shows that the *Gradient Selection Criterion* simply suggests to pick the most misclassified example

$$k_G \;=\; \arg\min_{k\notin\mathcal{S}} \; y_k\hat{y}(x_k). \tag{12}$$

4.2 Active Selection

Always picking the most misclassified example is reasonable when one is very confident of the training example labels. On noisy data sets, this strategy is simply going to pick mislabelled examples or examples that sit on the wrong side of the optimal decision boundary.

When training example labels are unreliable, a conservative approach chooses the example k_A that yields the strongest minimax gradient:

$$k_A \;=\; \arg\min_{k\notin\mathcal{S}} \;\max_{y=\pm1}\; y\,\hat{y}(x_k) \;=\; \arg\min_{k\notin\mathcal{S}}\; |\hat{y}(x_k)|\,. \tag{13}$$

This *Active Selection Criterion* simply chooses the example that comes closest to the current decision boundary. Such a choice yields a gradient approximatively equal to $1+\delta/2$ regardless of the true class of the example.

Criterion (13) does not depend on the labels y_k. The resulting learning algorithm only uses the labels of examples that have been selected during the previous online iterations. This is related to the *Pool Based Active Learning* paradigm (Cohn et al., 1990).

Early active learning literature, also known as *Experiment Design* (Fedorov, 1972), contrasts the passive learner, who observes examples (x,y), with the active learner, who constructs queries x and observes their labels y. In this setup, the active learner cannot beat the passive learner because he lacks information about the input pattern distribution (Eisenberg and Rivest, 1990). Pool-based active learning algorithms observe the pattern distribution from a vast pool of unlabelled examples. Instead of constructing queries, they incrementally select unlabelled examples x_k and obtain their labels y_k from an oracle.

Several authors (Campbell et al., 2000; Schohn and Cohn, 2000; Tong and Koller, 2000) propose incremental active learning algorithms that clearly are related to Active Selection. The initialization consists of obtaining the labels for a small random subset of examples. A SVM is trained using all the labelled examples as a training set. Then one searches the pool for the unlabelled example that comes closest to the SVM decision boundary, one obtains the label of this example, retrains the SVM and reiterates the process.

4.3 Randomized Search

Both criteria (12) and (13) suggest a search through all the training examples. This is impossible in the online setup and potentially expensive in the offline setup.

It is however possible to locate an approximate optimum by simply examining a small constant number of randomly chosen examples. The randomized search first samples M random training examples and selects the best one among these M examples. With probability $1-\eta^M$, the value of the criterion for this example exceeds the η-quantile of the criterion for all training examples (Schölkopf and Smola, 2002, theorem 6.33) regardless of the size of the training set. In practice this means that the best among 59 random training examples has 95% chances to belong to the best 5% examples in the training set.

Randomized search has been used in the offline setup to accelerate various machine learning algorithms (Domingo and Watanabe, 2000; Vishwanathan et al., 2003; Tsang et al., 2005). In the online setup, randomized search is the only practical way to select training examples. For instance, here is a modification of the basic LASVM algorithm to select examples using the Active Selection Criterion with Randomized Search:

```
LASVM + Active Example Selection + Randomized Search
  1) Initialization:
     Seed $\mathcal{S}$ with a few examples of each class.
     Set $\alpha \leftarrow 0$ and $g \leftarrow 0$.
  2) Online Iterations:
     Repeat a predefined number of times:
       - Pick $M$ random examples $s_1 \dots s_M$.
       - $k_t \leftarrow \arg\min_{i=1...M} \ |\hat{y}(x_{s_i})\,|$
       - Run PROCESS($k_t$).
       - Run REPROCESS once.
  3) Finishing:
     Repeat REPROCESS until $\delta \le \tau$.
```

Each online iteration of the above algorithm is about M times more computationally expensive that an online iteration of the basic LASVM algorithm. Indeed one must compute the kernel expansion (2) for M fresh examples instead of a single one (9). This cost can be reduced by heuristic techniques for adapting M to the current conditions. For instance, we present experimental results where one stops collecting new examples as soon as $\mathcal{M}$ contains five examples such that $|\,\hat{y}(x_s)\,| < 1+\delta/2$.

Finally the last two paragraphs of appendix A discuss the convergence of LASVM with example selection according to the gradient selection criterion or the active selection criterion. The gradient selection criterion always leads to a solution of the SVM problem. On the other hand, the active selection criterion only does so when one uses the sampling method. In practice this convergence occurs very slowly. The next section presents many reasons to prefer the intermediate kernel classifiers visited by this algorithm.

4.4 Example Selection for Online SVMs

This section experimentally compares the LASVM algorithm using different example selection methods. Four different algorithms are compared:

- RANDOM example selection randomly picks the next training example among those that have not yet been PROCESSed. This is equivalent to the plain LASVM algorithm discussed in Section 3.2.

- GRADIENT example selection consists in sampling 50 random training examples among those that have not yet been PROCESSed. The sampled example with the smallest $y_k\,\hat{y}(x_k)$ is then selected.

- ACTIVE example selection consists in sampling 50 random training examples among those that have not yet been PROCESSed. The sampled example with the smallest $|\hat{y}(x_k)|$ is then selected.

- AUTOACTIVE example selection attempts to adaptively select the sampling size. Sampling stops as soon as 5 examples are within distance $1+\delta/2$ of the decision boundary. The maximum sample size is 100 examples. The sampled example with the smallest $|\hat{y}(x_k)|$ is then selected.

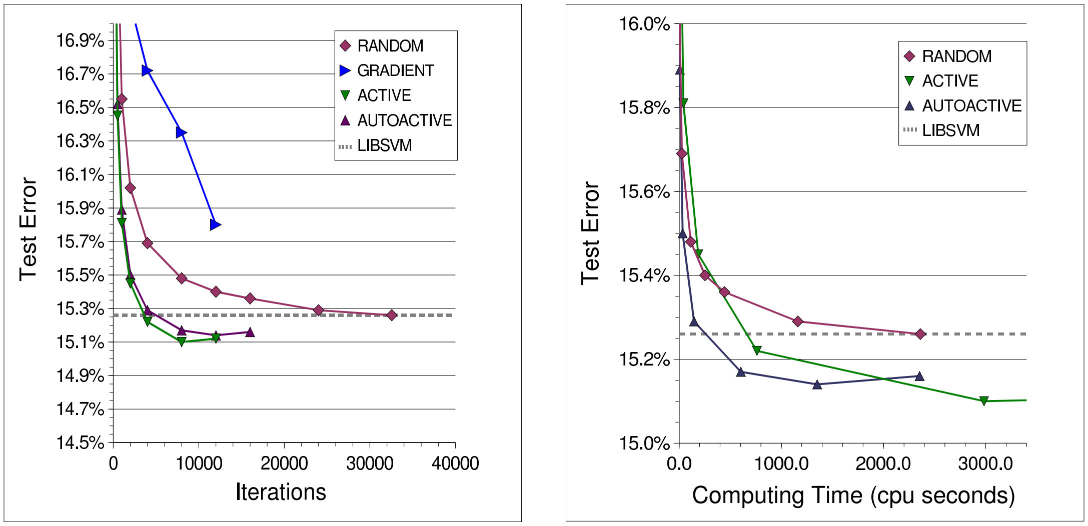


Figure 11: Comparing example selection criteria on the Adult data set. Measurements were performed on 65 runs using randomly selected training sets. The graphs show the error measured on the remaining testing examples as a function of the number of iterations and the computing time. The dashed line represents the LIBSVM test error under the same conditions.

Adult Data Set We first report experiments performed on the "Adult" data set. This data set provides a good indication of the relative performance of the Gradient and Active selection criteria under noisy conditions.

Reliable results were obtained by averaging experimental results measured for 65 random splits of the full data set into training and test sets. Paired tests indicate that test error differences of 0.25% on a single run are statistically significant at the 95% level. We conservatively estimate that average error differences of 0.05% are meaningful.

Figure 11 reports the average error rate measured on the test set as a function of the number of online iterations (left plot) and of the average computing time (right plot). Regardless of the training example selection method, all reported results were measured after performing the LASVM finishing step. More specifically, we run a predefined number of online iterations, save the LASVM state, perform the finishing step, measure error rates and number of support vectors, and restore the saved LASVM state before proceeding with more online iterations. Computing time includes the duration of the online iterations and the duration of the finishing step.

The GRADIENT example selection criterion performs very poorly on this noisy data set. A detailed analysis shows that most of the selected examples become support vectors with coefficient reaching the upper bound C. The ACTIVE and AUTOACTIVE criteria both reach smaller test error rates than those achieved by the SVM solution computed by LIBSVM. The error rates then seem to increase towards the error rate of the SVM solution (left plot). We believe indeed that continued iterations of the algorithm eventually yield the SVM solution.

Figure 12 relates error rates and numbers of support vectors. The RANDOM LASVM algorithm performs as expected: a single pass over all training examples replicates the accuracy and the num-

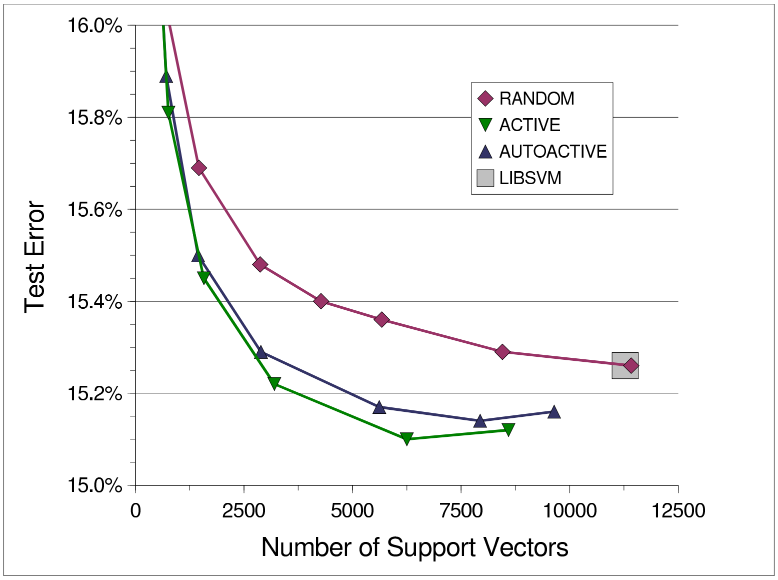


Figure 12: Comparing example selection criteria on the Adult data set. Test error as a function of the number of support vectors.

ber of support vectors of the LIBSVM solution. Both the ACTIVE and AUTOACTIVE criteria yield kernel classifiers with the same accuracy and much less support vectors. For instance, the AUTOACTIVE LASVM algorithm reaches the accuracy of the LIBSVM solution using 2500 support vectors instead of 11278. Figure 11 (right plot) shows that this result is achieved after 150 seconds only. This is about one fifteenth of the time needed to perform a full RANDOM LASVM epoch.[2]

Both the ACTIVE LASVM and AUTOACTIVE LASVM algorithms exceed the LIBSVM accuracy after a few iterations only. This is surprising because these algorithms only use the training labels of the few selected examples. They both outperform the LIBSVM solution by using only a small subset of the available training labels.

MNIST Data Set The comparatively clean MNIST data set provides a good opportunity to verify the behavior of the various example selection criteria on a problem with a much lower error rate.

Figure 13 compares the performance of the RANDOM, GRADIENT and ACTIVE criteria on the classification of digit “8” versus all other digits. The curves are averaged on 5 runs using different random seeds. All runs use the standard MNIST training and test sets. Both the GRADIENT and ACTIVE criteria perform similarly on this relatively clean data set. They require about as much computing time as RANDOM example selection to achieve a similar test error.

Adding ten percent label noise on the MNIST training data provides additional insight regarding the relation between noisy data and example selection criteria. Label noise was not applied to the testing set because the resulting measurement can be readily compared to test errors achieved by training SVMs without label noise. The expected test errors under similar label noise conditions can be derived from the test errors measured without label noise. Figure 14 shows the test errors achieved when 10% label noise is added to the training examples. The GRADIENT selection cri-

[2] The timing results reported in Figure 8 were measured on a faster computer.

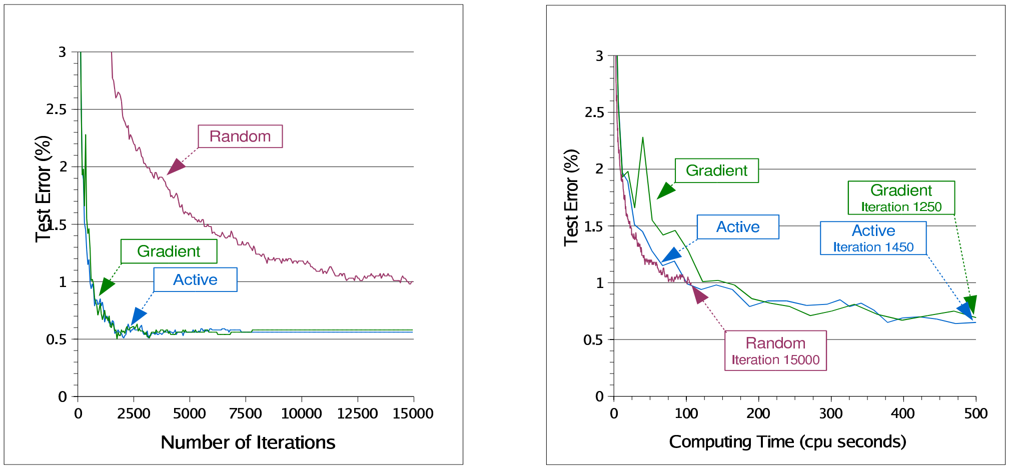


Figure 13: Comparing example selection criteria on the MNIST data set, recognizing digit "8" against all other classes. Gradient selection and Active selection perform similarly on this relatively noiseless task.

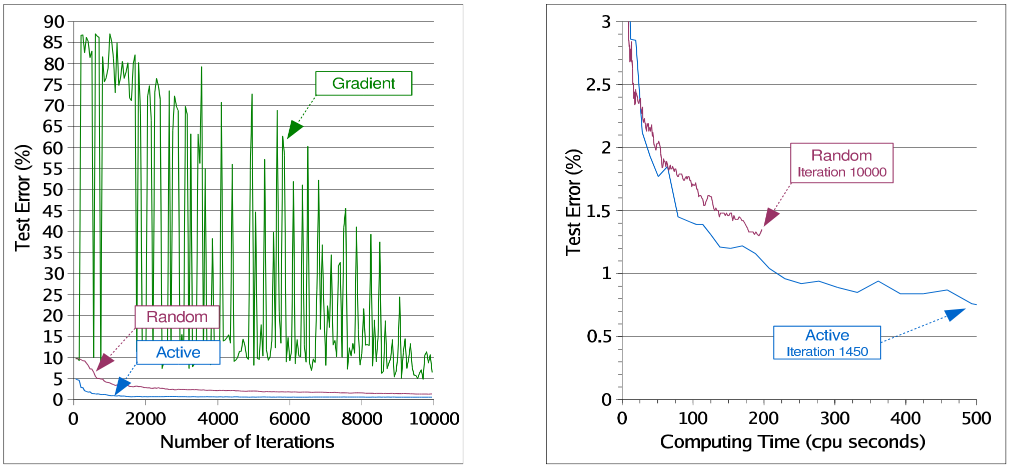


Figure 14: Comparing example selection criteria on the MNIST data set with 10% label noise on the training examples.

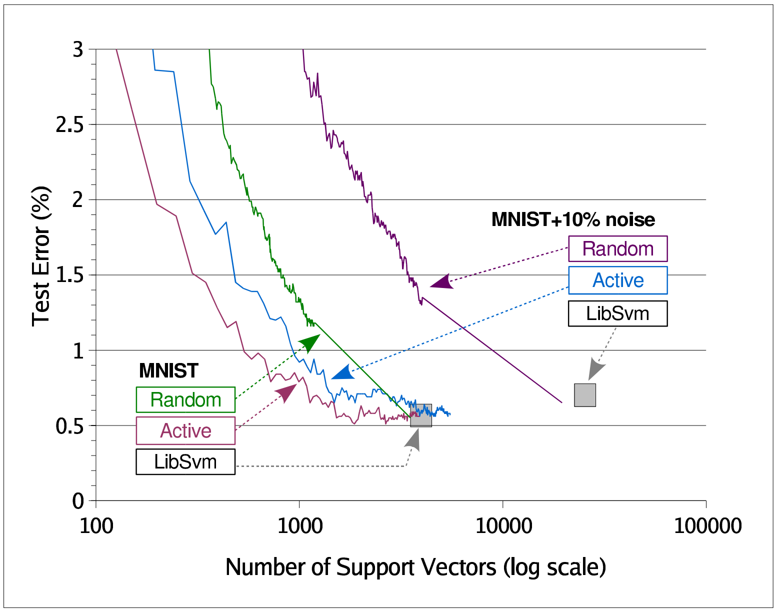


Figure 15: Comparing example selection criteria on the MNIST data set. Active example selection is insensitive to the artificial label noise.

terion causes a very chaotic convergence because it keeps selecting mislabelled training examples. The ACTIVE selection criterion is obviously undisturbed by the label noise.

Figure 15 summarizes error rates and number of support vectors for all noise conditions. In the presence of label noise on the training data, LIBSVM yields a slightly higher test error rate, and a much larger number of support vectors. The RANDOM LASVM algorithm replicates the LIBSVM results after one epoch. Regardless of the noise conditions, the ACTIVE LASVM algorithm reaches the accuracy and the number of support vectors of the LIBSVM solution obtained with clean training data. Although we have not been able to observe it on this data set, we expect that, after a very long time, the ACTIVE curve for the noisy training set converges to the accuracy and the number of support vectors achieved of the LIBSVM solution obtained for the noisy training data.

4.5 Online SVMs for Active Learning

The ACTIVE LASVM algorithm implements two dramatic speedups with respect to existing active learning algorithms such as (Campbell et al., 2000; Schohn and Cohn, 2000; Tong and Koller, 2000). First it chooses a query by sampling a small number of random examples instead of scanning all unlabelled examples. Second, it uses a single LASVM iteration after each query instead of fully retraining the SVM.

Figure 16 reports experiments performed on the Reuters and USPS data sets presented in table 7. The RETRAIN ACTIVE 50 and RETRAIN ACTIVE ALL select a query from 50 or all unlabeled examples respectively, and then retrain the SVM. The SVM solver was initialized with the solution from the previous iteration. The LASVM ACTIVE 50 and LASVM ACTIVE ALL do not retrain the SVM, but instead make a single LASVM iteration for each new labeled example.

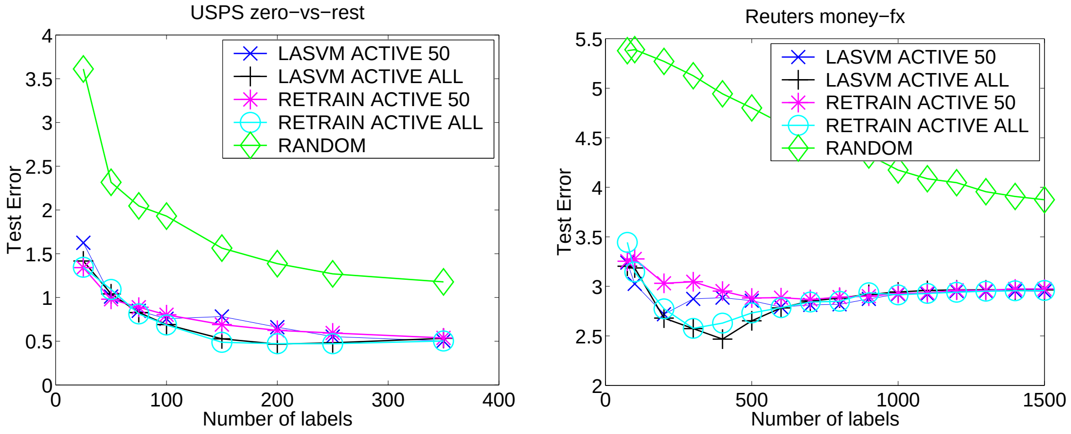


Figure 16: Comparing active learning methods on the USPS and Reuters data sets. Results are averaged on 10 random choices of training and test sets. Using LASVM iterations instead of retraining causes no loss of accuracy. Sampling $M = 50$ examples instead of searching all examples only causes a minor loss of accuracy when the number of labeled examples is very small.

All the active learning methods performed approximately the same, and were superior to random selection. Using LASVM iterations instead of retraining causes no loss of accuracy. Sampling $M = 50$ examples instead of searching all examples only causes a minor loss of accuracy when the number of labeled examples is very small. Yet the speedups are very significant: for 500 queried labels on the Reuters data set, the RETRAIN ACTIVE ALL, LASVM ACTIVE ALL, and LASVM ACTIVE 50 algorithms took 917 seconds, 99 seconds, and 9.6 seconds respectively.

5. Discussion

This work started because we observed that the data set sizes are quickly outgrowing the computing power of our calculators. One possible avenue consists of harnessing the computing power of multiple computers (Graf et al., 2005). Instead we propose learning algorithms that remain closely related to SVMs but require less computational resources. This section discusses their practical and theoretical implications.

5.1 Practical Significance

When we have access to an abundant source of training examples, the simple way to reduce the complexity of a learning algorithm consists of picking a random subset of training examples and running a regular training algorithm on this subset. Unfortunately this approach renounces the more accurate models that the large training set could afford. This is why we say, by reference to statistical efficiency, that an *efficient* learning algorithm should at least pay a brief look at every training example.

The online LASVM algorithm is very attractive because it matches the performance of a SVM trained on all the examples. More importantly, it achives this performance after a single epoch, faster than a SVM (figure 2) and using much less memory than a SVM (figure 6). This is very important in practice because modern data storage devices are most effective when the data is accessed sequentially.

Active Selection of the LASVM training examples brings two additional benefits for practical applications. It achieves equivalent performances with significantly less support vectors, further reducing the required time and memory. It also offers an obvious opportunity to parallelize the search for informative examples.

5.2 Informative Examples and Support Vectors

By suggesting that all examples should not be given equal attention, we first state that all training examples are not equally informative. This question has been asked and answered in various contexts (Fedorov, 1972; Cohn et al., 1990; MacKay, 1992). We also ask whether these differences can be exploited to reduce the computational requirements of learning algorithms. Our work answers this question by proposing algorithms that exploit these differences and achieve very competitive performances.

Kernel classifiers in general distinguish the few training examples named support vectors. Kernel classifier algorithms usually maintain an active set of potential support vectors and work by iterations. Their computing requirements are readily associated with the training examples that belong to the active set. Adding a training example to the active set increases the computing time associated with each subsequent iteration because they will require additional kernel computations involving this new support vector. Removing a training example from the active set reduces the cost of each subsequent iteration. However it is unclear how such changes affect the number of subsequent iterations needed to reach a satisfactory performance level.

Online kernel algorithms, such as the kernel perceptrons usually produce different classifiers when given different sequences of training examples. Section 3 proposes an online kernel algorithm that converges to the SVM solution after many epochs. The final set of support vectors is intrinsically defined by the SVM QP problem, regardless of the path followed by the online learning process. Intrinsic support vectors provide a benchmark to evaluate the impact of changes in the active set of current support vectors. Augmenting the active set with an example that is not an intrinsic support vector moderately increases the cost of each iteration without clear benefits. Discarding an example that is an intrinsic support vector incurs a much higher cost. Additional iterations will be necessary to recapture the missing support vector. Empirical evidence is presented in Section 3.5.

Nothing guarantees however that the most informative examples are the support vectors of the SVM solution. Bakır et al. (2005) interpret Steinwart's theorem (Steinwart, 2004) as an indication that the number of SVM support vectors is asymptotically driven by the examples located on the wrong side of the optimal decision boundary. Although such outliers might play a useful role in the construction of a decision boundary, it seems unwise to give them the bulk of the available computing time. Section 4 adds explicit example selection criteria to LASVM. The Gradient Selection Criterion selects the example most likely to cause a large increase of the SVM objective function. Experiments show that it prefers outliers over honest examples. The Active Selection Criterion bypasses the problem by choosing examples without regard to their labels. Experiments show that it leads to competitive test error rates after a shorter time, with less support vectors, and using only the labels of a small fraction of the examples.

5.3 Theoretical Questions

The appendix provides a comprehensive analysis of the convergence of the algorithms discussed in this contribution. Such convergence results are useful but limited in scope. This section underlines some aspects of this work that would vastly benefit from a deeper theoretical understanding.

- Empirical evidence suggests that a single epoch of the LASVM algorithm yields misclassification rates comparable with a SVM. We also know that LASVM exactly reaches the SVM solution after a sufficient number of epochs. Can we theoretically estimate the expected difference between the first epoch test error and the many epoch test error? Such results exist for well designed online learning algorithms based on stochastic gradient descent (Murata and Amari, 1999; Bottou and LeCun, 2005). Unfortunately these results do not directly apply to kernel classifiers. A better understanding would certainly suggest improved algorithms.

- Test error rates are sometimes improved by active example selection. In fact this effect has already been observed in the active learning setups (Schohn and Cohn, 2000). This small improvement is difficult to exploit in practice because it requires very sensitive early stopping criteria. Yet it demands an explanation because it seems that one gets a better performance by using less information. There are three potential explanations: (*i*) active selection works well on unbalanced data sets because it tends to pick equal number of examples of each class (Schohn and Cohn, 2000), (*ii*) active selection improves the SVM loss function because it discards distant outliers, (*iii*) active selection leads to more sparse kernel expansions with better generalization abilities (Cesa-Bianchi et al., 2005). These three explanations may be related.

- We know that the number of SVM support vectors scales linearly with the number of examples (Steinwart, 2004). Empirical evidence suggests that active example selection yields transitory kernel classifiers that achieve low error rates with much less support vectors. What is the scaling law for this new number of support vectors?

- What is the minimal computational cost for learning n independent examples and achieving “optimal” test error rates? The answer depends of course of how we define these “optimal” test error rates. This cost intuitively scales at least linearly with n because one must pay a look at each example to fully exploit them. The present work suggest that this cost might be smaller than n times the reduced number of support vectors achievable with the active learning technique. This range is consistent with previous work showing that stochastic gradient algorithms can train a fixed capacity model in linear time (Bottou and LeCun, 2005). Learning seems to be much easier than computing the optimum of the empirical loss.

5.4 Future Directions

Progress can also be achieved along less arduous directions.

- Section 3.5 suggests that better convergence speed could be attained by cleverly modulating the number of calls to REPROCESS during the online iterations. Simple heuristics might go a long way.

- Section 4.3 suggests a heuristic to adapt the sampling size for the randomized search of informative training examples. This AUTOACTIVE heuristic performs very well and deserves further investigation.
- Sometimes one can generate a very large number of training examples by exploiting known invariances. Active example selection can drive the generation of examples. This idea was suggested in (Loosli et al., 2004) for the SimpleSVM.

6. Conclusion

This work explores various ways to speedup kernel classifiers by asking which examples deserve more computing time. We have proposed a novel online algorithm that converges to the SVM solution. LASVM reliably reaches competitive accuracies after performing a single pass over the training examples, outspeeding state-of-the-art SVM solvers. We have then shown how active example selection can yield faster training, higher accuracies and simpler models using only a fraction of the training examples labels.

Acknowledgments

Part of this work was funded by NSF grant CCR-0325463. We also thank Eric Cosatto, Hans-Peter Graf, C. Lee Giles and Vladimir Vapnik for their advice and support, Ronan Collobert and Chih-Jen Lin for thoroughly checking the mathematical appendix, and Sathiya Keerthi for pointing out reference (Takahashi and Nishi, 2003).

Appendix A. Convex Programming with Witness Families

This appendix presents theoretical elements about convex programming algorithms that rely on successive direction searches. Results are presented for the case where directions are selected from a well chosen finite pool, like SMO (Platt, 1999), and for the stochastic algorithms, like the online and active SVM discussed in the body of this contribution.

Consider a compact convex subset $\mathcal{F}$ of $\mathbb{R}^n$ and a concave function f defined on $\mathcal{F}$. We assume that f is twice differentiable with continuous derivatives. This appendix discusses the maximization of function f over set F:

$$\max_{x\in\mathcal{F}} f(x). \tag{14}$$

This discussion starts with some results about feasible directions. Then it introduces the notion of witness family of directions which leads to a more compact characterization of the optimum. Finally it presents maximization algorithms and establishes their convergence to approximate solutions

A.1 Feasible Directions

Notations Given a point $x \in \mathcal{F}$ and a direction $u \in \mathbb{R}^n_* = \mathbb{R}^n$, let

$$\begin{aligned} \phi(x,u) &= \max\{\lambda \geq 0 \,|\, x+\lambda u \in \mathcal{F}\} \\ f^*(x,u) &= \max\{f(x+\lambda u), x+\lambda u \in \mathcal{F}\}. \end{aligned}$$

In particular we write $\phi(x,0) = \infty$ and $f^*(x,0) = f(x)$.

Definition 1 *The cone of feasible directions in $x \in \mathcal{F}$ is the set*

$$\mathcal{D}_X = \{u \in \mathbb{R}^n \,|\, \phi(x,u) > 0\}.$$

All the points $x + \lambda u$, $0 \leq \lambda \leq \phi(x,u)$ belong to $\mathcal{F}$ because $\mathcal{F}$ is convex. Intuitively, a direction $u \neq 0$ is feasible in x when we can start from x and make a little movement along direction u without leaving the convex set $\mathcal{F}$.

Proposition 2 *Given $x \in \mathcal{F}$ and $u \in \mathbb{R}^n$,*

$$f^*(x,u) > f(x) \quad \iff \quad \left\{ \begin{array}{l} u'\nabla f(x) > 0 \\ u \in \mathcal{D}_X. \end{array} \right.$$

Proof Assume $f^*(x,u) > f(x)$. Direction $u \neq 0$ is feasible because the maximum $f^*(x,u)$ is reached for some $0 < \lambda^* \leq \phi(x,u)$. Let $\nu \in [0,1]$. Since set $\mathcal{F}$ is convex, $x + \nu\lambda^* u \in \mathcal{F}$. Since function f is concave, $f(x + \nu\lambda^* u)) \geq (1-\nu) f(x) + \nu f^*(x,u)$. Writing a first order expansion when $\nu \to 0$ yields $\lambda^* u'\nabla f(x) \geq f^*(x,u) - f(x) > 0$. Conversely, assume $u'\nabla f(x) > 0$ and $u \neq 0$ is a feasible direction. Recall $f(x+\lambda u) = f(x) + \lambda u'\nabla f(x) + o(\lambda)$. Therefore we can choose $0 < \lambda_0 \leq \phi(x,u)$ such that $f(x+\lambda_0 u) > f(x) + \lambda_0 u'\nabla f(x)/2$. Therefore $f^*(x,u) \geq f(x+\lambda_0 u) > f(x)$. ■

Theorem 3 (Zoutendijk (1960) page 22) *The following assertions are equivalent:*

i) *x is a solution of problem (14).*
ii) $\forall u \in \mathbb{R}^n \quad f^*(x,u) \leq f(x)$.
iii) $\forall u \in \mathcal{D}_X \quad u'\nabla f(x) \leq 0$.

Proof The equivalence between assertions (ii) and (iii) results from proposition 2. Assume assertion (i) is true. Assertion (ii) is necessarily true because $f^*(u,x) \leq \max_{\mathcal{F}} f = f(x)$. Conversely, assume assertion (i) is false. Then there is $y \in \mathcal{F}$ such that $f(y) > f(x)$. Therefore $f^*(x, y-x) > f(x)$ and assertion (ii) is false. ■

A.2 Witness Families

We now seek to improve this theorem. Instead of considering all feasible directions in $\mathbb{R}^n$, we wish to only consider the feasible directions from a smaller set $\mathcal{U}$.

Proposition 4 *Let $x \in \mathcal{F}$ and $v_1 \ldots v_k \in \mathcal{D}_X$ be feasible directions. Every positive linear combination of $v_1 \ldots v_k$ (i.e. a linear combination with positive coefficients) is a feasible direction.*

Proof Let u be a positive linear combination of the v_i. Since the v_i are feasible directions there are $y_i = x + \lambda_i v_i \in \mathcal{F}$, and u can be written as $\sum_i \gamma_i (y_i - x)$ with $\gamma_i \geq 0$. Direction u is feasible because the convex $\mathcal{F}$ contains $(\sum \gamma_i y_i) / (\sum \gamma_i) = x + (1/\sum \gamma_i)\, u$. ■

Definition 5 *A set of directions $\mathcal{U} \subset \mathbb{R}^n_*$ is a "witness family for $\mathcal{F}$" when, for any point $x \in \mathcal{F}$, any feasible direction $u \in \mathcal{D}_X$ can be expressed as a positive linear combination of a finite number of feasible directions $v_j \in \mathcal{U} \cap \mathcal{D}_x$.*

This definition directly leads to an improved characterization of the optima.

Theorem 6 *Let $\mathcal{U}$ be a witness family for convex set $\mathcal{F}$.*
The following assertions are equivalent:

i) x is a solution of problem (14).
ii) $\forall u \in \mathcal{U} \quad f^*(x,u) \leq f(x)$.
iii) $\forall u \in \mathcal{U} \cap \mathcal{D}_X \quad u'\nabla f(x) \leq 0$.

Proof The equivalence between assertions (ii) and (iii) results from proposition 2. Assume assertion (i) is true. Theorem 3 implies that assertion (ii) is true as well. Conversely, assume assertion (i) is false. Theorem 3 implies that there is a feasible direction $u \in \mathbb{R}^n$ on point x such that $u'\nabla f(x) > 0$. Since $\mathcal{U}$ is a witness family, there are positive coefficients $\gamma_1 \ldots \gamma_k$ and feasible directions $v_1, \ldots, v_k \in \mathcal{U} \cap \mathcal{D}_X$ such that $u = \sum \gamma_i v_i$. We have then $\sum \gamma_j v'_j \nabla f(x) > 0$. Since all coefficients γ_j are positive, there is at least one term j_0 such that $v'_{j_0}\nabla f(x) > 0$. Assertion (iii) is therefore false. ■

The following proposition provides an example of witness family for the convex domain $\mathcal{F}_\mathrm{s}$ that appears in the SVM QP problem (5).

Proposition 7 *Let $(e_1 \ldots e_n)$ be the canonical basis of $\mathbb{R}^n$. Set $\mathcal{U}_\mathrm{s} = \{e_i - e_j, i \neq j\}$ is a witness family for convex set $\mathcal{F}_\mathrm{s}$ defined by the constraints*

$$x \in \mathcal{F}_\mathrm{s} \quad \Longleftrightarrow \quad \begin{cases} \forall\, i \quad A_i \leq x_i \leq B_i \\ \sum_i x_i = 0. \end{cases}$$

Proof Let $u \in \mathbb{R}^n_*$ be a feasible direction in $x \in \mathcal{F}_\mathrm{s}$. Since u is a feasible direction, there is $\lambda > 0$ such that $y = x + \lambda u \in \mathcal{F}_\mathrm{s}$. Consider the subset $\mathcal{B} \subset \mathcal{F}_\mathrm{s}$ defined by the constraints

$$z \in \mathcal{B} \quad \Leftrightarrow \quad \begin{cases} \forall\, i,\, A_i \leq \min(x_i, y_i) \leq z_i \leq \max(x_i, y_i) \leq B_i \\ \sum_i z_i = 0. \end{cases}$$

Let us recursively define a sequence of points $z(j) \in \mathcal{B}$. We start with $z(0) = x \in \mathcal{B}$. For each $t \geq 0$, we define two sets of coordinate indices $I_t^+ = \{i \,|\, z_i(t) < y_i\}$ and $I_t^- = \{j \,|\, z_j(t) > y_j\}$. The recursion stops if either set is empty. Otherwise, we choose $i \in I_t^+$ and $j \in I_t^-$ and define $z(t+1) = z(t) + \gamma(t)\, v(t) \in \mathcal{B}$ with $v(t) = e_i - e_j \in \mathcal{U}_\mathrm{s}$ and $\gamma(t) = \min(y_i - z_i(t), z_j(t) - y_j) > 0$. Intuitively, we move towards y along direction $v(t)$ until we hit the boundaries of set $\mathcal{B}$.

Each iteration removes at least one of the indices i or j from sets I_t^+ and I_t^-. Eventually one of these sets gets empty and the recursion stops after a finite number k of iterations. The other set is also empty because

$$\sum_{i \in I_k^+} |y_i - z_i(k)| - \sum_{i \in I_k^-} |y_i - z_i(k)| \;=\; \sum_{i=1}^{n} y_i - z_i(k) \;=\; \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} z_i(k) \;=\; 0.$$

Therefore $z(k) = y$ and $\lambda u = y - x = \sum_t \gamma(t)\, v(t)$. Moreover the $v(t)$ are feasible directions on x because $v(t) = e_i - e_j$ with $i \in I_t^+ \subset I_0^+$ and $j \in I_t^- \subset I_0^-$. ■

Assertion (iii) in Theorem 6 then yields the following necessary and sufficient optimality criterion for the SVM QP problem (5):

$$\forall\,(i,j) \in \{1\dots n\}^2 \qquad x_i < B_i \text{ and } x_j > A_j \;\Rightarrow\; \frac{\partial f}{\partial x_i}(x) - \frac{\partial f}{\partial x_j}(x) \leq 0.$$

Different constraint sets call for different choices of witness family. For instance, it is sometimes useful to disregard the equality constraint in the SVM polytope $\mathcal{F}_s$. Along the lines of proposition 7, it is quite easy to prove that $\{\pm e_i,\, i = 1\dots n\}$ is a witness family. Theorem 6 then yields an adequate optimality criterion.

A.3 Finite Witness Families

This section deals with *finite witness families*. Theorem 9 shows that $\mathcal{F}$ is necessarily a convex polytope, that is a bounded set defined by a finite number of linear of linear equality and inequality constraints (Schrijver, 1986).

Proposition 8 *Let $\mathcal{C}_x = \{x + u\,,\, u \in \mathcal{D}_x\}$ for $x \in \mathcal{F}$. Then $\mathcal{F} = \bigcap_{x\in\mathcal{F}} \mathcal{C}_x$.*

Proof We first show that $\mathcal{F} \subset \bigcap_{x\in\mathcal{F}} \mathcal{C}_x$. Indeed $\mathcal{F} \subset \mathcal{C}_x$ for all x because every point $z \in \mathcal{F}$ defines a feasible direction $z - x \in \mathcal{D}_x$.

Conversely, Let $z \in \bigcap_{x\in\mathcal{F}} \mathcal{C}_x$ and assume that z does not belong to $\mathcal{F}$. Let $\hat{z}$ be the projection of z on $\mathcal{F}$. We know that $z \in \mathcal{C}_{\hat{z}}$ because $z \in \bigcap_{x\in\mathcal{F}} \mathcal{C}_x$. Therefore $z - \hat{z}$ is a feasible direction in $\hat{z}$. Choose $0 < \lambda < \phi(\hat{z}, z - \hat{z})$. We know that $\lambda < 1$ because z does not belong to $\mathcal{F}$. But then $\hat{z} + \lambda(z - \hat{z}) \in \mathcal{F}$ is closer to z than $\hat{z}$. This contradicts the definition of the projection $\hat{z}$. ■

Theorem 9 *Let $\mathcal{F}$ be a bounded convex set.*
If there is a finite witness family for $\mathcal{F}$, then $\mathcal{F}$ is a convex polytope.[3]

Proof Consider a point $x \in \mathcal{F}$ and let $\{v_1 \dots v_k\} = \mathcal{U} \cap \mathcal{D}_x$. Proposition 4 and definition 5 imply that $\mathcal{D}_x$ is the polyhedral cone $\{z = \sum \gamma_i v_i,\, \gamma_i \geq 0\}$ and can be represented (Schrijver, 1986) by a finite number of linear equality and inequality constraints of the form $nz \leq 0$ where the directions n are unit vectors. Let $\mathcal{K}_x$ be the set of these unit vectors. Equality constraints arise when the set $\mathcal{K}_x$ contains both n and $-n$. Each set $\mathcal{K}_x$ depends only on the subset $\{v_1 \dots v_k\} = \mathcal{U} \cap \mathcal{D}_x$ of feasible witness directions in x. Since the finite set $\mathcal{U}$ contains only a finite number of potential subsets, there is only a finite number of distinct sets $\mathcal{K}_x$.

Each set $\mathcal{C}_x$ is therefore represented by the constraints $nz \leq nx$ for $n \in \mathcal{K}_x$. The intersection $\mathcal{F} = \bigcap_{x\in\mathcal{F}} \mathcal{C}_x$ is then defined by all the constraints associated with $\mathcal{C}_x$ for any $x \in \mathcal{F}$. These constraints involve only a finite number of unit vectors n because there is only a finite number of distinct sets $\mathcal{K}_x$.

Inequalities defined by the same unit vector n can be summarized by considering only the most restrictive right hand side. Therefore $\mathcal{F}$ is described by a finite number of equality and inequality constraints. Since $\mathcal{F}$ is bounded, it is a polytope. ■

A convex polytope comes with useful continuity properties.

Proposition 10 *Let $\mathcal{F}$ be a polytope, and let $u \in \mathbb{R}^n$ be fixed.*
Functions $x \mapsto \phi(x,u)$ and $x \mapsto f^*(x,u)$ are uniformly continous on $\mathcal{F}$.

Proof The polytope $\mathcal{F}$ is defined by a finite set of constraints $n\,x \leq b$. Let $\mathcal{K}_\mathcal{F}$ be the set of pairs (n,b) representing these constraints. Function $x \mapsto \phi(x,u)$ is a continuous on $\mathcal{F}$ because we can write:

$$\phi(x,u) = \min\left\{ \frac{b-n\,x}{n\,u} \quad \text{for all } (n,b) \in \mathcal{K}_\mathcal{F} \text{ such that } n\,u > 0 \right\}.$$

Function $x \mapsto \phi(x,u)$ is uniformly continuous because it is continuous on the compact $\mathcal{F}$.

Choose $\varepsilon > 0$ and let $x,y \in \mathcal{F}$. Let the maximum $f^*(x,u)$ be reached in $x+\lambda^*u$ with $0 \leq \lambda^* \leq \phi(x,u)$. Since f is uniformly continous on compact $\mathcal{F}$, there is $\eta > 0$ such that $|f(x+\lambda^*u) - f(y+\lambda'u)| < \varepsilon$ whenever $\|x-y+(\lambda^*-\lambda')u\| < \eta(1+\|u\|)$. In particular, it is sufficient to have $\|x-y\| < \eta$ and $|\lambda^*-\lambda'| < \eta$. Since ϕ is uniformly continuous, there is $\tau > 0$ such that $|\phi(y,u)-\phi(x,u)| < \eta$ whenever $\|x-y\| < \tau$. We can then select $0 \leq \lambda' \leq \phi(y,u)$ such that $|\lambda^*-\lambda'| < \eta$. Therefore, when $\|x-y\| < \min(\eta,\tau)$, $f^*(x,u) = f(x+\lambda^*u) \leq f(y+\lambda'u)+\varepsilon \leq f^*(y,u)+\varepsilon$.

By reversing the roles of x and y in the above argument, we can similary establish that $f^*(y,u) \leq f^*(x,u)+\varepsilon$ when $\|x-y\| \leq \min(\eta,\tau)$. Function $x \mapsto f^*(x,u)$ is therefore uniformly continuous on $\mathcal{F}$. ■

A.4 Stochastic Witness Direction Search

Each iteration of the following algorithm randomly chooses a feasible witness direction and performs an optimization along this direction. The successive search directions u_t are randomly selected (step 2a) according to some distribution P_t defined on $\mathcal{U}$. Distribution P_t possibly depends on values observed before time t.

```
Stochastic Witness Direction Search (WDS)
  1) Find an initial feasible point $x_0 \in \mathcal{F}$.
  2) For each $t = 1,2,\ldots,$
       2a) Draw a direction $u_t \in \mathcal{U}$ from a distribution $P_t$
       2b) If   $u \in \mathcal{D}_{x_{t-1}}$  and  $u_t'\nabla f(x_{t-1}) > 0$ ,
                 $x_t \leftarrow$ argmax $f(x)$ under $x \in \{x_{t-1}+\lambda u_t \in \mathcal{F}, \lambda \geq 0\}$
           otherwise
                 $x_t \leftarrow x_{t-1}$.
```

Clearly the Stochastic WDS algorithm does not work if the distributions P_t always give probability zero to important directions. On the other hand, convergence is easily established if all feasible directions can be drawn with non zero minimal probability at any time.

[3] We believe that the converse of Theorem 9 is also true.

Theorem 11 *Let f be a concave function defined on a compact convex set $\mathcal{F}$, differentiable with continuous derivatives. Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$, and let the sequence x_t be defined by the Stochastic WDS algorithm above. Further assume there is $\pi > 0$ such that $P_t(u) > \pi$ for all $u \in \mathcal{U} \cap \mathcal{D}_{x_{t-1}}$. All accumulation points of the sequence x_t are then solutions of problem (14) with probability* 1.

Proof We want to evaluate the probability of event Q comprising all sequences of selected directions $(u_1, u_2, \dots)$ leading to a situation where x_t has an accumulation point x^* that is not a solution of problem (14).

For each sequence of directions $(u_1, u_2, \dots)$, the sequence $f(x_t)$ is increasing and bounded. It converges to $f^* = \sup_t f(x_t)$. We have $f(x^*) = f^*$ because f is continuous. By Theorem 6, there is a direction $u \in \mathcal{U}$ such that $f^*(x^*, u) > f^*$ and $\phi(x^*, u) > 0$. Let x_{k_t} be a subsequence converging to x^*. Thanks to the continuity of ϕ, f^* and ∇f, there is a t_0 such that $f^*(x_{k_t}, u) > f^*$ and $\phi(x_{k_t}, u) > 0$ for all $k_t > t_0$.

Choose $\varepsilon > 0$ and let $Q_T \subset Q$ contain only sequences of directions such that $t_0 = T$. For any $k_t > T$, we know that $\phi(x_{k_t}, u) > 0$ which means $u \in \mathcal{U} \cap \mathcal{D}_{x_{k_t}}$. We also know that $u_{k_t} \neq u$ because we would otherwise obtain a contradiction $f(x_{k_t+1}) = f^*(x_{k_t}, u) > f^*$. The probability of selecting such a u_{k_t} is therefore smaller than $(1 - \pi)$. The probability that this happens simultaneously for N distinct $k_t \geq T$ is smaller than $(1 - \pi)^N$ for any N. We get $P(Q_T) \leq \varepsilon / T^2$ by choosing N large enough.

Then we have $P(Q) = \sum_T P(Q_T) \leq \varepsilon \left(\sum_T 1/T^2 \right) = K\varepsilon$. Hence $P(Q) = 0$ because we can choose ε as small as we want, We can therefore assert with probability 1 that all accumulation points of sequence x_t are solutions. ■

This condition on the distributions P_t is unfortunately too restrictive. The PROCESS and REPROCESS iterations of the Online LASVM algorithm (Section 3.2) only exploit directions from very specific subsets.

On the other hand, the Online LASVM algorithm only ensures that any remaining feasible direction at time T will eventually be selected with probability 1. Yet it is challenging to mathematically express that there is no coupling between the subset of time points t corresponding to a subsequence converging to a particular accumulation point, and the subset of time points t corresponding to the iterations where specific feasible directions are selected.

This problem also occurs in the deterministic Generalized SMO algorithm (Section 3.1). An asymptotic convergence proof (Lin, 2001) only exist for the important case of the SVM QP problem using a specific direction selection strategy. Following Keerthi and Gilbert (2002), we bypass this technical difficulty by defining a notion of approximate optimum and proving convergence in finite time. It is then easy to discuss the properties of the limit point.

A.5 Approximate Witness Direction Search

Definition 12 *Given a finite witness family $\mathcal{U}$ and the tolerances $\kappa > 0$ and $\tau > 0$, we say that x is a $\kappa\tau$-approximate solution of problem (14) when the following condition is verified:*

$$\forall u \in \mathcal{U}, \quad \phi(x, u) \leq \kappa \ \text{ or } \ u' \nabla f(x) \leq \tau.$$

A vector $u \in \mathbb{R}_n$ such that $\phi(x, u) > \kappa$ and $u' \nabla f(x) > \tau$ is called a $\kappa\tau$-violating direction in point x.

This definition is inspired by assertion (*iii*) in Theorem 6. The definition demands a *finite* witness family because this leads to proposition 13 establishing that $\kappa\tau$-approximate solutions indicate the location of actual solutions when κ and τ tend to zero.

Proposition 13 *Let $\mathcal{U}$ be a finite witness family for bounded convex set $\mathcal{F}$. Consider a sequence $x_t \in \mathcal{F}$ of $\kappa_t\tau_t$-approximate solutions of problem (14) with $\tau_t \to 0$ and $\kappa_t \to 0$. The accumulation points of this sequence are solutions of problem (14).*

Proof Consider an accumulation point x^* and a subsequence x_{k_t} converging to x^*. Define function

$$(x,\tau,\kappa) \;\; \mapsto \;\; \psi(x,\tau,\kappa,u) = \big(u'\nabla f(x) - \tau\big) \, \max\{0, \phi(x,u) - \kappa\}$$

such that u is a $\kappa\tau$-violating direction if and only if $\psi(x,\kappa,\tau,u) > 0$. Function ψ is continuous thanks to Theorem 9, proposition 10 and to the continuity of ∇f. Therefore, we have $\psi(x_{k_t}, \kappa_{k_t}, \tau_{k_t}, u) \leq 0$ for all $u \in \mathcal{U}$. Taking the limit when $k_t \to \infty$ gives $\psi(x^*, 0, 0, u) \leq 0$ for all $u \in \mathcal{U}$. Theorem 6 then states that x^* is a solution. ■

The following algorithm introduces the two tolerance parameters $\tau > 0$ and $\kappa > 0$ into the Stochastic Witness Direction Search algorithm.

```
Approximate Stochastic Witness Direction Search
   1) Find an initial feasible point $x_0 \in \mathcal{F}$.
   2) For each $t = 1, 2, \ldots$,
        2a) Draw a direction $u_t \in \mathcal{U}$ from a probability distribution $P_t$
        2b) If $u_t$ is a $\kappa\tau$-violating direction,
                 $x_t \leftarrow \text{argmax } f(x) \text{ under } x \in \{x_{t-1} + \lambda u_t \in \mathcal{F},\, \lambda \geq 0\}$
            otherwise
                 $x_t \leftarrow x_{t-1}$.
```

The successive search directions u_t are drawn from some unspecified distributions P_t defined on $\mathcal{U}$. Proposition 16 establishes that this algorithm always converges to some $x^* \in \mathcal{F}$ after a finite number of steps, regardless of the selected directions (u_t). The proof relies on the two intermediate results that generalize a lemma proposed by Keerthi and Gilbert (2002) in the case of quadratic functions.

Proposition 14 *If u_t is a $\kappa\tau$-violating direction in x_{t-1},*

$$\phi(x_t, u_t)\, u_t' \nabla f(x_t) = 0.$$

Proof Let the maximum $f(x_t) = f^*(x_{t-1}, u_t)$ be attained in $x_t = x_{t-1} + \lambda^* u_t$ with $0 \leq \lambda^* \leq \phi(x_{t-1}, u_t)$. We know that $\lambda^* \neq 0$ because u_t is $\kappa\tau$-violating and proposition 2 implies $f^*(x_{t-1}, u_t) > f(x_{t-1})$. If λ^* reaches its upper bound, $\phi(x_t, u_t) = 0$. Otherwise x_t is an unconstrained maximum and $u_t' \nabla f(x_t) = 0$. ■

Proposition 15 *There is a constant $K > 0$ such that*

$$\forall t \,, \quad f(x_t) - f(x_{t-1}) \;\; \geq \;\; K \, \|x_t - x_{t-1}\|.$$

Proof The relation is obvious when u_t is not a $\kappa\tau$-violating direction in x_{t-1}. Otherwise let the maximum $f(x_t) = f^*(x_{t-1}, u_t)$ be attained in $x_t = x_{t-1} + \lambda^* u_t$.
Let $\lambda = \nu\lambda^*$ with $0 < \nu \leq 1$. Since x_t is a maximum,

$$f(x_t) - f(x_{t-1}) = f(x_{t-1} + \lambda^* u_t) - f(x_{t-1}) \geq f(x_{t-1} + \lambda u_t) - f(x_{t-1}).$$

Let H be the maximum over $\mathcal{F}$ of the norm of the Hessian of f.
A Taylor expansion with the Cauchy remainder gives

$$\left|\ f(x_{t-1} + \lambda u_t) - f(x_{t-1}) - \lambda u_t' \nabla f(x_{t-1})\ \right| \leq \frac{1}{2}\lambda^2 \|u_t\|^2 H$$

or, more specifically,

$$f(x_{t-1} + \lambda u_t) - f(x_{t-1}) - \lambda u_t' \nabla f(x_{t-1}) \ \geq\ -\frac{1}{2}\lambda^2 \|u_t\|^2 H.$$

Combining these inequalities yields

$$f(x_t) - f(x_{t-1}) \geq f(x_{t-1} + \lambda u_t) - f(x_{t-1}) \geq \lambda u_t' \nabla f(x_{t-1}) - \frac{1}{2}\lambda^2 \|u_t\|^2 H.$$

Recalling $u_t' \nabla f(x_{t-1}) > \tau$, and $\lambda\|u_t\| = \nu\|x_t - x_{t-1}\|$, we obtain

$$f(x_t) - f(x_{t-1}) \geq \|x_t - x_{t-1}\| \left(\nu \frac{\tau}{U} - \nu^2 \frac{1}{2} DH \right)$$

where $U = \max_{\mathcal{U}} \|u\|$ and D is the diameter of the compact convex $\mathcal{F}$.
Choosing $\nu = \min\left(1, \dfrac{\tau}{UDH}\right)$ then gives the desired result. ■

Proposition 16 *Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$. The Approximate Stochastic WDS algorithm converges to some $x^* \in \mathcal{F}$ after a finite number of steps.*

Proof Sequence $f(x_t)$ converges because it is increasing and bounded. Therefore it satisfies Cauchy's convergence criterion:

$$\begin{aligned} &\forall \varepsilon > 0, \exists t_0, \forall t_2 > t_1 > t_0, \\ &\qquad f(x_{t_2}) - f(x_{t_1}) = \sum_{t_1 < t \leq t_2} f(x_t) - f(x_{t-1}) < \varepsilon. \end{aligned}$$

Using proposition 15, we can write

$$\begin{aligned} &\forall \varepsilon > 0, \exists t_0, \forall t_2 > t_1 > t_0, \\ &\qquad \|x_{t_2} - x_{t_1}\| \leq \sum_{t_1 < t \leq t_2} \|x_t - x_{t-1}\| \leq \sum_{t_1 < t \leq t_2} \frac{f(x_t) - f(x_{t-1})}{K} < \frac{\varepsilon}{K}. \end{aligned}$$

Therefore sequence x_t satisfies Cauchy's condition and converges to some $x^* \in \mathcal{F}$.

Assume this convergence does not occur in a finite time. Since $\mathcal{U}$ is finite, the algorithm exploits at least one direction $u \in \mathcal{U}$ an infinite number of times. Therefore there is a strictly increasing sequence of positive indices k_t such that $u_{k_t} = u$ is $\kappa\tau$-violating in point $x_{k_t - 1}$. We have then

$\phi(x_{k_t-1},u) > \kappa$ and $u'\nabla f(x_{k_t-1}) > \tau$. By continuity we have $\phi(x^*,u) \geq \kappa$ and $u'\nabla f(x^*) \geq \tau$. On the other hand, proposition 14 states that $\phi(x_{k_t},u)\,u'\nabla f(x_{k_t}) = 0$. By continuity when $t \to 0$, we obtain the contradiction $\phi(x^*,u)\,u'\nabla f(x^*) = 0$. ■

In general, proposition 16 only holds for $\kappa > 0$ and $\tau > 0$. Keerthi and Gilbert (2002) assert a similar property for $\kappa = 0$ and $\tau > 0$ in the case of SVMs only. Despite a mild flaw in the final argument of the initial proof, this assertion is correct (Takahashi and Nishi, 2003).

Proposition 16 does not prove that the limit x^* is related to the solution of the optimization problem (14). Additional assumptions on the direction selection step are required. Theorem 17 addresses the deterministic case by considering trivial distributions P_t that always select a $\kappa\tau$-violating direction if such directions exist. Theorem 18 addresses the stochastic case under mild conditions on the distribution P_t.

Theorem 17 *Let the concave function f defined on the compact convex set $\mathcal{F}$ be twice differentiable with continuous second derivatives. Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$, and let the sequence x_t be defined by the Approximate Stochastic WDS algorithm above. Assume that step (2a) always selects a $\kappa\tau$-violating direction in x_{t-1} if such directions exist. Then x_t converges to a $\kappa\tau$-approximate solution of problem (14) after a finite number of steps.*

Proof Proposition 16 establishes that there is t_0 such that $x_t = x^*$ for all $t \geq t_0$. Assume there is a $\kappa\tau$-violating direction in x^*. For any $t > t_0$, step (2a) always selects such a direction, and step (2b) makes x_t different from $x_{t-1} = x^*$. This contradicts the definition of t_0. Therefore there are no $\kappa\tau$-violating direction in x^* and x^* is a $\kappa\tau$-approximate solution. ■

Example (SMO) The SMO algorithm (Section 3.1) is[4] an Approximate Stochastic WDS that always selects a $\kappa\tau$-violating direction when one exists. Therefore Theorem 17 applies.

Theorem 18 *Let the concave function f defined on the compact convex set $\mathcal{F}$ be twice differentiable with continuous second derivatives. Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$, and let the sequence x_t be defined by the Approximate Stochastic WDS algorithm above. Let p_t be the conditional probability that u_t is $\kappa\tau$-violating in x_{t-1} given that $\mathcal{U}$ contains such directions. Assume that* $\limsup p_t > 0$. *Then x_t converges with probability one to a $\kappa\tau$-approximate solution of problem (14) after a finite number of steps.*

Proof Proposition 16 establishes that for each sequence of selected directions u_t, there is a time t_0 and a point $x^* \in \mathcal{F}$ such that $x_t = x^*$ for all $t \geq t_0$. Both t_0 and x^* depend on the sequence of directions $(u_1, u_2, \dots)$.

We want to evaluate the probability of event Q comprising all sequences of directions $(u_1, u_2, \dots)$ leading to a situation where there are $\kappa\tau$-violating directions in point x^*. Choose $\varepsilon > 0$ and let $Q_T \subset Q$ contain only sequences of decisions $(u_1, u_2, \dots)$ such that $t_0 = T$.

Since $\limsup p_t > 0$, there is a subsequence k_t such that $p_{k_t} \geq \pi > 0$. For any $k_t > T$, we know that $\mathcal{U}$ contains $\kappa\tau$-violating directions in $x_{k_t-1} = x^*$. Direction u_{k_t} is not one of them because this

[4] Strictly speaking we should introduce the tolerance $\kappa > 0$ into the SMO algorithm. We can also claim that (Keerthi and Gilbert, 2002; Takahashi and Nishi, 2003) have established proposition 16 with $\kappa = 0$ and $\tau > 0$ for the specific case of SVMs. Therefore Theorems 17 and 18 remain valid.

would make x_{k_t} different from $x_{k_t-1} = x^*$. This occurs with probability $1 - p_{k_t} \leq 1 - \pi < 1$. The probability that this happens simultaneously for N distinct $k_t > T$ is smaller than $(1-\pi)^N$ for any N. We get $P(Q_T) \leq \varepsilon/T^2$ by choosing N large enough.

Then we have $P(Q) = \sum_T P(Q_T) \leq \varepsilon\left(\sum_T 1/T^2\right) = K\varepsilon$. Hence $P(Q) = 0$ because we can choose ε as small as we want. We can therefore assert with probability 1 that $\mathcal{U}$ contains no $\kappa\tau$-violating directions in point x^*. ■

Example (LASVM) The LASVM algorithm (Section 3.2) is[5] an Approximate Stochastic WDS that alternates two strategies for selecting search directions: PROCESS and REPROCESS. Theorem 18 applies because $\limsup p_t > 0$.

Proof Consider a arbitrary iteration T corresponding to a REPROCESS.

Let us define the following assertions:

A – There are τ-violating pairs (i, j) with both $i \in \mathcal{S}$ and $j \in \mathcal{S}$.

B – A is false, but there are τ-violating pairs (i, j) with either $i \in \mathcal{S}$ or $j \in \mathcal{S}$.

C – A and B are false, but there are τ-violating pairs (i, j).

Q_t – Direction u_t is τ-violating in x_{t-1}.

A reasoning similar to the convergence discussion in Section 3.2 gives the following lower bounds (where n is the total number of examples).

$$\begin{array}{llll}
P(Q_T|A) = 1 & & & \\
P(Q_T|B) = 0 & P(Q_{T+1}|B) \geq n^{-1} & & \\
P(Q_T|C) = 0 & P(Q_{T+1}|C) = 0 & P(Q_{T+2}|C) = 0 & P(Q_{T+3}|C) \geq n^{-2}.
\end{array}$$

Therefore

$$\begin{array}{l}
P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid A\ ) \geq n^{-2} \\
P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid B\ ) \geq n^{-2} \\
P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid C\ ) \geq n^{-2}.
\end{array}$$

Since $p_t = P(Q_t \mid A \cup B \cup C)$ and since the events A, B, and C are disjoint, we have

$$p_T + p_{T+1} + p_{T+2} + p_{T+4} \geq P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid A \cup B \cup C\ ) \geq n^{-2}.$$

Therefore $\limsup p_t \geq \frac{1}{4}\, n^{-2}$. ■

Example (LASVM + Gradient Selection) The LASVM algorithm with Gradient Example Selection remains an Approximate WDS algorithm. Whenever Random Example Selection has a non zero probability to pick a τ-violating pair, Gradient Example Selection picks the a τ-violating pair with maximal gradient with probability one. Reasoning as above yields $\limsup p_t \geq 1$. Therefore Theorem 18 applies and the algorithm converges to a solution of the SVM QP problem.

Example (LASVM + Active Selection + Randomized Search) The LASVM algorithm with Active Example Selection remains an Approximate WDS algorithm. However it does not necessarily verify the conditions of Theorem 18. There might indeed be τ-violating pairs that do not involve the example closest to the decision boundary.

However, convergence occurs when one uses the Randomized Search method to select an example near the decision boundary. There is indeed a probability greater than $1/n^M$ to draw a sample

[5] See footnote 4 discussing the tolerance κ in the case of SVMs.

containing M copies of the same example. Reasonning as above yields $\limsup p_t \geq \frac{1}{4} n^{-2M}$. Therefore, Theorem 18 applies and the algorithm eventually converges to a solution of the SVM QP problem.

In practice this convergence occurs very slowly because it involves very rare events. On the other hand, there are good reasons to prefer the intermediate kernel classifiers visited by this algorithm (see Section 4).

References

M. A. Aizerman, É. M. Braverman, and L. I. Rozonoér. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

G. Bakır, L. Bottou, and J. Weston. Breaking SVM complexity with cross-training. In Lawrence Saul, Bernhard Schölkopf, and Léon Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17, pages 81–88. MIT Press, 2005.

A. Bordes and L. Bottou. The Huller: a simple and efficient online SVM. In *Proceedings of the 16th European Conference on Machine Learning (ECML2005)*, Lecture Notes in Artificial Intelligence, to appear. Springer, 2005.

L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Conference B: Computer Vision & Image Processing.*, volume 2, pages 77–82, Jerusalem, October 1994. IEEE.

L. Bottou and Y. LeCun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 21(2):137–151, 2005.

C. Campbell, N. Cristianini, and A. J. Smola. Query learning with large margin classifiers. In *Proceedings of ICML'2000*, 2000.

G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Processing Systems*, 2001.

N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Worst-case analysis of selective sampling for linear-threshold algorithms. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 241–248. MIT Press, Cambridge, MA, 2005.

C.-C. Chang and C.-J. Lin. LIBSVM : a library for support vector machines. Technical report, Computer Science and Information Engineering, National Taiwan University, 2001-2004. http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

D. Cohn, L. Atlas, and R. Ladner. Training connectionist networks with queries and selective sampling. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufmann.

R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14,* Cambridge, MA, 2002. MIT Press.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, UK, 2000.

C. Domingo and O. Watanabe. MadaBoost: a modification of AdaBoost. In *Proceedings of the 13th Annual Conference on Computational Learning Theory, COLT'00,* pages 180–189, 2000.

B. Eisenberg and R. Rivest. On the sample complexity of PAC learning using random and chosen examples. In M. Fulk and J. Case, editors, *Proceedings of the Third Annual ACM Workshop on Computational Learning Theory*, pages 154–162, San Mateo, CA, 1990. Kaufmann.

V. V. Fedorov. *Theory of Optimal Experiments.* Academic Press, New York, 1972.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel Adatron algorithm: a fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *15th International Conf. Machine Learning*, pages 188–196. Morgan Kaufmann Publishers, 1998. See (Cristianini and Shawe-Taylor, 2000, section 7.2) for an updated presentation.

C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

H.-P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The Cascade SVM. In Lawrence Saul, Bernhard Schölkopf, and Léon Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005.

I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In S. J. Hanson, J. D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46:351–360, 2002.

Y. Li and P. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46:361–387, 2002.

C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

N. Littlestone and M. Warmuth. Relating data compression and learnability. Technical report, University of California Santa Cruz, 1986.

G. Loosli, S. Canu, S.V.N. Vishwanathan, A. J. Smola, and M. Chattopadhyay. Une boîte à outils rapide et simple pour les SVM. In Michel Liquière and Marc Sebban, editors, *CAp 2004 - Confrence d'Apprentissage*, pages 113–128. Presses Universitaires de Grenoble, 2004. ISBN 9-782706-112249.

D. J. C. MacKay. Information based objective functions for active data selection. *Neural Computation*, 4(4):589–603, 1992.

N. Murata and S.-I. Amari. Statistical analysis of learning dynamics. *Signal Processing*, 74(1): 3–28, 1999.

N. J. Nilsson. *Learning machines: Foundations of Trainable Pattern Classifying Systems*. McGraw–Hill, 1965.

A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, 1962.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 839–846. Morgan Kaufmann, June 2000.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1986.

I. Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

N. Takahashi and T. Nishi. On termination of the SMO algorithm for support vector machines. In *Proceedings of International Symposium on Information Science and Electrical Engineering 2003 (ISEE 2003)*, pages 187–190, November 2003.

S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, San Francisco, California, 2000. Morgan Kaufmann.

I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Very large SVM training using core vector machines. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTAT'05)*. Society for Artificial Intelligence and Statistics, 2005.

V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, Berlin, 1982.

V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

V. N. Vapnik, T. G. Glaskova, V. A. Koscheev, A. I. Mikhailski, and A. Y. Chervonenkis. *Algorihms and Programs for Dependency Estimation*. Nauka, 1984. In Russian.

S. V. N. Vishwanathan, A. J. Smola, and M. Narasimha Murty. SimpleSVM. In *Proceedings of ICML 2003*, pages 760–767, 2003.

J. Weston, A. Bordes, and L. Bottou. Online (and offline) on an even tighter budget. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pages 413–420. Society for Artificial Intelligence and Statistics, 2005.

G. Zoutendijk. *Methods of Feasible Directions*. Elsevier, 1960.

Ignorance is Bliss: Non-Convex Online Support Vector Machines

Şeyda Ertekin SERTEKIN@CSE.PSU.EDU
*Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802 USA
NEC Laboratories America
4 Independence Way, Suite 200
Princeton, NJ 08540*

Léon Bottou LEON@BOTTOU.ORG
*NEC Laboratories America
4 Independence Way, Suite 200
Princeton, NJ 08540*

C. Lee Giles GILES@IST.PSU.EDU
*College of Information Science and Technology
Pennsylvania State University
University Park, PA 16802 USA*

Editor:

Abstract

In this paper, we propose a non-convex online Support Vector Machine (SVM) algorithm (LASVM-NC) based on the *Ramp Loss*, which has strong ability of suppressing the influence of outliers. Then, again in the online learning setting, we propose an outlier filtering mechanism (LASVM-I) based on approximating non-convex behavior in convex optimization. These two algorithms are built upon another novel SVM algorithm (LASVM-G) that is capable of generating accurate intermediate models in its iterative steps by leveraging the primal/dual gap. We present experimental results that demonstrate the merit of our frameworks in achieving significant robustness to outliers in noisy data classification where mislabeled training instances are in abundance. Experimental results show that the proposed approaches yield more scalable online SVM algorithm with sparser models and less computational running time both in the training and recognition phases without sacrificing generalization performance. We also point out the relation between the non-convex behavior in SVMs and active learning.

Keywords: Online Learning, Non-Convex Optimization, Support Vector Machines, Active Learning

1. Introduction

In supervised learning systems, the generalization performance of classification algorithms are shown to be greatly improved with large margin training. Large margin classifiers find the maximal margin hyperplane that separates the training data in the appropriately chosen kernel induced feature space. It has been shown numerous times that if a large margin is obtained, the separating hyperplane is likely to have a small misclassification rate during recognition (or prediction) (Bousquet and Elisseeff, 2002; Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004). The maximal margin methodology forms the fundamental principles of Support Vector Machines (SVMs). In the presence of noise, however, the standard maximum margin algorithm can be subject to overfitting. Cortes and Vapnik (1995) address this problem by proposing the soft margin criterion, which allows some misclassified examples in the training phase for better predictive power. However, the soft margin approach in SVMs has brought a serious shortcoming along with its advantages. With the soft margin criterion, patterns are allowed to be misclassified for a certain cost and outlier (misclassified) examples start to play a dominant role in determining the decision hyperplane, since they tend to have the largest margin loss according to the Hinge Loss. Nonetheless, due to its convex property and practicality, Hinge Loss became a commonly used loss function in SVMs.

Convexity is viewed as a virtue by most of the machine learning researchers both from a theoretical and experimental point of view. Convex methods can easily be mathematically analyzed and bounds can be produced. In addition, convex solutions are guaranteed to reach to the global optimum, avoiding the fear of ending up in the local optimum. The popularity of convexity further increased after the success of convex algorithms, particularly with SVMs, which yield good generalization performance and strong theoretical foundations. However, despite many advantages of convex modeling, the price we pay for insisting on convexity is an increase in the size of the model and the scaling properties of the algorithm. In this paper, we show that shifting gears from convexity to non-convexity can be very effective for achieving sparse and scalable solutions, particularly when the data consists of abundant label noise. We present herein experimental results showing how a non-convex loss function, *Ramp Loss*, can be efficiently integrated to an online SVM algorithm in order to suppress the influence of misclassified examples.

Various works in the history of machine learning research focused on using non-convex loss functions as an alternate to convex Hinge Loss, in large margin classifiers. While Mason et al. (2000) and Krause and Singer (2004) applied it to Boosting, Perez-Cruz et al. (2002) and Linli Xu (2006) proposed training algorithms for SVMs with the Ramp Loss and solved the non-convex optimization by utilizing semi-definite programming and convex relaxation techniques. On the other hand, some previous work of Liu et al. (2005) and Wang et al. (2008) used the concave-convex programming (CCCP) for non-convex optimization as the work presented here. Those studies are worthwhile in the endeavor of achieving sparse models or competitive generalization performance, nevertheless none of them are efficient in terms of computational running time and scalability for real-world data mining applications and yet the improvement in classification accuracy is only marginal. Collobert et al. (2006) pointed out the scalability advantages of non-convex approaches and used CCCP for non-convex optimization in order to achieve faster batch SVMs and Transductive SVMs. In this paper, we focus on bringing the scalability advantages of non-convexity to the online learning setting by using an online SVM algorithm, LASVM (Bordes et al., 2005).

Online learning is advantageous when dealing with streaming, or very large scale data. Online learners incorporate the information of new seen training data into the model without retraining it with the previously seen entire training data. Since they process the data one at a time in the training phase, selective sampling can be applied and evaluation of the informativeness of the data prior to the processing by the learner becomes possible. We implement an online SVM training with non-convex loss function (LASVM-NC), which yields a significant speed improvement in training and builds a sparser model, hence resulting in faster recognition than its convex version as well. Based on selective sampling, we further propose an SVM algorithm (LASVM-I) that ignores the instances that lie in the flat region of the Ramp Loss in advance, before they are processed by the learner. Although this may sound like an over-aggressive training sample elimination process, we point out that those instances do not play role in determining the decision hyperplane according to the Ramp Loss anyway. Making a right decision about whether to eliminate or process a training data highly depends on the trustworthiness of the current model. The intermediate models should be well enough trained in order to capture the characteristics of the training data, but on the other hand, should not be over-optimized since only part of the entire training data is seen at that point in time. We build a balance within those two situations by leveraging the gap between primal and dual functions during the optimization steps of online SVM (LASVM-G). We then build a non-convex optimization scheme and a training sample ignoring mechanism on top of LASVM-G. We show that for a particular case of sample elimination scenario, misclassified instances according to the current learned model are not taken into account at the training process ($s = 0$). For another case, only the instances in the margin pass the barrier of elimination and are processed in the training, hence leading to an extreme case of the well-known *small pool active learning* framework (Ertekin et al., 2007) in online SVMs (when $s = -1$).

The proposed non-convex implementation and selective sample ignoring policy yields sparser models with fewer support vectors (SVs) and faster training with less computational time and kernel computations which overall leads to a more scalable online SVM algorithm. The advantages of the proposed methods become more pronounced in noisy data classification where mislabeled samples are in abundance.

2. Support Vector Machines

Support Vector Machines (Cortes and Vapnik, 1995) are well known for their strong theoretical foundations, generalization performance and ability to handle high dimensional data. In the binary classification setting, let $((x_1, y_1) \cdots (x_n, y_n))$ be the training dataset where x_i are the feature vectors representing the instances and $y_i \in (-1, +1)$ are the labels of those instances. Using the training set, SVM builds an optimum hyperplane – a linear discriminant in a higher dimensional feature space – that separates the two classes by the largest margin. The SVM solution is obtained by minimizing the following objective function:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \min_{\boldsymbol{w},b,\xi_i} \; \|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{with} \quad \left\{ \begin{array}{ll} \forall\, i & y_i(\boldsymbol{w}^T\Phi(x_i) - b) \geq 1 - \xi_i \\ \forall\, i & \xi_i \geq 0 \end{array} \right. \tag{1}$$

where $\boldsymbol{w}$ is the norm of the hyperplane, b is the offset, y_i are the labels, $\Phi(\cdot)$ is the mapping from input space to feature space, and ξ_i are the slack variables that permit the non-separable case by allowing misclassification of training instances. In practice, the convex quadratic programming (QP) problem (1) is solved through its dual formulation:

$$\max_{\boldsymbol{\alpha}} G(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j}) \tag{2}$$

$$\text{subject to} \quad \begin{cases} \sum_i \alpha_i = 0 \\ A_i \leq \alpha_i \leq B_i \\ \quad A_i = \min(0, Cy_i) \\ \quad B_i = \max(0, Cy_i) \end{cases} \tag{3}$$

where $K(\mathbf{x_i}, \mathbf{x_j}) = \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$ is the kernel matrix representing the dot products $\Phi(\mathbf{x_i})\Phi(\mathbf{x_j})$ in feature space and the α_i are the *Lagrange multipliers*. This formulation slightly deviates from the original formulation; the coefficients α_i in Eq. 3 inherit the signs of the labels y_i, permitting α's to take on negative values. The training instances with $\alpha_i \neq 0$ are called *support vectors* and they define the position of the hyperplane, with its norm $\boldsymbol{w}$ represented as

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \Phi(x_i) \tag{4}$$

Once a model is trained, a soft margin SVM classifies a pattern x according to the sign of a decision function, which can be represented as a *kernel expansion*

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b \tag{5}$$

where the sign of $\hat{y}(\mathbf{x})$ represents the predicted classification of $\mathbf{x}$. A widely popular methodology for solving the SVM QP problem is Sequential Minimal Optimization (SMO) (Platt, 1999). SMO works by making successive direction searches and taking steps along directions based on "τ-violating pairs"

$$(i, j) \text{ is a } \tau\textit{-violating pair} \iff \begin{cases} \alpha_i < B_i \\ \alpha_j > A_j \\ g_i - g_j > \tau \end{cases}$$

where A and B follow the same definition as the constraint in Eq. 3 and τ is a small positive tolerance. Each SMO step, then, involves finding such a pair and taking an optimization step along that feasible direction. The coefficients α_i and α_j are modified by opposite amounts, satisfying the first constraint in (3). This ability to break down large optimization problems into small bits of pairwise optimizations that can be solved analytically is the underlying reason for SMO's efficiency and wide adoption in optimization of SVM solvers.

```
SMO Algorithm
    Denote $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
1)  Set $\boldsymbol{\alpha} \leftarrow \mathbf{0}$ and compute the initial gradient $g_i = y_i - \sum_k \alpha_k K_{ik}$
2)  Choose a $\tau$-violating pair $(i, j)$. Stop if no such pair exists.
3)  $\lambda \leftarrow \min\left\{ \dfrac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}},\ B_i - \alpha_i,\ \alpha_j - A_j \right\}$
    $\alpha_i \leftarrow \alpha_i + \lambda\ ,\quad \alpha_j \leftarrow \alpha_j - \lambda$
    $g_s \leftarrow g_s - \lambda(K_{is} - K_{js}) \quad \forall\, s \in \{1 \dots n\}$
4)  Return to step (2)
```

Although SMO was originally developed for batch (offline) SVMs, SMO-like optimization schemes have enabled the development of online SVMs that can achieve the generalization performance of batch learners at a faster rate. The next section presents LASVM, an efficient online SVM algorithm (LASVM)that provides the basis for the online non-convex SVM solver presented in this paper.

3. LASVM

LASVM is an efficient online SVM solver that uses less memory and trains significantly faster than other state-of-the-art SVM solvers while yielding competitive misclassification rates after a single pass over the training examples. LASVM realizes these benefits due its novel optimization steps that has been inspired by SMO. LASVM applies the same pairwise optimization principle to online learning by defining two direction search operations. The first operation, PROCESS, attempts to insert an example $k \notin \mathcal{S}$ into $\mathcal{S}$, the set of current support vector indices (steps 1-2 of the algorithm). In the online setting, this relates to processing a new example at time t. Then it searches a second example in $\mathcal{S}$ to find the τ-violating pair with maximal gradient (steps 3-4) and performs a direction search (step 5). Direction searches of the PROCESS kind involve at least one example that is not a support vector of the current kernel expansion. They potentially can change the coefficient of this example and make it a support vector.

```
LASVM PROCESS($k$)
1)  Bail out if $k \in \mathcal{S}$.
2)  $\alpha_k \leftarrow 0\ ,\quad g_k \leftarrow y_k - \sum_{s \in \mathcal{S}} \alpha_s K_{ks}\ ,\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{k\}$
3)  If $y_k = +1$ then
        $i \leftarrow k\ ,\quad j \leftarrow \arg\min_{s \in S} g_s$ with $\alpha_s > A_s$
    else
        $j \leftarrow k\ ,\quad i \leftarrow \arg\max_{s \in S} g_s$ with $\alpha_s < B_s$
4)  Bail out if $(i, j)$ is not a $\tau$-violating pair.
5)  $\lambda \leftarrow \min\left\{ \dfrac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}},\ B_i - \alpha_i,\ \alpha_j - A_j \right\}$
    $\alpha_i \leftarrow \alpha_i + \lambda\ ,\quad \alpha_j \leftarrow \alpha_j - \lambda$
    $g_s \leftarrow g_s - \lambda(K_{is} - K_{js}) \quad \forall\, s \in \mathcal{S}$
```

The second operation, REPROCESS, removes some elements from $\mathcal{S}$. It first searches the τ-violating pairs of instances from $\mathcal{S}$ with maximal gradient (steps 1-2), and performs a direction search (step 3). Then it removes blatant non support vectors (step 4). Direction searches of the REPROCESS kind involve two examples that are support vectors in the current kernel expansion. They potentially can zero the coefficient of one or both support vectors and thus remove them from the kernel expansion. In short, PROCESS adds new instances to $\mathcal{S}$ and REPROCESS removes the ones that the learner does not benefit from anymore. Repeating LASVM iterations on randomly chosen training set examples provably converges to the SVM solution.

```
LASVM REPROCESS
  1) $i \leftarrow \arg\max_{s\in\mathcal{S}} g_s$  with  $\alpha_s < B_s$
     $j \leftarrow \arg\min_{s\in\mathcal{S}} g_s$  with  $\alpha_s > A_s$
  2) Bail out if $(i,j)$ is not a $\tau$-violating pair.
  3) $\lambda \leftarrow \min\left\{ \dfrac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}},\ B_i - \alpha_i,\ \alpha_j - A_j \right\}$
     $\alpha_i \leftarrow \alpha_i + \lambda$ ,   $\alpha_j \leftarrow \alpha_j - \lambda$
     $g_s \leftarrow g_s - \lambda(K_{is} - K_{js}) \quad \forall\, s \in \mathcal{S}$
  4) $i \leftarrow \arg\max_{s\in\mathcal{S}} g_s$  with  $\alpha_s < B_s$
     $j \leftarrow \arg\min_{s\in\mathcal{S}} g_s$  with  $\alpha_s > A_s$
     For all $s \in \mathcal{S}$ such that $\alpha_s = 0$
         If $y_s = -1$ and $g_s \geq g_i$  then  $\mathcal{S} = \mathcal{S} - \{s\}$
         If $y_s = +1$ and $g_s \leq g_j$  then  $\mathcal{S} = \mathcal{S} - \{s\}$
  5) $b \leftarrow (g_i + g_j)/2$ ,   $\delta \leftarrow g_i - g_j$
```

Online Iterations In its original formulation, after initializing the state variables (step 1), the Online LASVM algorithm alternates between single PROCESS and REPROCESS operations (step 2). Then it simplifies the kernel expansion by running REPROCESS to remove all τ-violating pairs from the kernel expansion, a step known as FINISHING (step 3). The optimizations performed in the FINISHING step reduce the number of support vectors in the SVM model.

```
LASVM
  1) Initialization:
     Seed $\mathcal{S}$ with a few examples of each class.
     Set $\boldsymbol{\alpha} \leftarrow \mathbf{0}$ and compute the initial gradient $\boldsymbol{g} = y - \hat{y}(\boldsymbol{x}) + b$
  2) Online Iterations:
     Repeat a predefined number of times:
       - Pick an example $k_t$
       - Run PROCESS($k_t$).
       - Run REPROCESS once.
  3) Finishing:
     Repeat REPROCESS until $\delta \leq \tau$.
```

4. LASVM with Gap-based Optimization – LASVM-G

In this section, we present LASVM-G – an efficient online SVM algorithm that brings performance enhancements to LASVM. Instead of running a single REPROCESS operation after each PROCESS step, LASVM-G adjusts the number of REPROCESS operations at each online iteration by leveraging the gap between the primal and the dual functions. Further, LASVM-G replaces LASVM's one time FINISHING optimization and cleaning stage with the optimizations performed in each REPROCESS cycle at each iteration and the periodic non-SV removal steps. These improvements enable LASVM-G to generate more reliable intermediate models than LASVM, which lead to sparser SVM solutions that have better generalization performance.

4.1 Leveraging the Gap Between the Primal and the Dual Functions

One question regarding the optimization scheme in LASVM is the rate at which to perform REPROCESS operations. A simple approach would be to perform one REPROCESS operation after each PROCESS step. However, this heuristic approach may result in under optimization of the objective function in the intermediate steps if this rate is smaller than the optimal proportion. Another option would be to run REPROCESS until a small predefined threshold ε exceeds the L_∞ norm of the projection of the gradient $(\partial G(\alpha)/\partial \alpha_i)$. Little work has been done to determine the correct value of the threshold ε. A geometrical argument relates this norm to the position of the support vectors relative to the margins (Keerthi et al., 2001). As a consequence, one usually chooses a relatively small threshold, typically in the range 10^{-4} to 10^{-2}. Using such a small threshold to determine the rate of REPROCESS operations results in many REPROCESS steps after each PROCESS operation. This will not only increase the training time and computational complexity, but can potentially over optimize the objective function at each iteration. Since non-convex iterations work towards suppressing some training instances (outliers), the intermediate learned models should be well enough trained in order to capture the characteristics of the training data but on the other hand, should not be over-optimized since only part of the entire training data is seen at that point in time. Therefore, it is necessary to employ a criteria to determine an accurate rate of REPROCESS operations after each PROCESS. We define this policy as *the minimization of the gap between the primal and the dual* (Schölkopf and Smola, 2002).

Optimization of Primal/Dual Gap From the formulations of the primal and dual functions in (1) and (2) respectively, it can be shown that the optimal values of the primal and dual are same (Chapelle, 2007). At any non-optimal point, the primal function is *guaranteed* to lie above the dual curve. In formal terms, let $\hat{\theta}$ and $\hat{\alpha}$ be solutions of problems (1) and (2), respectively. The strong duality asserts that for any feasible θ and α,

$$G(\alpha) \leq G(\hat{\alpha}) = J(\hat{\theta}) \leq J(\theta) \qquad with \qquad \hat{\theta} = \sum_i \hat{\alpha}_i y_i \Phi(x_i) \tag{6}$$

That is, at any time during the optimization, the value of the primal $J(\cdot)$ is higher than the dual $G(\cdot)$. Using the equality $w = \sum_l \alpha_l x_l$, we show that this holds as follows:

$$\begin{aligned}
J(\theta) - G(\alpha) &= \frac{1}{2}\|w\|^2 + C\sum_l |1 - y_l(w\cdot x_l + b)|_+ - \sum_l \alpha_l y_l + \frac{1}{2}\|w\|^2 \\
&= \|w\|^2 - \sum_l \alpha_l y_l + C\sum_l |1 - y_l(w\cdot x_l + b)|_+ \\
&= w(\sum_l \alpha_l y_l) - \sum_l \alpha_l y_l + C\sum_l |1 - y_l(w\cdot x_l + b)|_+ \\
&= -\sum_l \alpha_l y_l |1 - y_l(w\cdot x_l + b)|_+ + C\sum_l |1 - y_l(w\cdot x_l + b)|_+ \\
&= \sum_l \underbrace{(C - \alpha_l y_l)}_{\geq 0} \underbrace{|1 - y_l(w\cdot x_l + b)|_+}_{\geq 0} \\
&\geq 0
\end{aligned}$$

where $C - \alpha_l y_l \geq 0$ is satisfied by the constraint of the dual function (3). Then, the SVM solution is obtained when one reaches $\bar{\theta}, \bar{\alpha}$ such that

$$\varepsilon > J(\bar{\theta}) - G(\bar{\alpha}) \qquad \textit{where} \qquad \bar{\theta} = \sum_i \bar{\alpha}_i y_i \Phi(x_i) \tag{7}$$

The strong duality in Equation 6 then guarantees that $J(\bar{\theta}) < J(\hat{\theta}) + \varepsilon$. Few solvers implement this criterion since it requires the additional calculation of the gap $J(\theta) - G(\alpha)$. In this paper, we advocate using criterion (7) using a threshold value ε that grows sublinearly with the number of examples. Letting ε grow makes the optimization coarser when the number of examples increases. As a consequence, the asymptotic complexity of optimizations in online setting can be smaller than that of the exact optimization.

Most SVM solvers use the dual formulation of the QP problem. However, increasing the dual does not necessarily reduce the primal/dual gap. The dual function follows a nice monotonically increasing pattern at each optimization step, whereas the primal shows significant up and down fluctuations. In order to keep the size of the primal/dual gap in check, before each PROCESS operation we compute the standard deviation of the primal, which we call the *Gap Target* $\hat{\mathbb{G}}$

$$\hat{\mathbb{G}} = \max(0, \sqrt{\sum_{i=1}^{n} h_i^2 - \frac{(\sum_{i=1}^{n} h_i)^2}{l}}) \tag{8}$$

where l is the number of support vectors and $h_i = C_i y_i g_i$. After computing the gap target, we run a PROCESS step and check the new Gap $\mathcal{G}$ between the primal and the dual. After an easy derivation, the gap is computed as

$$\mathcal{G} = -\sum_{i=1}^{n} (\alpha_i g_i + \max(0, C \cdot g_i)) \tag{9}$$

We cycle between running REPROCESS and computing the gap $\mathcal{G}$ until the termination criteria $\mathcal{G} \leq \max(C, \hat{\mathbb{G}})$ is reached. That is, we require the primal/dual gap after the

REPROCESS operations to be smaller than or equal to initial gap target $\hat{\mathbb{G}}$. After this point, the learner continues with computing the new Gap Target and running PROCESS and REPROCESS operation on the next fresh instance from the unseen example pool.

4.2 Building Blocks

The implementation of LASVM-G maintains the following pieces of information as its key building blocks: the coefficients α_i of the current kernel expansion $\mathcal{S}$, the bounds for each α, and the partial derivatives of the instances in the expansion, given as

$$g_k \;=\; \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_k} \;=\; y_k - \sum_i \alpha_i K(x_i, x_k) \;=\; y_k - \hat{y}(x_k) \tag{10}$$

The kernel expansion here maintains all the training instances in the learner's active set, both the support vectors and the instances with $\alpha = 0$.

Optimization is driven by two kinds of direction searches. The first operation, PROCESS, inserts an instance into the kernel expansion and initializes its α_i and gradient g_i (Step 1). After computing the step size (Step 2), it performs a direction search (Step 3). We set the offset term for kernel expansion b to zero for computational simplicity. This removes the necessity of satisfying the constraint $\sum_{i\in\mathcal{S}} \alpha_i = 0$, enabling the algorithm to update a single α at a time, both in PROCESS and REPROCESS operations.

```
LASVM-G PROCESS(i)
  1) $\alpha_i \leftarrow 0, \quad g_i \leftarrow y_k - \sum_{s\in S} \alpha_s K_{is}$
  2) If $g_i < 0$ then
         $\lambda = \max\left\{A_i - \alpha_i, \frac{g_i}{K_{ii}}\right\}$
     Else
         $\lambda = \max\left\{B_i - \alpha_i, \frac{g_i}{K_{ii}}\right\}$
  3) $\alpha_i \leftarrow \alpha_i + \lambda$
     $g_s \leftarrow g_s - \lambda K_{is} \quad \forall s$ in kernel expansion
```

The second operation, REPROCESS, searches all of the instances in the kernel expansion and selects the instance with the maximal gradient (Steps 1-3). Once an instance is selected, LASVM-G computes a step size (Step 4) and performs a direction search (Step 5).

```
LASVM-G REPROCESS()
  1) $i \leftarrow \arg\min_{s\in S} g_s$ with $\alpha_s > A_s$
     $j \leftarrow \arg\max_{s\in S} g_s$ with $\alpha_s < B_s$
  2) Bail out if $(i, j)$ is not a $\tau$-violating pair.
  3) If $g_i + g_j < 0$ then $g \leftarrow g_i$
     Else $g \leftarrow g_j$
  4) If $g < 0$ then
         $\lambda = \max\left\{A_i - \alpha_i, \frac{g}{K_{ii}}\right\}$
     Else
         $\lambda = \min\left\{B_i - \alpha_i, \frac{g}{K_{ii}}\right\}$
  5) $\alpha_i \leftarrow \alpha_i + \lambda$
     $g_s \leftarrow g_s - \lambda K_{is} \quad \forall s$ in kernel expansion
```

Both PROCESS and REPROCESS operate on the instances in the kernel expansion, but neither of them remove any instances from it. A removal step is necessary for improved efficiency because as the learner evolves, the instances that were admitted to the kernel expansion in earlier iterations as support vectors may not serve as support vectors anymore. Keeping such instances in the kernel expansion slows down the optimization steps without serving much benefit to the learner and increases the application's requirement for computational resources. A straightforward approach to address this inefficiency would be to remove all of the instances with $\alpha_i = 0$, namely all non-support vectors. One concern with this approach is that once an instance is removed, it will not be seen by the learner again, and thus, it will no longer be eligible to become a support vector in the later stages of training. It is important to find a balance between maintaining the efficiency of a small sized kernel expansion and not aggressively removing instances from the kernel expansion. Therefore, the cleaning policy needs to preserve the instances that can potentially become SVs at a later stage of training while removing instances that have the lowest possibility of becoming SV's in the future.

```
CLEAN
    $n$ : number of non-SVs in the kernel expansion.
    $m$ : maximum number of allowed non-SVs.
    $\vec{v}$ : Array of partial derivatives.

  1) If $n < m$ return
  2) $\vec{v} \leftarrow \vec{v} \cup |g_i|_+, \ \forall i$ with $\alpha_i = 0$
  3) Sort the gradients in $\vec{v}$ in ascending order.
     $g_{threshold} \leftarrow v[m]$
  4) If $|g_i|_+ \geq g_{threshold}$ then remove $x_i, \ \forall i$ with $\alpha_i = 0$
```

Our cleaning procedure periodically checks the number of non-SVs in the kernel expansion If the number of non-SVs n is more than the number of instances that is permitted in the expansion m by the algorithm, CLEAN selects the extra non-SV instances with highest gradients for removal. Note that, it is immaterial to distinguish whether an instance has not been an SV for many iterations or it has just become a non-SV. In either case, those examples do not currently contribute to the classifier and are treated equally from a cleaning point of view.

4.3 Online Iterations in LASVM-G

LASVM-G exhibits the same learning principle as LASVM, but in a more systematic way. Both algorithms make one pass (one epoch) over the training set. Empirical evidence suggests that a single epoch over the entire training set yields a classifier as good as the SVM solution. The β parameter shown in the LASVM-G algorithm block controls the optimization behavior and will be described in detail when LASVM-NC is introduced in the next section. LASVM-G initially sets $\boldsymbol{\beta} \leftarrow \mathbf{0}$ and never updates the β's and thus maintains convex optimization behavior based on Hinge Loss.

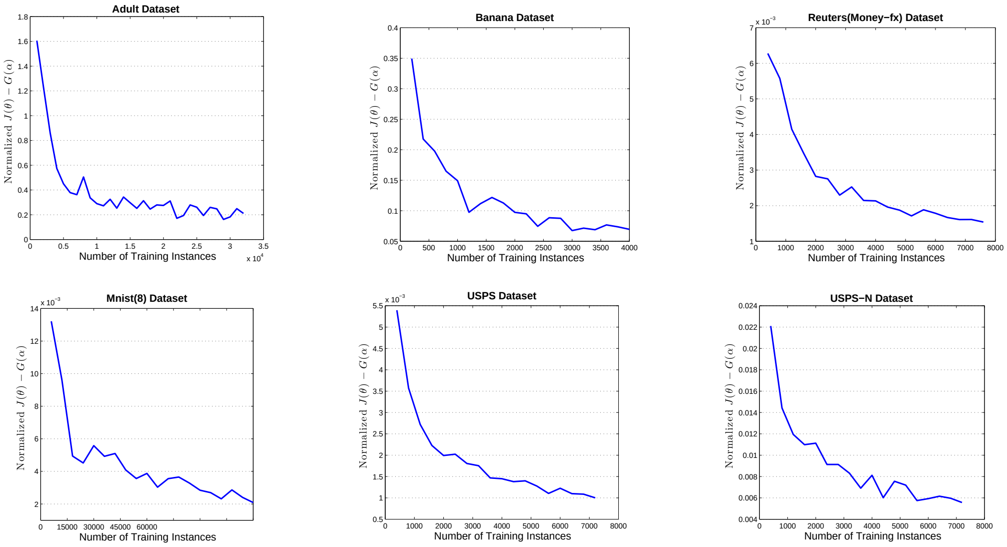
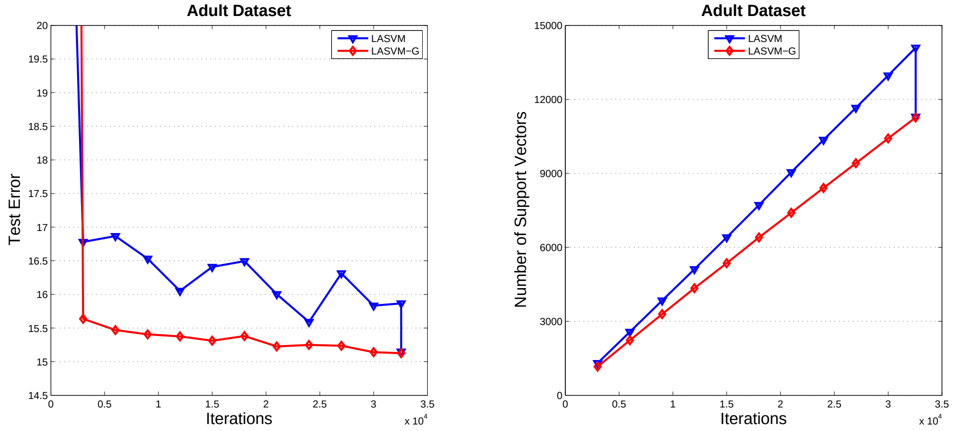


Figure 1: The primal/dual gap $(J(\theta) - G(\alpha))$, normalized by the number of training instances. The normalization eliminates the bias on the primal and dual values caused by different number of support vectors at various snapshots of training LASVM-G

Upon initialization, LASVM-G alternates between its PROCESS and REPROCESS steps during the epoch like LASVM, but distributes LASVM's one time FINISHING step to the optimizations performed in each REPROCESS cycle at each iteration and the periodic CLEAN operations. Another important property of LASVM-G is that it leverages the gap between the primal and the dual functions to determine the number of REPROCESS steps after each PROCESS (the -G suffix emphasizes this distinction). Reducing the primal/dual gap too fast can cause over optimization in early stages without yet observing sufficient training data. Conversely, reducing the gap too slow can result in under optimization in the intermediate iterations. Figure 1 shows that as the learner sees more training examples, the primal/dual gap gets smaller.

```
LASVM-G
  1) Initialization:
     Set $\boldsymbol{\beta} \leftarrow \mathbf{0}$, $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
  2) Online Iterations:
     Pick an example $x_i$
     Compute Gap Target $\hat{\mathbb{G}}$
     $Threshold \leftarrow \max(C, \hat{\mathbb{G}})$
     Run PROCESS($x_i$)
     while Gap $\mathcal{G} > Threshold$
        Run REPROCESS
     end
     Periodically run CLEAN
```



(a) Test Error Convergence

(b) Growth of the Number of Support Vectors

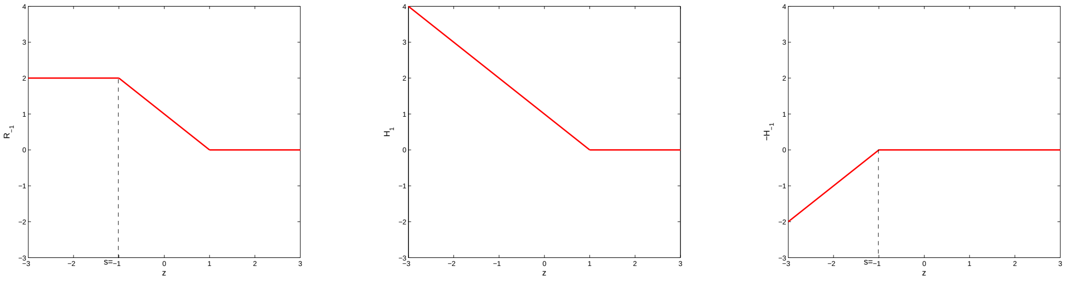
Figure 2: Comparison of LASVM and LASVM-G for Adult dataset. We see that LASVM-G arrives at a more accurate SVM solution (Fig. (a)) with fewer support vectors at a faster rate (Fig. (b)). The drop of the Test Error and the number of support vectors in the end of one pass of the iterations for LASVM is the result of the optimizations done by the FINISHING step.

The major enhancements that are introduced to LASVM enable LASVM-G to achieve higher prediction accuracies than LASVM in the intermediate stages of training. Figure 2 presents a comparative analysis of LASVM-G versus LASVM for the Adult dataset. While both algorithms report the same generalization performance in the end of training, LASVM-G reaches a better classification accuracy at an earlier point in training than LASVM and is able to maintain its performance relatively stable with a more reliable model over the course of training. Furthermore, LASVM-G maintains fewer number of support vectors in the intermediate training steps, as evidenced in Figure 2(b).

In the next sections, we further introduce three SVM algorithms that are implemented based on LASVM-G, namely LASVM-NC, LASVM-I and FULL SVM. While these SVM algorithms share the main building blocks of LASVM-G, each algorithm exhibits a distinct learning principle. LASVM-NC uses the LASVM-G methodology in a non-convex learner setting. LASVM-I is a learning scheme that we propose as a convex variant of LASVM-NC. FULL SVM does not take advantage of the non-convexity or the efficiency of the CLEAN operation, and acts as a baseline case for comparisons in our experimental evaluation.

5. Non-convex Online SVM Solver – LASVM-NC

In this section, we present LASVM-NC, a non-convex online SVM solver that achieves sparser SVM solutions in less time than online convex SVMs and batch SVM solvers. We first introduce the non-convex Ramp Loss function and discuss how non-convexity can overcome the inefficiencies and scalability problems of convex SVM solvers. We then present the



(a) Ramp Loss

(b) Convex Hinge Loss

(c) Concave Loss

Figure 3: The Ramp Loss 3(a) can be decomposed into a Convex Hinge Loss 3(b) and a Concave Loss 3(c)

methodology to optimize the non-convex objective function, followed by the description of the online iterations of LASVM-NC.

5.1 Ramp Loss

Traditional convex SVM solvers rely on the Hinge Loss H_1 (shown in Figure 3(b)) to solve the QP problem, which can be represented in Primal form as

$$\min_{\boldsymbol{w},b} J(\boldsymbol{w},b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{l=1}^{n} H_1(y_i f(x_i)) \tag{11}$$

In the Hinge Loss formulation $H_s(z) = \max(0, s - z)$, s indicates the Hinge point and the elbow at $s = 1$ indicates the point at which $y_l f_\theta(x_l) = y_l(w \cdot \Phi(x_l) + b) = 1$. Assume for simplicity that the Hinge Loss is made differentiable with a smooth approximation on a small interval $z \in [1 - \epsilon, 1 + \epsilon]$ near the hinge point. Differentiating (11) shows that the minimum $\boldsymbol{w}$ must satisfy

$$\boldsymbol{w} = -C\sum_{l=1}^{L} y_l H_1^{'}(y_l) f_\theta(\boldsymbol{x}_l)\Phi(\boldsymbol{x}_i) \tag{12}$$

In this setting, correctly classified instances outside of the margin ($z \geq 1$) can not become SVs because $H_1^{'}(z) = 0$. On the other hand, for the training examples with ($z < 1$), $H_1^{'}(z)$ is 1, so they cost a penalty term at the rate of misclassification of those instances. One problem with Hinge Loss based optimization is that it imposes no limit on the influences of the outliers; that is, the misclassification penalty is unbounded. Furthermore in Hinge Loss based optimization, all misclassified training instances become support vectors. Consequently, the number of support vectors scales linearly with the number of training examples (Steinwart, 2003). Specifically,

$$\frac{\#SV}{\#Examples} \rightarrow 2\mathfrak{B}_\Phi \tag{13}$$

where $\mathfrak{B}_\Phi$ is the best possible error achievable linearly in the feature space $\Phi(\cdot)$. Such fast pace of growth of the number of support vectors becomes prohibitive for training SVMs in large scale datasets.

In practice, not all misclassified training examples are necessarily informative to the learner. For instance in noisy datasets, many instances with label noise become support vectors due to misclassification, even though they are not informative about the correct classification of new instances in recognition. Thus it is reasonable to limit the influence of the outliers and allow the real informative training instances define the model. Since Hinge Loss admits all outliers into the SVM solution, we need to select an alternative loss function that enables to selectively ignore the instances that are misclassified according to the current model. For this purpose, we propose to use the Ramp Loss (Figure 3(a))

$$R_s(z) = H_1(z) - H_s(z) \tag{14}$$

to control the score window for z at which we are willing to convert instances into support vectors. Replacing $H_1(z)$ with $R_s(z)$ in (12), we see that the Ramp Loss suppresses the influence of the instances with score $z < s$ by not converting them into support vectors. However, since Ramp Loss is non-convex, it prohibits us from using widely popular optimization schemes devised for convex functions.

While convexity has many advantages and nice mathematical properties, we point out that non-convexity has its own benefits of yielding faster and sparser solutions. In this work, our aim is to achieve the best of both worlds; generate a reliable and robust SVM solution that is faster and sparser than traditional convex optimizers. This can be achieved by reducing the complexity of non-convex loss function by transforming the problem into a difference of convex parts. We employ the Concave-Convex Procedure (CCCP) (Yuille and Rangarajan, 2002) to solve the non-convex optimization problem in this fashion. CCCP is closely related to the “Difference of Convex” methods that have been applied to many problems, including dealing with missing values in SVMs (Smola et al., 2005), improving boosting algorithms (Krause and Singer, 2004), and implementing ψ-learning (Shen et al., 2003; Liu et al., 2005). The elegance of CCCP comes from the fact that it first decomposes a non-convex cost function into a combination of convex parts (by a local approximation of the concave part) and performs optimization on the difference of these convex functions. Formally, CCCP can be described as follows.

Assume that a cost function $J(\boldsymbol{\theta})$ can be decomposed into the sum of a convex part $J_{vex}(\boldsymbol{\theta})$ and a concave part $J_{cav}(\boldsymbol{\theta})$. Each iteration of CCCP approximates the concave part by its tangent and minimizes the resulting convex function.

Algorithm 1 The Concave-Convex Procedure (CCCP)

```
Initialize $\boldsymbol{\theta}^0$ with a best guess
repeat
    $\boldsymbol{\theta}^{t+1} = \arg\min_{\boldsymbol{\theta}}(J_{vex}(\boldsymbol{\theta}) + J'_{cav}(\boldsymbol{\theta}^t) \cdot \boldsymbol{\theta})$
until convergence of $\boldsymbol{\theta}^t$
```

By summing two inequalities for $\boldsymbol{\theta}$ and from the concavity of $J_{cav}(\boldsymbol{\theta})$, it is easy to infer that the cost $J(\boldsymbol{\theta}^t)$ decreases after each iteration:

$$\begin{aligned} J_{vex}(\boldsymbol{\theta^{t+1}}) + J^{'}_{cav}(\boldsymbol{\theta^{t}}) \cdot \theta^{t+1} &\leq J_{vex}(\boldsymbol{\theta^{t}}) + J^{'}_{cav}(\boldsymbol{\theta^{t}}) \cdot \boldsymbol{\theta}^{t} \\ J_{cav}(\boldsymbol{\theta^{t+1}}) &\leq J_{cav}(\boldsymbol{\theta^{t}}) + J^{'}_{cav}(\boldsymbol{\theta^{t}}) \cdot (\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^{t}) \end{aligned} \tag{15}$$

We do not need any hyper-parameters for this optimization, and since the problem is now purely convex, we can use any efficient convex algorithm to solve this problem. Similarly, the Ramp Loss can be decomposed into a difference convex parts (as shown in Figure 3 and Equation 14), which makes it amenable to CCCP optimization. The new cost $J^s(\boldsymbol{\theta})$ after substituting the Hinge Loss with the Ramp Loss then reads:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \; J^s(\boldsymbol{\theta}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{l=1}^{n} R_s(y_i f(x_i)) \\ &= \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{l=1}^{n} H_1(y_i f(x_i))}_{J^s_{vex}(\boldsymbol{\theta})} \underbrace{-C\sum_{l=1}^{n} H_s(y_i f(x_i))}_{J^s_{cav}(\boldsymbol{\theta})} \end{aligned} \tag{16}$$

For simplification purposes, we introduce the notation

$$\beta_l = y_l \frac{\partial J^s_{cav}(\boldsymbol{\theta})}{\partial f_{\boldsymbol{\theta}}(x_l)} = \begin{cases} C & \text{if } \; y_l f_{\boldsymbol{\theta}}(x_l) < s \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where $f_{\boldsymbol{\theta}}(x_l) = \sum_{i=1}^{n} \alpha_i K(x_l, x_i) + b$ is the kernel expansion. The cost function in Equation 16, along with the notation introduced in Equation 17 is then reformulated as the following dual optimization problem:

$$\max_{\alpha} \; G(\alpha) = \sum_i y_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j K_{i,j} \quad \text{with} \begin{cases} A_i \leq \alpha_i \leq B_i \\ A_i = \min(0, Cy_i) - \beta_i y_i \\ B_i = \max(0, Cy_i) - \beta_i y_i \\ \beta_i \;\text{ from Equation 17} \end{cases} \tag{18}$$

There is a fundamental difference between non-convex optimization in batch and online SVMs. Batch non-convex SVMs alternate between solving (18) and updating the β's of *all* training instances. LASVM-NC, on the other hand, adjusts the β of only the new fresh instance based on the current model and solves (18) while the online algorithm is progressing. We also would like to point out that if the β's of all of the training instances are initialized to zero and left unchanged in the online iterations, the algorithm becomes traditional Hinge Loss SVM. From another viewpoint, if $s \ll 0$, then the β's will remain zero and the effect of Ramp Loss will not be realized. Therefore, (18) can be viewed as a generic algorithm that can act as both Hinge Loss SVM and Ramp Loss SVM with CCCP that enables non-convex optimization.

5.2 Online Iterations in LASVM-NC

The online iterations in LASVM-NC are similar to LASVM-G in the sense that they are also based on alternating PROCESS and REPROCESS steps, with the distinction of replacing the Hinge Loss with the Ramp Loss. When a new example x_i is encountered, LASVM-NC first computes the β_i for this instance as presented in the algorithm block, where y_i is the class label, $f_{\boldsymbol{\theta}}(x_i)$ is the decision score for x_i, and s is the score threshold for permitting instances to become support vectors.

```
LASVM-NC
  1) Initialization:
     Set $\boldsymbol{\beta} \leftarrow \mathbf{0}$, $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
  2) Online Iterations:
     Pick an example $x_i$
     Set $\beta_i = \begin{cases} C & \text{if } y_i f_{\boldsymbol{\theta}}(x_i) < s \\ 0 & \text{otherwise} \end{cases}$
     Set $\alpha_i$ bounds for $x_i$ to $(\min(0, Cy_i) - \beta_i y_i \leq \alpha_i \leq \max(0, Cy_i) - \beta_i y_i)$
     Compute Gap Target $\hat{\mathbb{G}}$
     $Threshold \leftarrow \max(C, \hat{\mathbb{G}})$
     Run PROCESS($x_i$)
     while Gap $\mathcal{G} > Threshold$
       Run REPROCESS
     end
     Periodically run CLEAN
```

Note from (18) that the α bounds for instances with $\beta = 0$ follow the formulation for the traditional convex setting. On the other hand, the bounds for the instances with $\beta = C$, that is, the outliers with score $(z < s)$ are assigned new bounds based on the Ramp Loss criteria. Once LASVM-NC establishes the α bounds for the new instance, it computes the Gap Target $\hat{\mathbb{G}}$ and takes a PROCESS step. Then, it makes optimizations of the REPROCESS kind until the size of the primal/dual gap comes down to the Gap Threshold. Finally, LASVM-NC periodically runs CLEAN operation to keep the size of the kernel expansion under control and to maintain its efficiency throughout the training stage.

6. LASVM with Ignoring Instances – LASVM-I

This SVM algorithm employs the Ramp function in Figure 3(a) as a filter to the learner *prior* to the PROCESS step. That is, once the learner is presented with a new instance, it first checks if the instance is on the ramp region of the function $(1 > y_i \sum_j \alpha_j K_{ij} > s)$. The instances that are outside of the ramp region are not eligible to participate in the optimization steps and they are immediately discarded without further action. The rationale is that the instances that lie on the flat regions of the Ramp function will have derivative $H^{'}(z) = 0$, and based on Equation 12, these instances will not play role in determining the decision hyperplane $\boldsymbol{w}$.

LASVM-I algorithm is also based on the following recordkeeping that we conducted when running LASVM-G experiments. In LASVM-G, we kept track of two important data points. First, we recorded the position of all instances on the Ramp Loss curve right before inserting the instance into the kernel expansion. Second, we kept track of the number of instances that were removed from the kernel expansion which were on the flat region of the Ramp Loss curve when they were admitted. The numeric breakdown is presented in Table 1.

Table 1: Analysis of Adult dataset in the end of the training of the models. “Admitted” column shows the number of examples that lie on the flat region (left and right) of the Ramp Loss (with $s = -1$) when they were inserted into the expansion. “Cleaned” column shows the number of examples removed during CLEAN.

	Expansion		Admitted		Cleaned	
	# SV	# Non-SV	Ramp(L)	Ramp(R)	Ramp(L)	Ramp(R)
FULL SVM	11831	20731	32562		0	
LASVM-G	11265	0	1340	20252	1	19562

Based on the distribution of these cleaned instances, it is evident that most of the cleaned examples that were initially admitted from ($z > 1$) region were removed from the kernel expansion with CLEAN at a later point in time. This is expected, since the instances with ($z > 1$) are already correctly classified by the current model with a certain confidence and hence do not become support vectors.

On the other hand, Table 1 shows that almost all of the instances inserted from left flat region (misclassified examples due to $z < s$) became SVs and therefore were never removed from the kernel expansion. Intuitively, the examples that are misclassified by a wide margin should not become support vectors. Ideally, the support vectors should be the instances that are within the margin of the hyperplane. As studies on Active Learning show (Ertekin et al., 2007; Schohn and Cohn, 2000), the most informative instances to determine the hyperplane lie within the margin. Thus, LASVM-I ignores the instances that are misclassified by a margin ($z < s$) up front and prevents them from becoming support vectors.

```
LASVM-I
  1) Initialization:
     Set $\boldsymbol{\beta} \leftarrow \mathbf{0}$, $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
  2) Online Iterations:
     Pick an example $x_i$
     Compute $z = y_i \sum_{j=0}^{n} \alpha_j K(x_i, x_j)$
     if ($z > 1$ or $z < s$)
        Skip $x_i$ and bail out
     else
        Compute Gap Target $\hat{\mathbb{G}}$
     $Threshold \leftarrow \max(C, \hat{\mathbb{G}})$
     Run PROCESS($x_i$)
     while Gap $\mathcal{G} > Threshold$
       Run REPROCESS
     end
     Periodically run CLEAN
```

Note that LASVM-I can not be regarded as a non-convex SVM solver since the instances with $\beta = C$ are already being ignored up front before the optimization steps. Consequently, all the instances visible to the optimization steps have $\beta = 0$, which converts objective function in (18) into the convex Hinge Loss from an optimization standpoint. Thus, combining these two filtering criteria ($z > 1$ and $z < s$), LASVM-I trades non-convexity with a filtering Ramp function to determine whether to ignore an instance or proceed with optimization steps. Our goal with designing LASVM-I is that, based on this initial filtering step, it is possible to achieve further speedups in training times while maintaining competitive generalization performance. The experimental results validate this claim.

7. LASVM-G without CLEAN – FULL SVM

This algorithm serves as a baseline case for comparisons in our experimental evaluation. The learning principle of FULL SVM is based on alternating between LASVM-G's PROCESS and REPROCESS steps throughout the training iterations. As in LASVM-G, the β's are initialized to zero and left that way throughout the online iterations, hence making the algorithm a Hinge Loss SVM. When a new example is encountered, FULL SVM computes the Gap Target (given in Eq. 8) and takes a PROCESS step. Then, it makes optimizations of the REPROCESS kind until the size of the primal/dual gap comes down to the Gap Threshold. In this learning scheme, FULL SVM admits every new training example into the kernel expansion without any removal step (i.e. no CLEAN operation).

```
FULL SVM
  1) Initialization:
     Set $\boldsymbol{\beta} \leftarrow \mathbf{0}$, $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
  2) Online Iterations:
     Pick an example $x_i$
     Compute Gap Target $\hat{\mathbb{G}}$
     $Threshold \leftarrow \max(C, \hat{\mathbb{G}})$
     Run PROCESS($x_i$)
     while Gap $\mathcal{G} > Threshold$
        Run REPROCESS
     end
```

This behavior mimics the behavior of traditional SVM solvers by providing that the learner has constant access to all training instances that it has seen during training and it can make any of them a support vector any time if necessary. The SMO-like optimization in the online iterations of FULL SVM enables it to converge to the batch SVM solution. Each PROCESS operation introduces a new instance to the learner, updates its α coefficient and optimizes the objective function. This is followed by potentially multiple REPROCESS steps, which exploit τ-violating pairs in the kernel expansion. Within each pair, REPROCESS selects the instance with maximal gradient, and potentially can zero the α coefficient of the selected instance. After sufficient iterations, as soon as a τ-approximate solution is reached, the algorithm stops updating the α coefficients. For full convergence to the batch SVM solution, running FULL SVM usually consists of performing a number of epochs where each epoch performs n online iterations by sequentially visiting the randomly shuffled training examples. Empirical evidence suggests that a single epoch yields a classifier almost as good as the SVM solution. For the theoretical explanation of the convergence results of the online iterations, please refer to (Bordes et al., 2005).

	Train Ex.	Test Ex.	# Features	C	$K(x,\bar{x})$
Adult (Census)	32562	16282	122	100	$e^{-0.005\|x-\bar{x}\|^2}$
Banana	4000	1300	2	10	$e^{-\|x-\bar{x}\|^2}$
Mnist (Digit 8)	60000	10000	784	100	$e^{-0.001\|x-\bar{x}\|^2}$
Reuters(Money-fx)	7770	3299	8315	1	$e^{-0.5\|x-\bar{x}\|^2}$
USPS	7329	1969	256	1	$e^{-2\|x-\bar{x}\|^2}$
USPS+N	7329	1969	256	1	$e^{-2\|x-\bar{x}\|^2}$

Table 2: Datasets and the train/test splits used in the experimental evaluations. The last two columns show the SVM parameters C and γ for the RBF kernel.

The freedom to maintain and access the whole pool of seen examples during training in FULL SVM does come with a price though. The kernel expansion needs to constantly grow as new training instances are introduced to the learner, and it needs to hold all non-SVs in addition to the SVs of the current model. Furthermore, the learner still needs to include those non-SVs in the optimization steps and this additional processing becomes a significant drag on the training time of the learner.

8. Experiments

The experimental evalutation involves evaluating these outlined SVM algorithms on various datasets in terms of both their classification performances and algorithmic efficiencies leading to scalability. In the experiments reported below, we run a single epoch over the training examples, all experiments use RBF kernels and the results averaged over 10 runs for each dataset. Table 2 presents the characteristics of the datasets and the SVM parameters for running the experiments. In LASVM-G, LASVM-NC and LASVM-I experiments, we empirically set the interval to perform CLEAN at every 300 new training instances.

Generalization Performances One of the metrics that we used in the evaluation of the generalization performances is Precision-Recall Breakeven Point (PRBEP), a widely used metric that measures the accuracy of the positive class where precision equals recall. Figure 4 shows the growth of PRBEP curves sampled over the course of training for the datasets. Compared to the baseline case FULL SVM, all algorithms are able to maintain competitive generalization performances in the end of training on all examples. Furthermore, LASVM-NC and LASVM-I actually yield better results on some datasets. This can be attributed to their ability to filter *bad* observations (i.e. noise) from training data. In noisy datasets, most of the noisy instances are misclassified and become support vectors in FULL SVM and LASVM-G due to the Hinge Loss. This increase in the number of support vectors (shown in Figure 6) causes the SVM to learn complex classification boundaries that can overfit to noise, which can adversely effect their generalization performances. LASVM-NC and LASVM-I are less sensitive to noise, and they learn simpler models that are able to yield better generalization performances under noisy conditions.

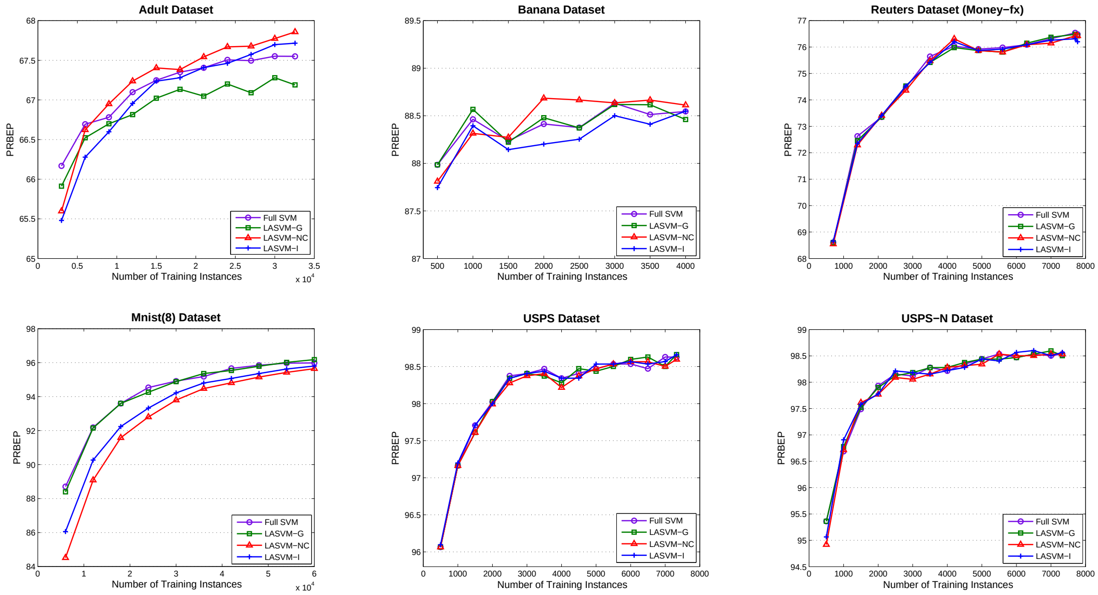


Figure 4: Precision/Recall Breakeven Point (PRBEP) vs. Number of Training Instances for all datasets. We used $s = -1$ for the Ramp Loss for LASVM-NC.

For the evaluation of classification performances, we report three other metrics, namely prediction accuracy, AUC and g-means in Table 3. Prediction accuracy measures a model's ability to correctly predict the class labels of unseen observations. The *area under the ROC curve* (AUC) is a numerical measure of a model's discrimination performance and shows how correctly the model separates the positive and negative observations and ranks them. g-means is the geometric mean of *sensitivity* and *specificity* where sensitivity is the accuracy on the positive instances, and specifity is the accuracy on the negative instances. We report that all LASVM algorithms yield as good results for these performance metrics as FULL SVM. Further, as is the case for PRBEP, LASVM-NC and LASVM-I achieve better results on these metrics for some datasets than FULL SVM and LASVM-G.

We study the impact of the s parameter on the generalization performances of LASVM-NC and LASVM-I and present our findings in Figure 5. Since FULL SVM and LASVM-G do not use Ramp Loss, they are represented with their testing errors and total number of support vectors achieved in the end of training. The Banana dataset shows a clean separation of LASVM-NC and LASVM-I plots, with LASVM-NC curve under the LASVM-I curve. This indicates that LASVM-NC achieves higher classification accuracy with fewer support vectors for all s values for this dataset. In all datasets, increasing the value of s into the positive territory actually has the effect of preventing correctly classified instances that are within the margin from becoming SVs. This becomes detrimental to the generalization performance of LASVM-NC and LASVM-I since those instances are among the most informative instances to the learner. Likewise, moving s into further down to the negative territory diminishes the effect of the Ramp Loss on the outliers. If $s \to -\infty$, then $R_s \to H_1$; in other words, if s

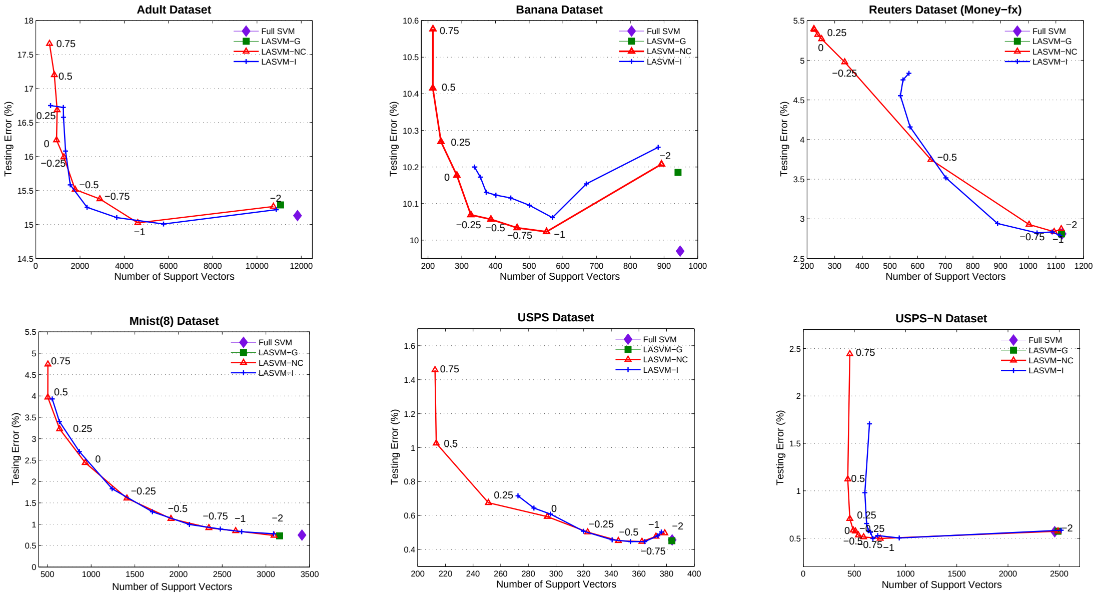


Figure 5: Testing Error vs. Number of Support Vectors for various settings of the s parameter of the Ramp Loss.

takes large negative values, the Ramp Loss will not help to remove outliers from the SVM kernel expansion.

It is important to note that at the point $s = -1$, the algorithm behaves as an *Active Learning* (Schohn and Cohn, 2000; Tong and Koller, 2001) framework. Active Learning is widely known as a querying technique for selecting the most informative instances from a pool of unlabeled instances to acquire their labels. Even in cases where the labels for all training instances are available beforehand, active learning can still be leveraged to select the most informative instances from training sets. In SVMs, informativeness of an instance is synonymous with its distance to the hyperplane and the instances closer to the hyperplane are the most informative. For this reason, traditional SVM based active learners focus on the instances that are within the margin of the hyperplane and pick an example from this region to process next by searching the entire training set. However, such an exhaustive search is impossible in the online setup and computationally expensive in the offline setup. Ertekin et al. (2007) suggest that querying for the most informative example does not need to be done from the entire training set, but instead, querying from randomly picked small pools can work equally well in a more efficient way. *Small pool active learning* first samples M random training examples from the entire training set and selects the best one among those M examples. With probability $1 - \eta^M$, the value of the criterion for this example exceeds the η-quantile of the criterion for all training examples regardless of the size of the training set. In practice this means that the best example among 59 random training examples has 95% chance to belong to the best 5% examples in the training set.

In the extreme case of small pool active learning, setting the size of the pool to 1 corresponds to investigating whether that instance is within the margin or not. In this

Table 3: Experimental Results for the Datasets and all Four SVM algorithms. The First Four Metrics are for Generalization Performance and the rest are for Computational Efficiency.

		Datasets					
		Adult	Mnist(8)	Banana	Reuters	USPS	USPSN
Accuracy	FULL SVM	84.87	99.25	90.03	97.19	99.54	98.43
	LASVM-G	84.81	99.27	89.81	97.19	99.54	99.42
	LASVM-NC	85.01	99.15	89.97	97.16	99.52	99.51
	LASVM-I	84.82	99.18	89.84	97.16	99.57	99.49
PRBEP	FULL SVM	67.55	95.98	88.54	76.48	98.62	98.53
	LASVM-G	67.18	96.18	88.46	76.42	98.66	98.50
	LASVM-NC	67.85	95.64	88.61	76.42	98.59	98.53
	LASVM-I	67.71	95.80	88.54	76.20	98.65	98.56
AUC	FULL SVM	0.897	0.998	0.965	0.987	0.999	0.998
	LASVM-G	0.893	0.998	0.966	0.987	0.999	0.998
	LASVM-NC	0.901	0.998	0.965	0.987	0.999	0.998
	LASVM-I	0.899	0.998	0.964	0.987	0.999	0.998
Gmeans	FULL SVM	73.13	97.29	89.51	81.00	98.93	98.42
	LASVM-G	72.87	97.42	89.30	81.17	98.92	98.42
	LASVM-NC	75.03	96.86	89.47	81.19	98.99	98.74
	LASVM-I	73.72	97.06	89.38	81.06	98.89	98.72
# SV	FULL SVM	11831	3412	947	1122	384	2455
	LASVM-G	11266	3157	941	1120	383	2288
	LASVM-NC	**4609**	**2653**	**551**	**1086**	**372**	**752**
	LASVM-I	5776	2722	669	1093	373	937
# Kernel ($x10^6$)	FULL SVM	709.0	2269.7	8.51	33.2	27	30.2
	LASVM-G	233.0	182.6	3.8	9.9	5.2	14.3
	LASVM-NC	116.9	153.5	3.1	9.9	5.2	7.3
	LASVM-I	**105.9**	**121.2**	**2.0**	**8**	**3.2**	**6.8**
Train Time	FULL SVM	1186	4757.4	1	14.4	43.7	36
	LASVM-G	479	547.1	0.6	5.3	17.3	17
	LASVM-NC	129	526.0	0.4	4.7	8	8.3
	LASVM-I	**92**	**491.4**	**0.3**	**3.6**	**5.8**	**6**

regard, setting $s = -1$ for the Ramp Loss in LASVM-NC and LASVM-I constrains the learner's focus only on the instances within the margin. Empirical evidence suggests that LASVM-NC and LASVM-I algorithms exhibit the benefits of active learning at $s = -1$ point, which seems to yield optimal results in most of our experiments. However, the exact setting for the s hyperparameter should be determined by the requirements of the classification task and the characteristics of the dataset.

Computational Efficiency A significant time consuming operation of SVMs is the computation of kernel products $K(i, j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$. For each new example, its kernel product

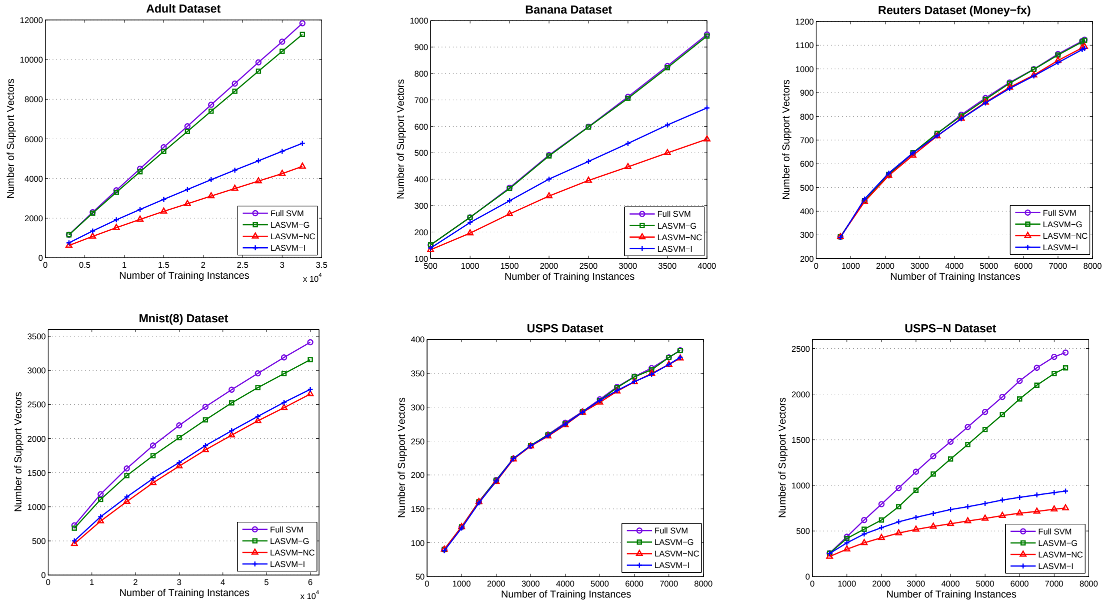


Figure 6: Number of Support Vectors vs. Number of Training Instances

with every instance in the kernel expansion needs to be computed. By reducing the number of kernel computations, it is possible to achieve significant computational efficiency improvements over traditional SVM solvers. In Figure 7, we report the number of kernel calculations performed over the course of training iterations. FULL SVM suffers from uncontrolled growth of the kernel expansion, which results in steep increase of the number of kernel products. This also shows why SVMs can not handle large scale datasets efficiently. In comparison, LASVM-G requires fewer kernel products than FULL SVM since LASVM-G keeps the number of instances in the kernel expansion under control by periodically removing uninformative instances through CLEAN operations.

LASVM-NC and LASVM-I yield significant reduction in the number of kernel computations and their benefit is most pronounced in the noisy datasets, Adult, Banana and USPS-N. LASVM-I achieves better reduction of kernel computations than LASVM-NC. This is due to the aggressive filtering done in LASVM-I where no kernel computation is performed for the instances on the flat regions of the Ramp Loss. On the other hand, LASVM-NC admits those instances into the kernel expansion but achieves sparsity through the non-convex optimization steps. The reason for the low number of kernel products in LASVM-NC is due to its ability to create sparser models than other three algorithms. A comparison of the growth of the number of support vectors during the course of training is shown in Figure 6. LASVM-NC and LASVM-I end up with smaller number of support vectors than FULL SVM and LASVM-G for all datasets. Furthermore, compared to LASVM-I, LASVM-NC builds noticeably sparser models with less support vectors in noisy Adult, Banana and USPS-N datasets. LASVM-I, on the other hand, makes fewer kernel calculations than LASVM-NC for those datasets. This is a key distinction of these two algorithms: The computational efficiency of LASVM-NC is the result of its ability to build sparse models. Conversely, LASVM-I creates comparably more support vectors than LASVM-NC, but makes fewer ker-

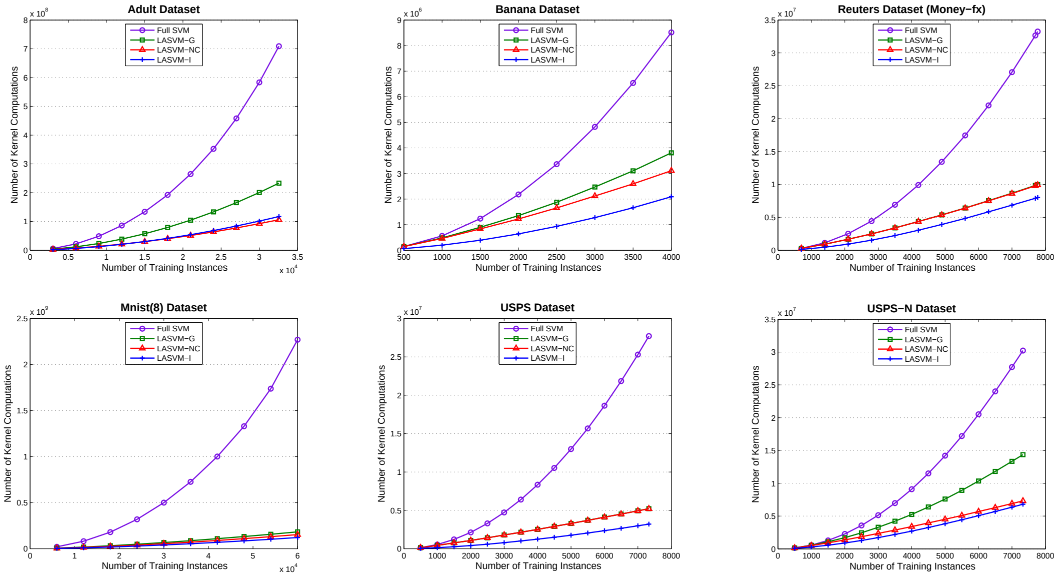


Figure 7: Number of Kernel Computations vs. Number of Training Instances

nel calculations due to early filtering. The overall training times for all datasets and all algorithms are presented both in Figure 8 and Table 3. All three LASVM algorithms are significantly more efficient than FULL SVM. The fastest training times belong to LASVM-I and LASVM-NC comes close second. The sparsest solutions are achieved by LASVM-NC and this time LASVM-I comes close second. These two algorithms represent a compromise between training time versus sparsity and recognition time, and the appropriate algorithm should be chosen based on the requirements of the classification task.

9. Conclusions

In traditional convex SVM optimization, the number of support vectors scales linearly with the number of training examples, which unreasonably increases the training time and computational resource requirements. This fact has hindered widespread adoption of SVMs for classification tasks in large-scale datasets. In this work, we have studied the ways in which the computational efficiency of an online SVM solver can be improved without sacrificing the generalization performance. This paper is concerned with suppressing the influences of the outliers, which particularly becomes problematic in noisy data classification. For this purpose, we first present a systematic optimization approach for an online learning framework to generate more reliable and trustworthy learning models in intermediate iterations (LASVM-G). We then propose two online algorithms, LASVM-NC and LASVM-I, which leverage the Ramp function to avoid the outliers to become support vectors. LASVM-NC replaces the traditional Hinge Loss with the Ramp Loss and brings the benefits of non-convex optimization using CCCP to an online learning setting. LASVM-I uses the Ramp function as a filtering mechanism to discard the outliers during online iterations. Empirical evidence suggests that the algorithms provide efficient and scalable learning with noisy datasets in two respects: i) *computational*: there is a significant decrease in the number of

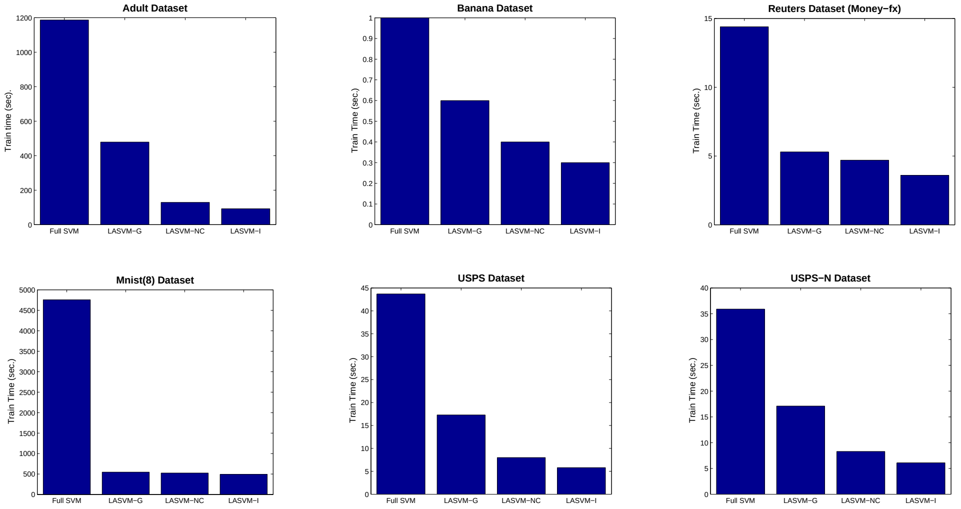


Figure 8: Training times of the algorithms for all datasets after one pass over the training instances. The speed improvement in training time becomes more evident in larger datasets.

computations and running time during training and recognition, and *ii*) *statistical*: there is a significant decrease in the number of examples required for good generalization. Our findings also reveal that discarding the outliers by leveraging the Ramp function is closely related to the working principles of margin based Active Learning.

References

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

Olivier Bousquet and Andre Elisseeff. Stability and generalization. *Journal of Machine Learning*, 2, 2002.

Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Trading convexity for scalability. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 201–208. ACM, 2006.

Corinna Cortes and Vladimir Vapnik. Support vector networks. *Machine Learning*, 20: 273–297, 1995.

Seyda Ertekin, Jian Huang, Leon Bottou, and Lee Giles. Learning on the border: active learning in imbalanced data classification. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 127–136, New York, NY, USA, 2007. ACM.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

Nir Krause and Yoram Singer. Leveraging the margin more carefully. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 63, New York, NY, USA, 2004. ACM.

Dale Schuurmans Linli Xu, Koby Cramer. Robust support vector machine training via convex outlier ablation. In *Twenty-First National Conference on Artificial Intelligence (AAAI)*, 2006.

Yufeng Liu, Xiaotong Shen, and Hani Doss. Multicategory ψ learning and support vector machine: Computational tools. *Journal of Computational and Graphical Statistics*, 14: 219–236, 2005.

Llew Mason, Peter L. Bartlett, and Jonathan Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 38:243–255, 2000.

Fernando Perez-Cruz, Angel Navia-Vazquez, and Anibal R. Figueiras-Vidal. Empirical risk minimization for support vector classifiers. *IEEE Tran. on Neural Networks*, 14, 2002.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. pages 185–208, 1999.

Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 839–846, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, USA, 2002.

J. Shawe-Taylor and N. Cristianini. Kernel methods for pattern analysis. 2004.

Xiaotong Shen, George C. Tseng, Xuegong Zhang, and Wing Hung Wong. On psi-learning. *Journal of the American Statistical Association*, 98:724–734, January 2003.

A. Smola, S. Vishwanathan, and T. Hofmann. Kernel methods for missing variables. In *Proc. of the Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.

Ingo Steinwart. Sparseness of support vector machines. *Journal of Machine Learninng Research*, 4:1071–1105, 2003.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

Lei Wang, Huading Jia, and Jie Li. Training robust support vector machine with smooth ramp loss in the primal space. *Neurocomputing*, pages 3020 – 3025, 2008.

Alan L. Yuille and Anand Rangarajan. The concave-convex procedure (CCCP). In Thomas G. Dietterich, Sue Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.