Learning from Mistakes: Towards a Correctable Learning Algorithm

Karthik Raman
Cornell University
Ithaca, NY, USA
karthik@cs.cornell.edu

Krysta M. Svore, Ran Gilad-Bachrach, Chris J.C. Burges
Microsoft Research
Redmond, WA, USA
{ksvore,rang,cburges}@microsoft.com

ABSTRACT

Many learning algorithms generate complex models that are difficult for a human to interpret, debug, and extend. In this paper, we address this challenge by proposing a new learning paradigm called *correctable learning*, where the learning algorithm receives external feedback about which data examples are incorrectly learned. We define a set of metrics which measure the *correctability* of a learning algorithm. We then propose a simple and efficient correctable learning algorithm which learns *local models* for different regions of the data space. Given an incorrect example, our method samples data in the neighborhood of that example and learns a new, more correct *local* model over that region. Experiments over multiple classification and ranking datasets show that our correctable learning algorithm offers significant improvements over the state-of-the-art techniques.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; H.3.3 [**Information Systems and Retrieval**]: Search Process

General Terms

Algorithms, Experimentation, Theory

Keywords

Classification, Regression, Correctable learning

1. INTRODUCTION

Large machine learning systems are increasingly common across a variety of tasks including Web Search, Advertising, Social Networking, and Collaborative Filtering. Although ensemble methods such as boosted trees [6] and random forests are widely used for these tasks [3], these algorithms have several drawbacks: (1) Training and testing is slow, (2) The learned models are difficult to interpret, (3) The models are not well-suited for parallelization, (4) It is difficult to incorporate feedback without retraining. Consequently, debugging and correcting these models can be challenging.

A particularly poignant example stems from Web Search. Training a commercial search engine over millions of example queries is time-consuming. When a search engine user reports that a query has failed, meaning that the returned URLs do not fulfill the intent of the query, it can be difficult to fix that particular query during training of the ranking model. Namely, feedback on a single query has very little effect among a pool of millions of training examples; even if upon retraining of the model the feedback is added to the training set, it is unlikely to change the learned model at all. Furthermore, the models are extremely complicated, non-linear, and non-intuitive to a human, making it almost impossible to debug a failed query. How can we learn a model that is easier to correct, and what are desirable properties for such a model?

In response to these challenges, we formalize a new learning paradigm called *correctable learning.* Our focus is threefold: (1) we focus on learning in the presence of feedback, (2) we focus on addressing the poor-performing regions highlighted by this feedback by "learning from our mistakes", and (3) we develop criteria and metrics with which to evaluate the correctability of a model. We aim to fix examples on which the current model makes a mistake, denoted as **MSTKs**, without hurting other examples.

Our approach *localizes* the effect of a training point, such that it only affects the performance of its local model and points within its neighborhood. Our method partitions the dataspace into *regions* and learns a separate *local model* for each region. The key insight is that since there are relatively few points "close" to the decision boundary defined by any local model, changing the boundary of a local model is likely to have less impact on other points than changing the boundary of a single *global* model. We thus achieve correctability by incrementally adding MSTK points to our training data. Our method outperforms the *correctability* and performance of far more complicated non-linear models.

2. RELATED WORK

Active Learning: Unlike *passive* supervised learning, the goal of active learning is to both identify the data points to label and learn a good model from them. A common way to identify points is to choose the points with the most uncertainty in the labels [5]. Our correctable learning method also learns incrementally as MSTK points are added, but there is no control over the next received point.

Online Learning: In online learning [4], examples are revealed one at a time. The goal is to efficiently and incrementally learn through simple updates to the model. There are two key differences between (stochastic) online learning

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
Copyright 2012 ACM 978-1-4503-1156-4/12/10 ...$15.00.

and correctable learning: (1) online learning typically assumes that points come from the same underlying distribution, while we do not assume that MSTK points match the underlying distribution, and (2) most online learning methods do not store, and are thus prone to making mistakes on, previously seen data, while instead we target generalization and do not allow performance degradation on previously seen data.

Localized Learning: Localized learning methods consider the local neighborhood of a given point when predicting its label and have been shown successful when used with various underlying learning techniques [9]. It has been shown that consistent classifiers must be localized [11], and that localizing linear methods such as SVMs [8] results in performance comparable to more complex methods. *Lazy Learning* [1] involves no *a priori* training; at test time, a classification rule is determined based on the region around the test point. Our approach is similar in spirit to lazy learning.

3. CORRECTABLE LEARNING

Today's large-scale learning systems use complex learning algorithms and vast amounts of training data to maximize performance. Aside from retraining an entirely new model, it is almost impossible to modify the learned models in an intuitive manner due to their complexity. This is particularly problematic when mistakes that the system makes upon deployment need to be corrected. For example, consider this real-world scenario: a senior administrator of a large search engine discovers one morning that the ranking results for an important query (*e.g., british airlines*) are poor. Due to the scale of the search engine, there is no easy, *principled* way to fix this query. While we could try using a *whitelist* (list of known *important* results for such queries), doing so would not generalize. Learning a new model typically would take hours, if not days, which would be too long and may not even fix the issue. Even if it did, retraining may very likely adversely impact the ranking quality of another important query, say *superbowl sunday.* In addition, the administrator would hope that fixing the original mistake on *british airlines* would result in improvements to most similar queries with mistakes, fixing say *delta airlines* if it also was of poor quality.

Such a scenario motivates the need for what we refer to as **Correctable Learning**. We believe this is a key problem that could be of significant interest to the learning community, and especially the IR community, in particular due to its potential for high impact on today's large-scale learning systems. To the best of our knowledge, this problem has not been previously identified or studied in the literature. Our main focus is to introduce and define the problem of correctable learning, and propose ways to measure correctability of a learned model. Below we introduce our notation:

$t : \{0 \leq t \leq n\}$	Time
$k_i = (x_i, y_i^*)$	i^{th} Mistake, *i.e* at time i.
τ	Mistake Threshold
M_0	Initial model learned
M_i $\{i \geq 1\}$	Model after i^{th} MSTK seen
$y_i = M_{i-1}(x_i)$	Predicted (Wrong) Label for i^{th} MSTK (*i.e.*, just before seeing it)
$\Delta(y, y^*) \in \mathbb{R}$	Loss function
$P_T(M)$	Performance of model M on test set *i.e.* $\sum_{(x,y^*)\in T} \Delta(M(x), y^*)$

Definition 1. *Given learned model M, we define point (x, y^*) to be a mistake point* (**MSTK**)*, where x is the feature vector and y^* is the true label,* **iff** $\Delta(y, y*) \geq \tau$*, where $M(x) = y$, y is the predicted label, Δ is the loss function, and τ is a pre-defined "error" threshold.*

Note that for binary classification $\Delta(y, y^*)$ is the *0-1* loss function with $\tau = 1$ while for ranking we use $1 - NDCG@3$. Definition 1 essentially says a MSTK is a point that the model performs *poorly* on, where τ quantifies how poorly.

Definition 2. *Given the current learned model M and a MSTK point $k = (x_k, y_k^*)$, we define* **correctable learning** *to be the learning of a new model M' (via modification of model M) such that the following hold:*

MSTK corrected: $\Delta(M'(x_k), y_k^*) < \tau$.

Stability maintained: $M'(x) = M(x)$ *is guaranteed for a previously known (large) fraction of the training samples, and also for a known (large) fraction of the already observed test samples.*

Similar MSTKs fixed: *Empirical risk of M' on test samples in the neighborhood of x_k is reduced.*

Similar complexity: *Model complexity increase is bounded by a small (pre-determined) amount.*

As illustrated in the search engine example, only when all of the conditions are met can we be satisfied with the *correctability* of the algorithm. Given such an algorithm and a stream of MSTKs, we can hope to improve the performance of the learned system over time, without having to retrain a new model from scratch every time a MSTK is received.

4. A CORRECTABLE ALGORITHM

We would like an algorithm that, given a MSTK point, updates the current model in a simple manner while achieving the conditions required for *correctability.* A simple approach is to use an existing algorithm and add MSTK points to the training data as they are received in an online fashion. However, this may not correct the mistake since so many points influence the model, or it may cause performance on other points to drop (if many points are close to the decision boundary, then changing the boundary can impact performance). We can overcome these two problems by using disjoint models for different regions of the data space, which reduces the number of points affecting each decision boundary (thus allowing for correction of a point), and also reduces the number of points close to any boundary (reducing the risk of worsening the performance of other points near the boundary).

In response, we propose a localized-learning-based algorithm **LocCL**, detailed in Algorithm 1. First, we precompute a *good region function*, which partitions the dataspace into N regions[1]. For instance, this can be done using any clustering algorithm, where the regions are given by the clusters. Next, we train separate models over each region, which we call *local models*, using an existing learning algorithm L[2]. Testing returns the predicted label of a given point using the model for the region in which the point belongs. To achieve *correctability*, we add the MSTK point x to the training set

[1] In our experiments, this is done once at the start. This can easily be extended to a dynamic partitioning which changes with data or depends on the task.

[2] We assume $\text{Train}_L(D)$ returns a model trained on dataset D; $\text{Test}_L(M, x)$ returns the prediction for test point x using model M. We refer to L as the **Base Learning Method**.

Algorithm 1 Correctable Learning Framework.

```
Notation: $D$: Training Data,
$N$: No. Regions, $F: D \to \{1, .., N\}$: Region Fctn.,
$L$: Learning Method, $\{M_1, ..., M_N\}$: Local Models,
$D_j = \{(x, y) | x \in D, F(x) = j\}$ : Local Training Data
TRAINING:
for $j = 1$ to $N$ do
    Output: $M_j = \text{Train}_L(D_j)$
TESTING:
Input: Datapoint $x$
Output: $\text{Test}_L(M_{F(x)}\ , x)$
CORRECTION:
Input: MSTK $k = (x, y)$
$D_{F(x)} = D_{F(x)} \cup \{(x, y)\}$.
Update: $M_{F(x)} = \text{Train}_L(D_{F(x)})$.
```

of the region in which it belongs and update that local model using the modified training set (as done in online learning).

Note that while the proposed algorithmic framework is simple, it is a step towards obtaining an efficient algorithm for this new learning paradigm. We also note that since this framework relates closely to importance weighting, there may exist alternate approaches to this problem as well.

In addition to correctability, our proposed method has the following advantages:

Training is faster since each training set contains fewer elements. Thus with N regions, an order d polynomial time algorithm roughly achieves a speed-up factor of N^d with parallelization and N^{d-1} without it.

Parallelization is easy since training each local model is an independent process. This is especially useful for methods that are inherently hard to parallelize.

Labeling cost savings are large since like active learning, desired performance can be achieved using much less labeled data.

Flexibility is inherent since local models are independent; this allows for different feature sets to be used in different parts of the data space. This is particularly desirable for a search engine, where different queries may require different features.

Further, our experiments show that *using linear methods* for learning can achieve comparable performance to ensemble methods, while producing more interpretable models and improved training times. Due to the partitioning of the data, performance in any region depends only on a single model, thus allowing for a better understanding of why we perform poorly over a region, thus making it easier to debug. However, one should refrain from having too many regions which will result in excess model complexity and over fitting. Overfitting can be controlled (to an extent), using the validation set or by having a good partitioning of the data. Smoothness of models across regions could also help.

5. MEASURING CORRECTABILITY

A key contribution of our work is a set of metrics which measure the correctability of a learning algorithm. The definitions for these metrics are task-independent and can be applied to ranking, binary and multi-class classification, and so on.

METRIC 1. **Average Correction Rate (ACR)** *is defined as the average change in error of a MSTK after correcting it:* $ACR = \frac{1}{n}\sum_{i=1}^{n}\Big(\Delta(M_{i-1}(x_i), y_i^*) - \Delta(M_i(x_i), y_i^*)\Big)$.

METRIC 2. **Average Performance Instability (API)** *is defined as the average drop in test-set performance, i.e.,* $API = \frac{1}{||W||}\sum_{i \in W}\Big(P_T(M_{i-1}) - P_T(M_i)\Big)$, *where* $W = \{i : P_T(M_i) < P_T(M_{i-1})\}$.

METRIC 3. **Overall Performance Gain (OPG)** *is defined as the overall increase in test-set performance over the correction process, i.e.,* $OPG = \frac{1}{n}\Big(P_T(M_0) - P_T(M_n)\Big)$.

These measures address the key requirements of correctable learning:

Correct the MSTK: ACR measures how often the learning method *corrects* a MSTK. A higher value indicates the ability of the method to easily fix MSTKs.

Do not hurt others: As we do not want performance to worsen elsewhere, we use the **API** metric to measure any decrease in test-set performance. A lower value indicates the method is less likely to negatively impact performance of others.

Learn from the MSTK: As we would like to learn from the MSTK and correct other similar errors, we use the **OPG** to measure the improvement in performance over the duration of the correction process. A higher value indicates better learning from the MSTKs.

6. EXPERIMENTS

In this section, we use our correctability metrics to evaluate different learning methods versus our proposed **LocCL** algorithm. We also evaluate the accuracy of the algorithms across several binary classification and ranking datasets.

To evaluate correctability of MSTKs in a fair setting, we divide each dataset into 4 parts:

1. *Start-Train*(**STr**): For training initial models ($t = 0$).
2. *MSTK-Pool* (**MPl**): Data from which MSTKs are drawn.
3. *Validation* (**Val**): Data for parameter validation.
4. *Test* (**Tes**) : Held-out data used for evaluation.

This split is performed randomly. For the ranking datasets, we use the provided validation and test sets, and randomly split the train set into the **STr** and **MPl** sets.

We first train a model on the **STr** set. Next, we iteratively draw MSTKs from the **MPl** set and call the correction routine of the algorithm. Among all possible MSTKs in the current MSTK pool, we choose one at random. Note that the set of MSTKs depends on the current classifier, and hence this set will vary for the different methods. This process is repeated until the total number of labeled points used for training reaches the predetermined **budget** (or there are no more MSTKs to be found).

We experimented on multiple standard datasets for these tasks, as given in Table 1. For classification, we chose handwriting recognition datasets to study correctability since a human can easily recognize errors and provide MSTKs to the system. We split the true labels into positive and negative sets and perform binary classification.

We report performance on our three correctability metrics and on classification accuracy (denoted by **Perf**)[3]. We choose parameters using the validation set. We partition the dataspace using the K-Means++ algorithm [2] ($k = 20$[4]). In

[3] Accuracy, ACR and OPG are reported as percentages.

[4] $k = 20$ was chosen arbitrarily. For OptDig we used $k = 10$ as $k = 20$ produced many empty clusters.

Dataset	**Description**	**Task**	**#Feat**	**|STr|**	**|Val|**	**|MPl|**	**|Tes|**	**Budget**
MNIST	Digit-Recognition	Classification	784	2400	12000	45600	10000	5960
USPS	Digit-Recognition	Classification	256	210	465	3975	4650	690
OptDig	Digit-Recognition	Classification	64	172	383	3440	1800	400
Letter	Letter-Recognition	Classification	16	800	2000	15200	2000	1800
MQ7	LETOR4-Million Query 07	Ranking	46	50	339	967	336	300
MQ8	LETOR4-Million Query 08	Ranking	46	50	157	421	156	200
YL1	Yahoo LTR Set 1	Ranking	519	200	2994	19744	6983	320
YL2	Yahoo LTR Set 2	Ranking	596	50	1266	1216	3798	150

Table 1: Datasets Used in experiments

Data	Existing Methods - Baseline						LocCL				
	Base Method	**ACR**	**API**	**OPG**	**Perf**	**NoCL**	**ACR**	**API**	**OPG**	**Perf**	**NoCL**
MNIST	Perceptron	87.50	5.13	**7.40**	82.80	79.73	81.64	0.28	4.25	89.52	86.44
	Linear-SVM (LSVM)	32.45	0.138	-0.57	85.68	86.75	79.56	0.027	1.87	95.63	94.36
	Kernel-SVM	-	-	-	96.20	-	**99.1***	**0.024**	3.50	**97.73***	96.10
USPS	Perceptron	87.50	2.48	**9.20**	80.20	79.58	83.80	0.25	7.31	90.65	86.15
	Linear-SVM (LSVM)	55.60	0.448	0.45	82.92	85.22	88.17	0.140	4.43	95.53	94.25
	Kernel-SVM	-	-	-	95.28	-	**97.9***	**0.130**	7.23	**96.26***	92.51
OptDig	Perceptron	89.80	1.63	4.47	79.65	79.30	**90.50**	0.69	2.46	92.42	87.70
	Linear-SVM (LSVM)	57.14	0.528	-1.00	84.70	86.03	76.34	**0.220**	1.95	94.33	91.99
	Kernel-SVM	-	-	-	96.33	-	79.40	0.240	**4.79**	**96.39**	93.43
Letter	Perceptron	81.90	2.12	-4.2	64.95	66.27	82.10	2.04	-0.80	67.70	71.00
	Linear-SVM (LSVM)	29.30	0.575	-3.01	69.41	71.42	66.60	0.189	3.25	82.12	82.06
	Kernel-SVM	-	-	-	90.41	-	**98.8***	**0.110**	**10.27**	**94.58***	88.25

Table 2: Correctability for classification and LocCL methods. Perf is test-set accuracy. NoCL is performance on an unbiased dataset without addition of MSTKs. *Kernel* Perf is accuracy when trained once on an unbiased sample (with size equal to the budget). Bold indicates best value, * indicates stat. sig. over best baseline.

the correction step of our method we add only the MSTK point to obtain a new model (though we study the effect of adding neighbors later). Results reported are an average of four independent runs[5].

6.1 Correctability of Existing Methods

First we study how existing classification methods perform on the task of correctable learning. We use linear SVMs[6] [10], and two non-linear methods: Kernel SVMs (representative of supervised learning) and the perceptron (representative of online learning). Since we want to learn from MSTKs, the simplest correction step is retraining on all given training data (including the incoming MSTK points). We then compute the different correctability metrics, shown in Table 2.

As shown by the **OPG** measure, on many of the datasets performance of the existing methods actually decreases over the duration of this correctability process. In particular, for the linear SVM we find that the correction rate for the various datasets is low — on average less than half of the MSTKs are corrected. Furthermore, we observe that for all datasets, performance of the linear SVM after the correction process (shown in the **Perf** column) is worse than using an unbiased sample of the dataset of the same size (shown in the **NoCl** column). Note that the results in the **Perf** and **NoCl** columns are on the **same** test set. The poor performance of these methods under correction shows that they are not well-suited to learn from such an adversarial data distribution, which is created due to the addition of MSTKs.

We find that the perceptron exhibits slightly better correctability; the method is able to improve overall performance, and achieve high correction and low volatility. However, despite high **OPG** values, it still performs poorly overall and is **significantly** worse than the Linear SVM. While it achieves a high correction rate, it does so by overcompensating for the MSTKs, as seen by the large **API** values, which indicate that the performance is volatile and tends to oscillate. This is true even for Linear SVMs (albeit on a smaller scale), as seen in the left panel of Fig. 1. Overcompensation is the primary reason why online learning methods are unsuitable for correctable learning.

Determining the correctability of Kernel SVMs was **infeasible** due to the large computational overhead incurred by the constant retraining of models as MSTKs are added (even for the small datasets). This points to why more complex methods, such as mixture-of-experts or nearest-neighbor based methods, are not suitable for this task (and hence not compared against), and do not generalize to other tasks such as ranking and regression.

6.2 Correctability of LocCL: Classification

To overcome the problem of learning from a biased distribution, we use our localized correctable learning method (*LocCL*). We experimented with different *base-learning* methods: Linear SVMs, Perceptron and Kernel SVMs. As seen in Table 2, LocCL performs significantly better on the correction metrics compared to the corresponding baseline methods. In particular, LocCL-LSVM outperforms Linear SVMs on all correction metrics for all datasets, indicating that it is not only more stable, but also better at correcting MSTKs while improving overall performance. In fact, for three datasets, our method significantly outperforms computationally more intense methods like Kernel SVMs and has complexity similar to Linear-SVMs[7](since the training sets for each SVM is much smaller than the original set) and

[5] We do not report the variance in the numbers for brevity, but found them to be small.

[6] http: //svmlight.joachims.org/svm_perf.html

[7] We found that the LocCL-LSVM method to be significantly faster than Kernel-SVMs without parallelization.

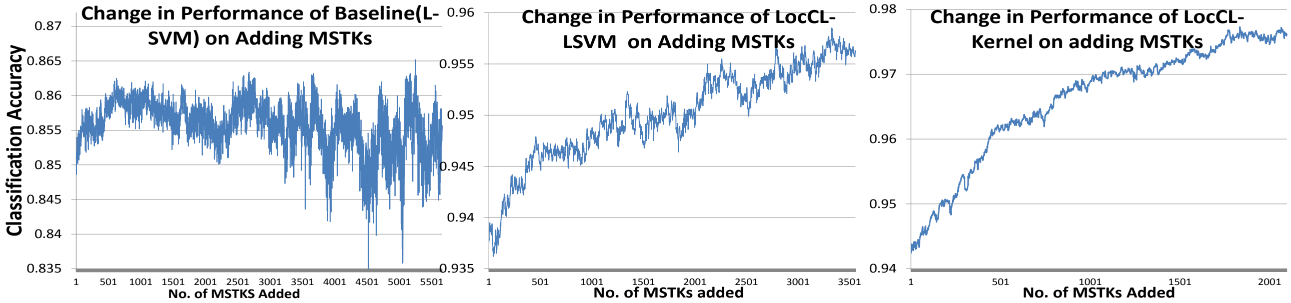

Figure 1: Change in MNIST test acc. on adding mistakes for *a) Baseline*; *b) LocCL-LSVM* ; *c) LocCL-Kernel*

the added advantage of being parallelizable. Given that Perf is better than NoCL for all cases, it is apparent that mistake-based learning provides significant advantages in the LocCL framework.

Next, we ran experiments with Kernel SVMs as the base method[8]. As seen in Table 2, the correction rates are highest (with almost perfect **ACR**) and the volatility is lowest for 3 of the 4 datasets. This is further seen in Figure 1 (*Right*), where LocCL-Kernel improves on the already low volatility of the LocCL-LinearSVM method (*Middle*). Finally, the performance achieved by the LocCL-Kernel method is the *highest* among studied methods across *all* datasets, despite using far less training data[9].

Data	SVM-Rank			LocCL		
	ACR	API	OPG	ACR	API	OPG
MQ7	1.39	0.26	0.78	**5.51***	0.26	**6.14**
MQ8	4.30	0.53	1.30	**7.50***	**0.28**	**1.90**
YL1	9.17	0.20	0.58	**10.90**	**0.05**	**1.60**
YL2	6.45	0.33	0.57	**19.30***	**0.17**	**2.50**

Table 3: Correctability performance of LocCL for the ranking datasets for NDCG@3.

6.3 Correctability of LocCL: Ranking

For ranking, we studied SVM-Rank[7], a well-known baseline for the LETOR datasets, as the base method for LocCL. The C parameter (10^{-5} to 10^{-3}) is chosen using the validation data. To obtain a *partitioning*, we clustered queries using features which depend only on the query. While the *MQ7* and *MQ8* datasets have 5 such features, the *YL1* and *YL2* have 20 and 18 respectively. For all datasets, we ran the clustering algorithm after normalizing the query-only features to zero mean, unit variance. Since the number of queries is not large, we set $k = 5$ in the clustering algorithm. Here, MSTKs for the current ranker are those examples with an NDCG@3$\leq$ 0.2. This corresponds to queries for which the initial set of results is of poor quality (and thus corresponds well to what a human would flag as a MSTK).

The results are shown in Table 3. We see that the baseline is unable to improve much via the mistake-based learning as seen from the low **OPG** scores. As seen from the API and ACR values, the baseline SVM-Rank model is quite volatile and is unable to significantly correct the MSTKs. On the other hand, we find that the LocCL method is able to significantly outperform the baseline with regards to the ACR, API and OPG correctability metrics. In particular, we find the ACR scores to be significantly better on all datasets, thus validating the claim that the method can better *fix* the MSTK points. We also find that the method is far less volatile on 3 of the 4 datasets, and is able to gain from the mistake-based learning with much larger OPG scores.

7. CONCLUSIONS AND FUTURE WORK

We have defined a new machine learning paradigm, *correctable learning*, that is strongly motivated by real-world challenges faced by practitioners. To evaluate characteristics of algorithms within this new paradigm, we have introduced three novel correctability metrics. We find that existing algorithms are not suitable for this problem, so in response, we have proposed a localized-learning-based framework that partitions the data-space into regions and learns local models over each region while correcting the models based on feedback. In addition to yielding improved correctability and performance, our method provides other benefits, including *interpretability* of models and easy parallelization. As future work, we would like to *dynamically* determine the partitioning of the data space. We would also like to *share* model information across region boundaries, to achieve continuity in the decision boundaries and increase robustness.

8. REFERENCES

[1] D. W. Aha. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[2] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.

[3] C. Burges, K. Svore, P. Bennett, A. Pastusiak, and Q. Wu. Learning to Rank using an Ensemble of Lambda-Gradient Models. *JMLR*, 14:25–35, 2011.

[4] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[5] S. Dasgupta and J. Langford. Tutorial summary: Active learning. In *ICML*, page 178, 2009.

[6] Y. Freund and R. Schapire. A Short Introduction to Boosting. In *IJCAI*, pages 1401–1406, 1999.

[7] T. Joachims. Training linear SVMs in linear time. In *KDD*, pages 217–226. ACM, 2006.

[8] L. Ladicky and P. Torr. Locally Linear Support Vector Machines. In *ICML*, pages 985–992, 2011.

[9] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

[10] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *JMLR*, 6:1453–1484, 2005.

[11] A. Zakai and Y. Ritov. Consistency and Localizability. *JMLR*, 10:827–856, Apr. 2009.

[8] This is possible as LocCL-Kernel is computationally less intensive than Kernel-SVMs.

[9] There are not enough MSTKs, hence the final training data size is 4400/490/365 for MNIST/USPS/OptDig.