Learning from Mistakes: Towards a Correctable Learning Algorithm

Karthik Raman
karthik@cs.cornell.edu
Cornell University
Ithaca, NY 14850 USA

Krysta M. Svore,
Ran Gilad-Bachrach,
Christopher J.C. Burges
{ksvore,rang,cburges}@microsoft.com
Microsoft Research
1 Microsoft Way, Redmond,
WA, 98052, USA

ABSTRACT

Many learning algorithms generate highly 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 during learning about which data examples are incorrectly learned. We define a set of metrics which measure the *correctability* and performance 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. Our experiments over multiple regression, classification and ranking datasets show that our correctable learning algorithm offers significant improvements over state-of-the-art techniques. Our method can be extended to correctable ranking algorithms, in particular for Web search.

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 have become increasingly common across a variety of tasks including Web Search, Advertising, Social Networking, Collaborative Filtering, and Medical systems. For example, ensemble methods such as boosted trees [12] and random forests [6] are widely used for these tasks due to their competitive performance, see for example [19, 7]. Despite their success, these algorithms have several drawbacks:

- Training and testing is slow,
- The learned models are difficult to interpret,
- The models are not well-suited for parallelization,
- It is difficult to incorporate feedback without retraining.

Consequently, debugging and correcting these models can be challenging, and existing algorithms are not well-suited to these tasks.

A particularly poignant example of the need for correctability stems from Web Search. Commercial search engines are trained over millions of example queries and corresponding result URLs. The training and debugging process can be time-consuming and tedious. 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 incorporate the fix for that particular query into the training of the ranking model. Namely, feedback on a single query has very little effect when considered among a pool of millions of training examples, so 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, in a complex model, determining why the query failed is almost impossible. The models are extremely complicated, non-linear, and non-intuitive to a human assigned the task of debugging a failing query. How can we learn a model that is easier to correct, and what are the desired properties of such a model?

In response to these challenges, we formalize a new learning paradigm called *correctable learning*. Our focus is three-fold: (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. In addition, given an example on which the current model makes a mistake, denoted by **MSTK**, we aim to fix the MSTK without hurting performance on other examples.

Our approach is based on *localizing* the effect of a training point, such that a training point will only affect the performance of its local model and points within that neighborhood. Although previous research has studied localized learning [22], it has not addressed the issue of correctability. Our method first partitions the data-space into different *regions*, and then learns separate *local models* for each region. The key insight is that there are relatively few points that are "close" to the decision boundary defined by any local model, as compared to a single *global* model, so that changing the boundary of a local model is likely to have less impact on other points than changing the boundary of a global model. Leveraging this observation, we achieve correctability by incrementally adding MSTK points to our training data as they are available. We also explore adding neighbors of the MSTK points, which can yield additional performance improvements and robustness. We show theoretically why a non-uniform sample of training points can be beneficial and lead to improved performance.

We perform experiments on multiple classification, regression and ranking datasets, to show the ability of our algorithm to correct and learn from mistakes. Our method is robust and comprehensively outperforms the *correctability* of existing methods. Furthermore, the method achieves gains in performance while using less training data than existing methods, resulting in large savings in labeling costs. We also study the specific case of using linear models in our correctable learning framework, in part because the models produced are far easier to interpret than models produced by existing ensemble/non-linear methods, and in part due to their fast training and testing speeds. Finally, we show that a linear, correctable model can nearly achieve the same performance as far more complicated non-linear models, such as RBF-Kernel SVMs or Boosted Trees, while being significantly faster to train.

2. RELATED WORK

Our work relates to several areas of machine learning.

Active Learning: Unlike *passive* supervised learning methods, active learning techniques identify which data points to label and then proceed by learning on those points. The goal of active learning is to not only learn a good model, but also to use as few labeled examples as possible to do so. A common technique used to determine the next point to be labeled is to choose the point with the most uncertainty in the label [11]. While correctable learning also requires learning incrementally as points are added, there is no control over which point is received next. The feedback points cannot be chosen in advance; they are simply provided when feedback is received.

Online Learning: Another related problem is online learning [17, 8], where examples are revealed one at a time. Here too the goal is to learn incrementally and as efficiently as possible, which is typically achieved with simple updates to the learned model. However, one key difference between (stochastic) online and correctable learning is that online learning methods typically assume that the points in the sequence are drawn from the same underlying distribution, while correctable methods cannot make this assumption since the sequence of *mistake points* need not match the underlying distribution. A second key difference is that most online learning methods do not store previously seen data and are prone to making mistakes on previously seen points after a few iterations of learning, while not necessarily predicting the correct label for the newer points. In the correctable learning framework, we neither assume an underlying distribution, nor allow degradation of performance on previously seen points and target generalization (unlike adversarial online learning), thus making the problem of correctable learning far more challenging than online learning as well.

Localized Learning: Our approach is also motivated by localized learning, a broad area that refers to methods that explicitly consider the local neighborhood of a given point when predicting its label. This idea has been successfully applied in several forms in conjunction with various underlying learning techniques [23, 20]. While many approaches over the years [10] have used this principle, recent work on localizing linear methods such as SVMs [9, 16] have achieved performance comparable to more complex methods while retaining the simplicity and efficiency of linear methods. Furthermore, [22] have shown that consistent classifiers must be localized. *Lazy Learning* [1, 13] is an extreme form of localized learning, where there is no *a priori* training. However at test time, a classification rule is quickly determined based on the region around the test point. In our approach, we will use ideas similar to lazy learning, since learning methods that use a single, monolithic model tend to be difficult to correct, as we show.

We emphasize here that our main focus is to introduce and define the problem of correctable learning, which to the best of our knowledge has not been studied before, and propose ways to measure correctability of a learned model. Our proposed algorithm belongs to the family of local learning algorithms; we introduce it here in part to study its correctability properties and demonstrate the effectiveness of our correctability measures.

3. CORRECTABLE LEARNING

Today's large-scale learning systems use complex learning algorithms and vast amounts of training data to maximize performance. However, the models learned typically use a complicated mapping from numerous parameters and features to outputs. Hence, aside from retraining an entirely new model, it is almost impossible to modify the learned models in an intuitive, straight-forward manner; this makes it very difficult to leverage human knowledge, such as that of a domain expert, in the learning of such systems and limits their improvement.

T	Held-out Test set
$t : \{0 \leq t \leq n\}$	Time
$K = \{k_1, .., k_n\}$	Random Stream of MSTKs
$k_i = (x_i, y_i^*)$	i^{th} Mistake, *i.e* at time i.
x_i	Feature representation of k_i
y_i^*	True Label of k_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.* $\sum_{(x,y^*)\in T} \Delta(M(x), y^*)$

Table 1: Notation Used

This is particularly problematic when we would like to correct mistakes that the system makes upon deployment or testing. For example, consider the following 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 of poor quality. 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**. To the best of our knowledge, this problem has not been previously identified or studied in the literature. 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.

More formally, we define a mistake point and thus correctable learning as follows.

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 $\tau = 1$, for regression we use the L_2 loss function, while for ranking we use $1 - NDCG@3$. Our notation is detailed in Table 1. Def 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[1]. 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 LEARNING ALGORITHM

[1]It may not be possible to satisfy all of the conditions for every MSTK point. However, a good correctable learning algorithm should do so for most MSTK points.

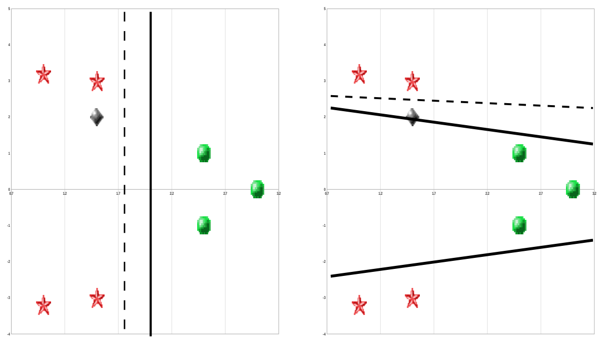


Figure 1: Left: Classification boundaries using one classifier. Right: Two separate classifiers: one above the x-axis, the other below. The gray diamond is a MSTK. Solid lines represent old boundaries; dashed lines are the new ones.

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, initial response is to use an existing algorithm and add MSTK points to the training data as they are received in an online-learning fashion. However, this approach may not satisfy the first and second conditions of correctability, *i.e.,* it may not be able to correct the MSTK itself and may adversely impact the performance of other points.

Consider the left pane of Fig. 1. The solid line represents the decision boundary of a linear classifier for the two-class classification problem (red stars and green circles). Now consider receiving feedback: let the gray diamond be a MSTK point that is improperly classified as a red star. We add the point to the training data, yet even upon retraining (the new classification boundary is denoted by the dashed line) the corresponding change in the decision boundary is unable to correct the MSTK.

This is a common problem for existing methods, due to the fact that the learned model is designed for use on *all* data-points. Hence adding a MSTK point may change the model, but may (1) not be enough to correct the point (since there are a lot of other points influencing the model as well), or (2) cause performance on other points to drop (if there are many points close to the decision boundary, then changing the boundary can impact performance). Furthermore, the distribution of the MSTK points may not match that of the underlying data, and thus training on this data will likely worsen the performance of the learned model.

Now imagine if in the above example we had instead used two separate classifiers, one each for above and below the x-axis (as shown in the right of Fig. 1). The initial boundaries are denoted by the solid lines. Adding the MSTK point (gray diamond) to the training set for the top classifier adjusts the boundary and corrects the mistake, as indicated by the updated decision boundary (dashed line). This leads to the following key observation: Using disjoint models for different regions of the data space can overcome the two aforementioned problems, which affect current methods. Having different models for each region reduces the number of points affecting each decision boundary (thus allowing for correction of a point), while also reducing the number of points close to any boundary (hence reducing the risk of worsening the performance of other points near the boundary).

Algorithm 1 Correctable Learning Framework.

```
Notation: $D$: Training Data,
$N$: No. Regions,
$L$ :                                Learning Method,
$F : D \rightarrow \{1, .., N\}$ :   Region Function,
$\{M_1, ..., M_N\}$ :                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: $M_{F(x)} = \text{Train}_L(D_{F(x)})$.
```

In response, we propose a localized-learning-based algorithm **LocCL**, detailed in Algorithm 1. First, we precompute a *good region function*, which partitions the data-space into N regions[2]. 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[3]. 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 of the region in which it belongs and update that local model using the modified training set (as done in online learning).

If we have some unlabeled data, we can extend the *correction* step of our algorithm, by adding points similar[4] to the MSTK point along with the MSTK. In other words, we first find q unlabeled neighbors of the MSTK, which belong to the same region as the MSTK (where q is a parameter). After requesting labels for them, we add them with the MSTK to the training set (as in active learning). Adding these neighboring points has two advantages:

1. *Correction:* Adding more points similar to k effectively increases the weight of x in the training set, enabling correction of x.

2. *Robustness:* If a point was wrongly labeled as a MSTK (*noisy feedback*) or is an outlier, then adding points close to k adds robustness to the method[5].

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.

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

- **Training is faster** since each training set contains fewer elements. Thus for example 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 result in more interpretable models and achieve comparable performance to ensemble methods while improving training times. Furthermore, due to the partitioning of the data, performance in any region depends only a single model, thus allowing for a better understanding of why we perform poorly over a region, thus making it easier to fix. 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. Additionally having a good partitioning of the data can greatly help avoid overfitting. For further robustness, smoothness of models across regions could help.

4.1 Learning Theoretic Justification

After multiple iterations of receiving MSTKs as feedback and adding them to the training data, the training distribution will no longer match that of the test data. Instead it will be biased toward regions with more mistakes *i.e.,* regions of low performance will occupy a higher fraction of the training data, and similarly, regions of high performance will have contributed fewer MSTKs to the training data. We justify why such a biased sample can lead to better performance as compared to an unbiased sample of the same size, using arguments from statistical learning theory.

Consider the case with just two regions, R_1 and R_2, with equal probability mass and hence the same number of points in an unbiased sample. Suppose the initial size of the training set is $2S$ (S points from each region) and we add $S' = pS$ new training points. Let this new set be a biased sample with $s_1 = \alpha S'$ ($0 \leq \alpha \leq 1$) points from R_1 and $s_2 = (1-\alpha)S'$ points from R_2. As described earlier, we train separately on both regions to classify points belonging to a region. Since our method allows different models to be used in different regions, let the local model for R_1 belong to family H_1 and correspondingly H_2 to R_2. Further, let the *Vapnik-Chervonenkis* (VC) dimension [5] for R_1 and R_2 be d_1 and d_2 for their respective families, with $d_2 = \gamma d_1$.

A well-known fact from learning theory [2] is that the gap between the *empirical error* and *true error* for a family of models is roughly proportional to either $\sqrt{\frac{d}{s}}$ (when the

[2]In our experiments, this is computed only once at the start. However this can be very easily extended to be a dynamic partitioning which changes as more data is received or depends on the task as done in [3, 18].

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

[4]Similarity can be measured for example by Euclidean closeness of the feature vectors.

[5]While the definition of correctable learning assumed no noise, this trick allows us to counter any noise in the MSTKs.

best model from the family has non-zero true error) or $\frac{d}{s}$ (when the best model has near-zero true error), where s is the training set size and d is the VC-dimension. Assuming the former, in the above example we derive the error rate for R_1 as roughly $\propto \sqrt{\frac{d_1}{S+s_1}}$ and for R_2 as $\propto \sqrt{\frac{d_2}{S+s_2}}$.

Thus, the overall error rate is approximately:

$$E = \beta/2 * \sqrt{\frac{d_1}{S+s_1}} + \beta/2 * \sqrt{\frac{\gamma d_1}{S+s_2}} \quad (1)$$

$$= \frac{\beta\sqrt{d_1}}{2\sqrt{S}} * (\sqrt{\frac{1}{1+p\alpha}} + \sqrt{\frac{\gamma}{1+p(1-\alpha)}}), \quad (2)$$

where β is the common proportionality constant. It is not hard to see that the minimum of this function is not the unbiased sample (with α=0.5), but rather a bias towards the region with higher complexity. Our algorithm uses a similarly biased sample since more MSTK points come from the region with higher error which in this case is the one with higher complexity.

Similarly, for the case when true error is nearly zero, we obtain the optimal α as $\frac{1+p-\gamma}{p(\gamma+1)}$. For large enough p this leads to the same distribution obtained by examples proportional to the initial error rate of the regions; our algorithm performs in this manner, by eventually having a training sample distribution that is proportional to the initial error rate of the various regions. Even if the MSTKs are sampled non-uniformly, under rather weak assumptions, it can be shown that the eventual training distribution of the algorithm lies closer to the *biased* optimal distribution rather than the unbiased distribution. Therefore this bias in the training sample helps our algorithm improve the overall error rate. Additionally, due to this decrease in the error rate, the number of MSTKs will likely decrease. Further, as per the algorithm, the addition of a MSTK can affect the predictions of only a fraction of the data, thus providing stability, while not drastically changing the model complexity.

5. MEASURING CORRECTABILITY

A key contribution of our work is a set of metrics with which to measure the correctability of a learning algorithm. In this section, we describe our three new metrics for correctability.

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 metrics are used to evaluate the following three key aspects required for correctable learning:

- **Correct the MSTK:** The **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 eventually 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.

Note that the definitions for these metrics are not task dependent and are applicable across multiple learning tasks, such as ranking, binary and multi-class classification, and regression.

6. EXPERIMENTS

In this section, we employ our correctability metrics to evaluate the correctability of different learning methods versus our proposed **LocCL** algorithm. We also evaluate the performance of the algorithms across several datasets (in terms of accuracy, RMSE, or NDCG). We study correctability across three different learning tasks: binary classification, regression and ranking. We also determine the robustness of our method against different baseline learning methods and various settings of bias within the incoming stream of MSTKs.

Experimental Setting: To evaluate correctability of MSTKs in a fair and realistic setting, we divide each dataset into four parts:

- *Start-Train* (**STr**): Data for training the initial models (at $t = 0$).
- *MSTK-Pool* (**MPl**): Data from which MSTKs and its (unlabeled) neighbors are drawn.
- *Validation* (**Val**): Data for parameter validation.
- *Test* (**Tes**) : Held-out data used for evaluation.

This split is performed randomly so that each part is a random sample. For the ranking datasets though, we used the provided validation and test sets, and instead randomly split the train set into the *STr* and *MPl* sets.

We first train a base learning model on the **STr** set. Next, we iteratively draw a MSTK from the **MPl** set (at random) and call the correction routine of the algorithm. Among all possible MSTKs in the current MSTK pool, we choose one at random (unless otherwise mentioned). 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 found).

We experimented on multiple standard datasets for these tasks, whose statistics are given in Table 2. For classification, we chose multiple handwriting recognition datasets due to the natural match between classification of handwriting and correctable learning, since a human can easily recognize classification errors and provide feedback to the system. We performed binary classification on these datasets by arbitrarily splitting the true labels into positive and negative sets. For binary classification, we consider all misclassifications as possible MSTKs.

We report correctability performance using the three metrics proposed in Section 5 and also report performance on the

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
Cal	Property Price Prediction	Regression	8	370	1860	18200	2065	800
Park	Parkinson Score Prediction	Regression	26	230	590	4490	590	500
MQ07	LETOR4-Million Query 07	Ranking	46	50	339	967	336	300
MQ08	LETOR4-Million Query 08	Ranking	46	50	157	421	156	200
YLTR1	Yahoo LTR Set 1	Ranking	519	200	2994	19744	6983	320
YLTR2	Yahoo LTR Set 2	Ranking	596	50	1266	1216	3798	150

Table 2: 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
	Boost-Tree	-	-	-	96.63	-					
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
	Boost-Tree	-	-	-	94.99	-					
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
	Boost-Tree	-	-	-	96.04	-					
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
	Boost-Tree	-	-	-	92.67	-					

Table 3: Correctability Performance for different existing classification methods and their corresponding LocCL method (with that method as the base learning methods). Perf refers to the test-set accuracy. NoCL indicates performance on an unbiased dataset, (*i.e.,* a dataset created by uniform sampling, *not* by the addition of MSTKs). The *Kernel* and *Boost-Tree* Perf values refer to classification accuracy when trained just once on an unbiased sample of the data (with size equal to the budget). Bold denotes the best value, * denotes statistical significance over best baseline (only applicable for the Perf and ACR metrics.).

task (*i.e.,* Accuracy/RMSE/NDCG@3) denoted by **Perf**[6]. We choose parameters using the validation set. To partition the data-space, as required by our method, we used the K-Means++ algorithm [4] ($k = 20$[7]). In 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 across four independent runs[8].

6.1 Correctability of Existing Methods

First we study how existing learning methods perform on the task of correctable learning for the task of classification. For classification, we used linear SVMs[9] [21], two non-linear methods: Kernel SVMs and Boosted Trees [12] (representative of supervised learning) and the perceptron (representative of online learning). Since we want to learn from the MSTKs, the simplest correction step is retraining on all given training data (including the incoming MSTK points). We then computed the different correctability metrics, as shown in Table 3.

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,

[6]Accuracy, ACR and OPG are reported as percentages.
[7]$k = 20$ was chosen arbitrarily, though $k = 10$ produced similar results. For OptDig we used $k = 10$ as $k = 20$ produced many empty clusters.
[8]We do not report the variance in the numbers for brevity, but found them to be small.
[9]http: //svmlight.joachims.org/svm_perf.html

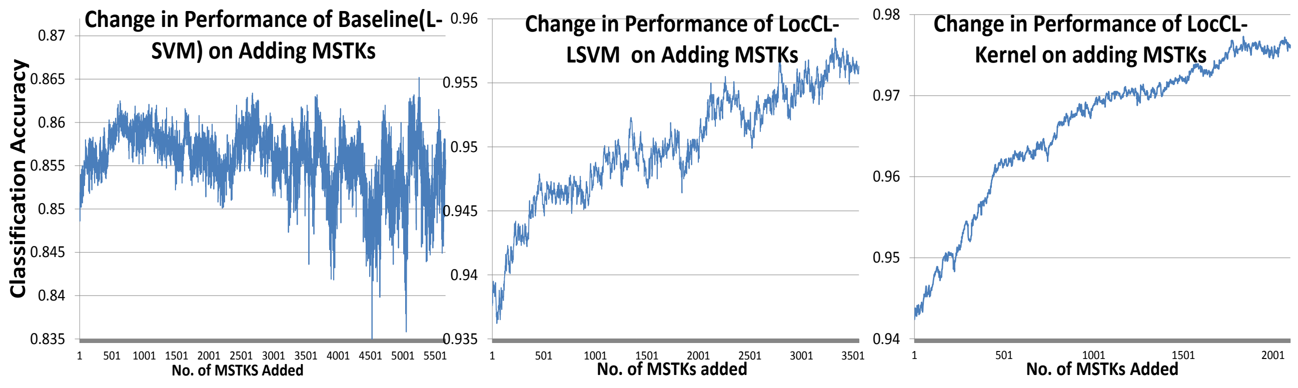


Figure 2: Change in (MNIST) test-set accuracy on adding mistakes for *a) Baseline*; *b) LocCL-LSVM* ; *c) LocCL-Kernel*

which is created due to the addition of MSTKs.

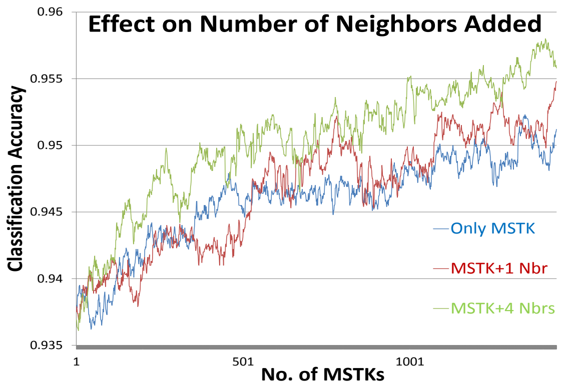
We find that the perceptron exhibits slightly better correctability, where the method is able to improve overall performance, and achieve high correction and low volatility. However, although the perceptron has high **OPG** values, it still performs poorly overall and is **significantly** worse than the Linear SVM in terms of overall performance. While it achieves a high correction rate, it does so by overcompensating for the MSTKs, and hence performs poorly on the rest of the data. This is clearly seen via the large **API** values, which indicate that the performance is volatile and tends to oscillate. This holds true even for Linear SVMs (albeit on a smaller scale) as seen in the left panel of Figure 2. This overcompensation is the primary reason why online learning methods are unsuitable for this correctable learning task.

We found the use of Kernel SVMs or ensemble methods (such as boosted trees) for correctability to be **infeasible** due to the very large computational overhead incurred by the constant retraining of models as MSTKs are added (even for the small datasets). Thus the correctability metrics are missing for them. This is the main reason 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). While there exists significant literature from the classification domain regarding different localized learning algorithms, our focus is on studying a simple **efficient** approach to solve this problem, while also being general enough to apply across other tasks such as ranking and regression. Furthermore there is no reason to believe that these methods will correct MSTKs efficiently. Hence we restrict our baselines to those mentioned above.

6.2 Correctability of LocCL

To overcome the problem of learning from a biased distribution, we use our localized correctable learning method (*LocCL*) which benefits from such a biased distribution, as justified theoretically in Section 4.1. We experimented with using different *base-learning* methods: Linear SVMs, Perceptron and Kernel SVMs for classification.

As seen in Table 3, the proposed method LocCL performs significantly better on the correction metrics as compared to the corresponding base-learning method. In particular, LocCL-LSVM outperforms the 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, we find our method is able to significantly outperform computationally more intensive methods, like Kernel SVMs and Boosted Trees. Furthermore, our method has complexity similar to Linear-SVMs[10](since the training sets for each SVM is much smaller than the original set) and 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.



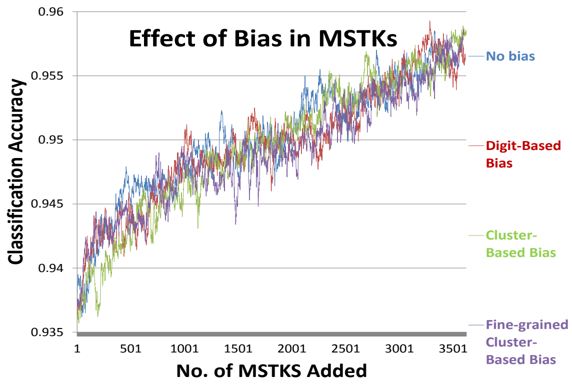


Figure 3: Effect of *a)* Neighbors; *b)* MSTK Bias

[10]In most cases we found that the LocCL-LSVM method to be significantly (order of magnitude) faster than Kernel-SVMs or boosting without parallelization.

Data	Linear Regression					LocCL				
	ACR	**API**	**OPG**	**Perf**	**NoCL**	**ACR**	**API**	**OPG**	**Perf**	**NoCL**
Cal	8.2e8	**56.2**	-3.6e3	7.4e4	7.1e4	**8.3e8**	180.3	**8.1e3**	**6.9e4***	**6.9e4**
Park	22.10	**8.6e-3**	-0.036	10.02	9.91	**149.1***	1.9e-2	**1.54**	**8.50***	9.17

Table 6: Correctability performance of LocCL for the regression datasets with *Perf* referring to RMSE error (similar to Table 3).

Bias	**ACR**	**API**	**OPG**	**Perf**
No bias	79.56	0.0274	1.87	95.63
Digit bias	79.88	0.0283	1.86	95.62
Coarse Cluster bias	79.91	0.0283	2.08	95.84
Finer Cluster bias	80.54	0.0285	2.09	95.85

Table 4: Correctability after adding bias in MSTKs

Bias	**ACR**	**API**	**OPG**	**Perf**
Only MSTK	72.85	0.0261	1.36	95.12
MSTK + 1 Nbr	69.44	0.0288	1.72	95.48
MSTK + 4 Nbr	65.83	0.0299	1.82	95.58

Table 5: Correctability after adding neighbors

Next, we ran experiments with Kernel SVMs as the base-learning method, to see if we could improve on the performance of the state-of-the-art. This is possible as LocCL-Kernel is computationally much less intensive than Kernel-SVMs. As seen in Table 3, not only do we find the correction rates to be the highest (with almost perfect correction every iteration), but we also observe the volatility to be the lowest for 3 of the 4 datasets. We observe trends similar to those in Linear-SVMs . This is further seen in Figure 2 (*Right*), where LocCL-Kernel improves on the already low volatility of the LocCL-LinearSVM method (*Middle*) and performance of nearly 98%. Finally, we note that the performance achieved by the LocCL-Kernel method is the **highest** across **all** datasets, amongst the methods studied, despite using far less training data than the others[11]. This clearly is a promising result, though we do not delve deeper as our focus here is on correctability.

6.3 Effect of Neighbors and Bias

We also study the effect that adding neighbors of MSTKs has on robustness. This is important due to potential noise in the mistakes caused by real users. To measure this, we experimented with the LocCL-LSVM method on the MNIST dataset and added varying numbers of neighbors at each correction phase. As seen in Figure 3 (Top), performance improves at a faster rate when 4 neighbors of the MSTK are labeled and added with the MSTK. This corresponds with the intuition of higher importance for the MSTK, which leads to improved performance. We also found the correction metrics do not change significantly despite adding more neighbors as seen in Table 5. In fact we find that the correction rate (ACR) actually drops on adding more neighbors. This can be explained by the fact that the added neighbors need not have the same label as the MSTK. We also observed a decreasing marginal benefit of adding more neighbors with regards to performance improvement every iteration, since the neighbors being added get progressively further from the MSTK point.

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

Next we study the sensitivity of our method to bias in the stream of MSTKs. This is particularly important as real-world users would find it hard to provide mistakes in an unbiased fashion. We ran experiments (with LocCL-LSVM on MNIST) with different biases controlling the stream of MSTKs. To introduce bias, we added a prior on *which particular* mistakes would be flagged. This prior was randomly generated in the different methods mentioned:

- Digit Based Bias: Mistakes on some digits have higher probability as compared to those on others.
- Coarse Cluster Based Bias: Mistakes from some clusters are considered more important, where the set of clusters are the same as those used in the initial partitioning.
- Fine-grained Cluster Bias: Using a clustering with $k = 50$, some (random) prior over clusters, determined the importance of different mistakes.

We see in Figure 3 (Bottom) that the method is largely agnostic to these biases, with the performance remaining steady, showing the robustness of our method. We observed minimal change in both performance and correctability metrics as seen in Table 4, across all the bias. We also found similar trends for other base learning methods and datasets. Thus these experiments indicate that the method is robust and likely to perform well under real user conditions.

6.4 Correctability for Regression

One of the key attributes of the proposed LocCL framework is that it is applicable across a wide variety of learning tasks. We therefore analyze the correctability of the proposed framework in a regression task. We evaluate the framework for regression over two different datasets from the UC-Irvine repository as shown in Table 2. In this setting, MSTKs for the current model are those examples (from the MSTK-Pool) with the highest 10% RMS error values. As a baseline we used regularized linear regression [14], where the regularization parameter is determined using the validation step at all times. This also serves as the base learning method in the LocCL framework.

As seen in Table 6, we find that the baseline performs worse at the end of the correction process, with the RMSE error increasing on both datasets. Furthermore, we also find that the performance (shown in the **Perf** column) is worse than when using an unbiased sample (shown in the **NoCL** column), mainly because it is ill-suited to learn from a skewed distribution. (Note that a lower performance metric for regression (*RMSE*) is better.)

We also observe that in this setting that the LocCL framework outperforms the baseline across most correctability

Data	SVM-Rank					LocCL				
	ACR	**API**	**OPG**	**Perf**	**NoCL**	**ACR**	**API**	**OPG**	**Perf**	**NoCL**
MQ07	1.39	0.26	0.78	39.90	**44.70**	**5.51***	0.26	**6.14**	41.90	41.90
MQ08	4.30	0.53	1.30	37.40	39.10	**7.50***	**0.28**	**1.90**	**37.51**	36.52
YLTR1	9.17	0.20	0.58	59.65	**59.75**	**10.90**	**0.05**	**1.60**	58.85	57.40
YLTR2	6.45	0.33	0.57	**62.11**	61.92	**19.30***	**0.17**	**2.50**	61.05	60.90

Table 7: Correctability performance of LocCL for the ranking datasets; Perf represents NDCG@3 (all values are multiplied by 100). Notation is the same as Table 3, with * representing statistical significance at $p = 0.1$ using a two-tailed unpaired t-test (applicable only for the ACR metric)

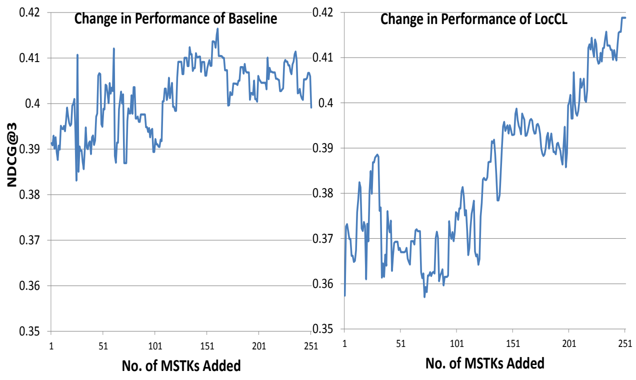


Figure 4: Change in NDCG@3 for MQ07 dataset on adding mistakes for *a) Baseline* method (SVM-Rank) ; *b) LocCL* method

metrics and the overall performance. It benefits from mistake-based learning and therefore significantly outperforms the baseline. However, we note that volatility is increased. This can be explained by the small number of training instances in the clusters, which tend to lead to larger changes in the predictions due to the volatility of the matrix inverse operation, as compared to the baseline (which has a significantly larger training set size and thus is less volatile).

It is worth noting that a key difference in the regression setting compared to the aforementioned classification setting is the semantics of the partitioning. While the clusters in the classification data could potentially map to something meaningful (*i.e.,* digits/letters), here it is far less clear what clusters correspond to. Thus the strong performance may indicate that the method can work well even if clusters do not correspond to meaningful, human-intuitive partitions.

6.5 Correctability for Ranking

We next performed experiments on the ranking datasets. We chose four large-scale, popular LETOR datasets, shown in Table 2. We used SVM-Rank[15] as a baseline and as the base learning method for our LocCL method, since it is a well-accepted baseline for these datasets. We validated the C parameter (over the range $1e-5$ to $1e3$) using the validation data.

To obtain a *partitioning*, we clustered queries using Query-only features (features which depend only on the query). While the *Million Query*(MQ) 2007 and 2008 datasets have 5 such features, the Yahoo LTR Set1 and Set 2 have 20 and 18 such features, respectively. For the MQ2007/08 datasets, we used the unnormalized data to partition the queries, but used the normalized features for SVM-rank. 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 very large, we set $k = 5$ in the clustering algorithm for all of the datasets.

In these experiments, MSTKs for the current ranker are those examples (from the MSTK-Pool) with an NDCG@3 below 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). All 4 datasets have graded relevance labels, which we use to compute the evaluation performance, measured by NDCG@3.

The results are shown in Table 7. We see that the baseline is unable to improve much via the mistake-based learning as seen from the low **OPG** scores. Comparing the Perf and NoCL values, confirms this as we find that it does not benefit from knowledge of mistakes, with performance worsening on 3 of the 4 datasets. 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.

We also note that the performance of the localized method appears to be limited by the poor clustering quality. This is clearly seen by comparing the *NoCL* scores of the baseline and the localized version. In particular we find that for the given clustering, training multiple local SVMs is far worse than training a single SVM (which does not hold true for any of the classification or regression datasets and methods). The clustering quality is likely affected by the paucity of query-only features in these datasets, as well as the relatively small number of instances (queries) used to obtain the clusters. Regardless as seen by their **Perf** values we find that LocCL method is able to leverage the MSTKs and outperforms the correctable learning baseline on both Million-Query datasets. Figure 4 clearly illustrates this difference between the two methods: While the baseline tends to oscillate it does not improve significantly as mistakes are added. On the other hand the LocCL methods continues to improve as more MSTKs are added.

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 machine learning practitioners, but that has so far not benefited from the attention of machine learning researchers. To define and evaluate characteristics of algortihms within this new paradigm, we have introduced three novel correctability metrics. We find that existing algorithms do not perform well under our correctability metrics, 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.

Given a new MSTK point, our correctable learning algorithm updates the model *only* for the region corresponding to the MSTK point by adding the point (and potentially some of its' neighbors) to the training set for that region. In addition to yielding improved correctability, our method has many benefits over existing methods, including *interpretability* of models and easy parallelization. We have also provided theoretical insight as to why our correctable method works better, despite the *bias* in the training data. Our experiments indicate that existing methods are poorly-suited for this new paradigm, while our proposed method, LocCL, exhibits statistically significant improvements in performance and correctability.

As future work, we would like to *dynamically* determine the partitioning of the data space. This would entail splitting regions that seem to have more MSTKs and low correctability scores into smaller regions. Alternatively, this could also lead to shifting the region boundaries if the model for the neighboring region is a better fit for some points. Lastly, we would also like to be able to *share* model information across region boundaries, to achieve continuity in the decision boundaries and possibly more robustness.

References

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

[2] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, 2009.

[3] C. V. Apte, R. Natarajan, E. P. D. Pednault, and F. A. Tipu. A probabilistic estimation framework for predictive modeling analytics. *IBM Syst. J.*, 41(3):438–448, 2002.

[4] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1027–1035, 2007.

[5] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learning and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, Oct. 1989.

[6] L. Breiman. Random Forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

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

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

[9] H. Cheng, P. Tan, and R. Jin. Efficient algorithm for localized support vector machine. *IEEE Trans. on Knowl. and Data Eng.*, 22:537–549, 2010.

[10] W. S. Cleveland and S. J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):pp. 596–610, 1988.

[11] S. Dasgupta and J. Langford. Tutorial summary: Active learning. In *International Conference on Machine Learning (ICML)*, page 178, 2009.

[12] Y. Freund and R. Schapire. A Short Introduction to Boosting. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1401–1406, 1999.

[13] E. K. Garcia, S. Feldman, M. R. Gupta, and S. Srivastava. Completely lazy learning. *IEEE Trans. on Knowl. and Data Eng.*, 22:1274–1285, 2010.

[14] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970.

[15] T. Joachims. Training linear SVMs in linear time. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 217–226. ACM, 2006.

[16] L. Ladicky and P. Torr. Locally Linear Support Vector Machines. In *International Conference on Machine Learning (ICML)*, pages 985–992, 2011.

[17] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2:285–318, April 1988.

[18] R. Natarajan and E. Pednault. Segmented regression estimators for massive data sets. In *SIAM International Conference on Data Mining (SDM)*, 2002.

[19] A. Schclar, A. Tsikinovsky, L. Rokach, A. Meisels, and L. Antwarg. Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *RecSys*, pages 261–264, 2009.

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

[21] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, 2005.

[22] A. Zakai and Y. Ritov. Consistency and Localizability. *Journal of Machine Learning Research (JMLR)*, 10:827–856, Apr. 2009.

[23] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2126–2136, 2006.