Active Evaluation of Classifiers on Large Datasets

Namit Katariya
IIT Bombay
namitk@cse.iitb.ac.in

Arun Iyer*
Yahoo! Research
aruniyer@cse.iitb.ac.in

Sunita Sarawagi
IIT Bombay
sunita@cse.iitb.ac.in

***Abstract*—The goal of this work is to estimate the accuracy of a classifier on a large unlabeled dataset based on a small labeled set and a human labeler. We seek to estimate accuracy and select instances for labeling in a loop via a continuously refined stratified sampling strategy. For stratifying data we develop a novel strategy of learning r bit hash functions to preserve similarity in accuracy values. We show that our algorithm provides better accuracy estimates than existing methods for learning distance preserving hash functions. Experiments on a wide spectrum of real datasets show that our estimates achieve between 15% and 62% relative reduction in error compared to existing approaches. We show how to perform stratified sampling on unlabeled data that is so large that in an interactive setting even a single sequential scan is impractical. We present an optimal algorithm for performing importance sampling on a static index over the data that achieves close to exact estimates while reading three orders of magnitude less data.**

***Keywords*-Accuracy estimation, active evaluation, learning hash functions.**

I. Introduction

In this paper we address the problem of evaluating the accuracy of a classifier $C(\mathbf{x})$ when deployed on a very large unlabeled dataset D. We are given a small labeled test set L. We consider situations where D is so large in comparison to L that the average accuracy over L is unlikely to be a reliable estimate of the classifier's real performance on the deployment data D. We present a method for more reliable accuracy estimates of $C(\mathbf{x})$ on D using the given labeled set L, and a method for selecting additional instances to be labeled to further refine the estimate.

This problem has applications in many modern systems that rely on the output of imperfect classifiers. Consider a concrete example. A search engine needs to deploy a classifier $C(\mathbf{x})$ to label if a Web page $\mathbf{x}$ is a homepage. Since the label is used to decide on the rank of a web page in a search result, it is important to calibrate reliably the accuracy of the classifier on a general web corpus D. Typically, editors hand pick a set of instances L, label them as homepage or not, and measure the accuracy of $C(\mathbf{x})$ on L. This method is likely to be flawed because the Web is so diverse and huge that it is difficult for humans to select a representative set while keeping L small. Other examples of the use of classifiers on large datasets include, classifying text snippets to an entity node in Wikipedia (1), classifying Web table columns to semantic types in an Ontology (2), and classifying the polarity of emotion in a tweet (3). All these examples share the property that a classifier is deployed on a very large dataset most of which is available in unlabeled form at the time of evaluating the classifier, and labeled data is scarce because human effort is required for labeling data.

In spite of the practical importance of the problem, existing work on the topic is surprisingly limited. The standard practice in classifier evaluation is to use a fixed labeled test set to evaluate a classifier which is then deployed for predictions on 'future' instances. Can we improve this process when the deployment set is available in unlabeled form? The use of unlabeled data for *learning* a classifier has received a lot of attention in the context of topics like active learning, semi-supervised, and transductive learning. However, the task of *learning* a classifier is very different from the task of *evaluating* a given classifier. The only existing work on selecting instances for evaluating a classifier are: (4) which presents a new proposal distribution for sampling, and (5; 6) which use stratified sampling (7). Both these methods assume that the classifier $C(\mathbf{x})$ is probabilistic and their selection is based solely on the classifier's $\Pr(y|\mathbf{x})$ scores. Our focus is more general. We wish to evaluate the accuracy of any classifier: be it a set of manually developed rules, a learned generative model with uncalibrated $\Pr(y|\mathbf{x})$ scores, or a non-probabilistic method like a decision tree — none of these can be handled by the methods in (4; 5; 6).

Our method is founded on the principles of stratified sampling like in (5; 6) but with important differences. Instead of *fixing* a stratification, we *learn* a stratification in terms of a generic feature space and the strategy evolves as more data gets labeled. For stratifying data, we use hashing instead of conventional clustering based approaches as the latter do not scale well. We design a novel algorithm for learning hash functions that cluster instances with similar accuracies more effectively than existing learning techniques for distance preserving hashing(8; 9; 10; 11; 12). Also, none of the existing estimation methods consider the case where the dataset D is so large that even a single sequential scan over it will take hours. Our method is designed to perform accuracy estimation and instance selection on D which can only be accessed via an index.

Our experiments cover an interesting range of classifica-

* Work performed as an external PhD student at IITB

tion tasks: table annotation, homepage and spam scoring of Web pages, and DNA classification; and range in size from 0.3 million to 50 million instances. We achieve upto 62% reduction in relative error over existing methods, and are able to match within 0.5% the estimates of exact methods based on full scan while sampling just 2500 instances from indexed D.

The rest of the paper is organized as follows. We present a formal statement of our problem and discuss existing approaches in Section II. In Section III we present an overview of our framework for active accuracy estimation. In Section IV we present a novel algorithm for using the labeled data to learn a stratification strategy. We empirically evaluate our method and contrast with existing techniques in Section V. In Section VI we present how we scale our algorithm to the case where the unlabeled data D is very large.

II. Problem statement and existing solutions

We present a formal description of our problem and discuss existing approaches for tackling it.

A. Problem statement

Given a classifier $C(\mathbf{x})$, a large unlabeled dataset $D = \mathbf{x}_1, \ldots, \mathbf{x}_N$, a function $a(y_i, C(\mathbf{x}_i)) \mapsto R$ that measures the accuracy of prediction $C(\mathbf{x}_i)$ when the (unknown) true label is y_i, we define the true accuracy μ of C over D as

$$\mu = \frac{1}{Z(D,C)} \sum\nolimits_{i=1}^{N} a(y_i, C(\mathbf{x}_i)) \tag{1}$$

where $Z(D,C)$ is any normalizer that can be calculated using only C and D and is independent of the unknown y_i. Many popular accuracy measures such as 0/1 accuracy, square error, and precision can be expressed in this format through appropriate choice of the $a()$ and $Z()$ functions. We use a_i for $a(y_i, C(\mathbf{x}_i))$ when the context is clear and overload N for $Z(D,C)$ because it is the most common case and it simplifies presentation. Our implementation handles the general case and our experiments are over three kinds of measures. We are given an initial labeled test set L that may or may not be representative of D. Let $n \ll N$ denote the size of L. We are allowed to augment L with additional instances selected sequentially from D and labeled by a human oracle. Our problem is to select additional instances to be labeled from D and provide an estimate $\hat{\mu}$ of the true quantity μ such that for a fixed number of additional labels, we minimize the square error $(\mu - \hat{\mu})^2$.

We next present existing approaches for solving this task.

B. Simple baseline

A baseline method for selecting instances for labeling and adding to L is sampling uniformly randomly from D. Thereafter, the accuracy of the classifier is measured on L using the standard method of averaging as:

$$\hat{\mu}_R = \frac{1}{n} \sum\nolimits_{i \in L} a_i \tag{2}$$

The square error of this estimate can be easily analyzed for the case where the entire L is chosen uniformly randomly from D. Since the choice of L is randomized, we analyze the *expectation* of the square error $(\mu - \hat{\mu}_R)^2$ over different uniform samples L of size n. We denote this quantity as $\text{Err}(\hat{\mu}_R)$. As shown in (7) and many other text books on statistical sampling, its value is $\frac{\sigma^2}{n}$ where σ^2 denotes the variance of the a_i values over D. Thus, the error of the estimate is more when the accuracy values exhibit high variance in D, and the error reduces inversely with the size of the labeled set. We next discuss two techniques for achieving smaller error for the same number of additional instances labeled.

C. Stratified sampling

A simple idea to reduce the estimation error is to stratify D into strata $D_1, \ldots, D_B$ such that within each stratum the true variance in a_i is likely to be small. Then perform stratified sampling (7) where we first select a stratum based on a policy to be discussed later in Section III and within each stratum select instances uniformly randomly. Let L_b be the set of labeled instances sampled in stratum b, n_b be the size of L_b, and $\hat{\mu}_b$ be the estimated accuracy measured over L_b using straight averaging as in Equation 2. The estimated accuracy is a weighted sum of accuracy in each stratum:

$$\hat{\mu}_S = \sum\nolimits_{b=1}^{B} p_b \hat{\mu}_b = \sum\nolimits_{b=1}^{B} p_b \frac{1}{n_b} \sum\nolimits_{i \in L_b} a_i \tag{3}$$

where p_b denotes the fraction of instances in stratum D_b. Unlike the estimator $\hat{\mu}_R$ that is calculated on the basis of only the labeled data L, the estimator $\hat{\mu}_S$ depends on both the labeled data L (for estimating $\hat{\mu}_b$) and the unlabeled data D (for p_b).

The *expected* square error $\text{Err}(\hat{\mu}_S)$ of this estimate can be shown (7) to be $\sum_b p_b^2 \frac{\sigma_b^2}{n_b}$. This means that if we can somehow achieve perfect stratification such that within each stratum all instances have the same accuracy, just one sample from each stratum is sufficient to guarantee zero error! The other extreme is where the variance of all stratum are the same — in this case stratified sampling achieves the same error as uniform sampling when $n_b \propto p_b$. In all other cases, the expected error of $\hat{\mu}_S$ is lower than $\hat{\mu}_R$ (7).

Therefore, the key to the success of this estimate is a stratification strategy that puts instances with similar accuracy values in the same stratum. Two recently proposed methods (5; 6) attempt to achieve this by assuming that $C(\mathbf{x})$ is probabilistic and its confidence score $\hat{p}(y|\mathbf{x})$ is correlated with the accuracy of prediction y on $\mathbf{x}$. They use the prediction scores to stratify the data D into equal sized bins. Thus, the goodness of this method totally hinges on the reliability of these scores.

D. Proposal distribution

Another independently developed technique for selecting instances is based on using a proposal distribution $q(\mathbf{x})$ for

sampling instances (4). The accuracy over an L sampled via $q(\mathbf{x})$ is estimated as:

$$\hat{\mu}_q = \frac{1}{\sum_{i \in L} \frac{1}{q(\mathbf{x}_i)}} \sum_{i \in L} \frac{a_i}{q(\mathbf{x}_i)} \tag{4}$$

Like in (5; 6), the classifier $C(\mathbf{x})$ is assumed to be probabilistic and capable of outputting reliable values for $\hat{p}(y|\mathbf{x})$. These scores are used to define $q(\mathbf{x})$ as:

$$q(\mathbf{x}) \propto \sqrt{\int (a(y, C(\mathbf{x})) - \mu_C)^2 \hat{p}(y|\mathbf{x}) dy} \tag{5}$$

where μ_C is $\frac{1}{N} \sum_{\mathbf{x} \in D} \int a(y, C(\mathbf{x})) \hat{p}(y|\mathbf{x}) dy$. Although the formula looks complicated, it can be easily derived by solving for the value of $q(\mathbf{x})$ that minimizes the expected square error between $\hat{\mu}_q$ and μ, and then substituting $\hat{p}(y|\mathbf{x})$ for the unknown true distribution $\Pr(y|\mathbf{x})$ and μ_C for the unknown true accuracy μ (4) . Thus, the method is crucially dependent on the goodness of these approximations.

E. Limitations of existing work

We highlight two limitations of the above two methods that we seek to remove in our work.

First, since our goal is to evaluate arbitrary classifiers, we want a method that does not so centrally depend on a single $\hat{p}(y|\mathbf{x})$ score output by the classifier. As mentioned earlier, our classifier might be a manually developed script, or a non-probabilistic method like decision trees. Even for trained probabilistic classifiers, a recent study (13) has shown that most classifier families do not provide well calibrated scores.

Second, we are aware of no existing methods for handling very large amounts of unlabeled data in the evaluation task. Even methods that depend on a few sequential scans of the whole data are not practical on such datasets. Our method can use any available index partition of the unlabeled data to avoid sequential scans as we elaborate in Section VI.

III. Our approach

The basis of our method is stratified sampling described in Section II-C. The key steps in this method are: choose a stratification strategy, use that to stratify the unlabeled data D and labeled data L, estimate accuracy $\hat{\mu}_S$ using formula 3 on the stratified data, perform stratified sampling to select more examples to label, label them, and add to L. This process is continued in a loop.

The first challenge in the above process is choosing a stratification strategy. As discussed earlier this approach is better than random sampling only if the instances within a stratum are homogeneous. Existing adaptation (5; 6) of this process for the accuracy estimation problem, have assumed a *fixed* stratification of data based on the classifier's $\hat{p}(y|\mathbf{x})$ values. In contrast, we *learn* a stratification strategy from L and our stratification strategy constantly evolves as more labeled data gets added. The second challenge is performing these steps when D is so large that stratifying the entire D is not practical. In the first part of the paper we concentrate on learning the stratification strategy and in Section VI we discuss how to manage the scalability challenge.

Algorithm 1 outlines the overall process. We next describe each of the steps in the algorithm.

Algorithm 1 Loop for active accuracy estimation

1: **Input** L, D, k, R, B
2: **repeat**
3: Learn stratification function $h(\mathbf{f}|\mathbf{w}_{1\ldots r}) \mapsto [1 \ldots B]$ if $\geq R$ instances added to L since last training.
4: Stratify L via $h(.)$ & compute $\{\hat{\mu}_b : 1 \leq b \leq B\}$
5: Stratify D via $h(.)$ & compute $\{p_b : 1 \leq b \leq B\}$
6: Display accuracy estimates: $\hat{\mu}_S = \sum_b p_b \hat{\mu}_b$.
7: Stratified sample set L' of k instances from D
8: For each $\mathbf{x}_i \in L'$, get label y_i, and add $(\mathbf{x}_i, y_i)$ to L.
9: **until** accuracy $\hat{\mu}_S$ not converged and labeler not bored.
10: **Return** $\hat{\mu}_S$

Learning stratification function: First, we use L to *learn* to stratify D into B disjoint parts such that instances within a stratum have similar accuracy values. We learn the stratification in terms of a feature vector $\mathbf{F}(\mathbf{x}, C(\mathbf{x}))$ defined over an instance $\mathbf{x}$ and the result of applying the classifier on $\mathbf{x}$. Let d denote the number of features in this feature vector. A component feature $F_j(\mathbf{x}, C(\mathbf{x}))$ $(1 \leq j \leq d)$ is a real-value representing either one of the original attributes x_j input to the classifier, or any score output by the classifier (for example, its $\hat{p}(y|\mathbf{x})$ value if available), or the predicted label itself or any other user-designed property like the fraction of missing attributes in $\mathbf{x}$ and the prior probability of the predicted label. In the rest of the paper we will use $\mathbf{f}_i$ to denote the feature vector $\mathbf{F}(\mathbf{x}_i, C(\mathbf{x}_i))$.

Once we transform each instance to a point in a d dimensional space, our problem can be recast as follows. Given N points in d dimensional space, where a small subset of size n of them are associated with an accuracy value a_i, partition the N points into B parts such that points in the same part are likely to have similar accuracy values.

A standard technique to perform such partitioning is *supervised* data clustering. The n labeled points are used to learn a distance function (14; 15) on the d features so that points with similar accuracy values have small distances in the $\mathbf{f}$ space. This learned distance function is used to cluster the N points into B parts using one of the many existing clustering algorithms. However, such clustering approaches do not scale well and in our setting, not only is N large, but as we get more labeled points, the distance function learning and clustering have to be repeated.

We instead propose to learn a *hashing function* using L such that we can independently hash each instance in D to one of B strata. This problem has recently gotten prominence (8; 9; 10; 11) in machine learning, because of the growing need to handle Web-scale data. Following the practice in this literature, we model our hash function as a

concatenation of $r = \log B$ hash functions $h_1, \ldots, h_r$ such that each $h_j(\mathbf{f}_i)$ maps an instance i to 0 or 1. A popular choice for $h_j(\mathbf{f})$ is sign($\mathbf{w}_j.\mathbf{f}$) that maps an instance based on the side[1] of the hyperplane $\mathbf{w}_j$ on which it lies in R^d. Let $\mathbf{w}_{1\ldots r}$ denote the r hyperplanes $\mathbf{w}_1, \ldots, \mathbf{w}_r$. Our proposed stratification function $h(\mathbf{f}_i|\mathbf{w}_{1\ldots r}) \mapsto [1 \ldots B]$ is then

$$h(\mathbf{f}_i|\mathbf{w}_{1\ldots r}) = \sum_{j=1}^{r} 2^{j-1} \text{sign}(\mathbf{w}_j.\mathbf{f}_i) \tag{6}$$

where $\text{sign}(x) = 0$ if $x \leq 0$ and 1 otherwise. We will often drop $\mathbf{w}_{1\ldots r}$ from h if the context makes it clear. Our task now reduces to learning the parameters $\mathbf{w}_1, \ldots, \mathbf{w}_r$ such that instances with similar accuracy values have the same h value. Section IV discusses how we solve this problem.

Estimating accuracy of a stratum: In step 4 of the algorithm we use the labeled data to assign an estimate $\hat{\mu}_b$ of the true accuracy μ_b in each stratum b. The straightforward estimate based on averaging accuracy of instances in L_b is prone to overfitting. We therefore smooth these estimates based on labeled data in neighboring buckets. Let $\text{Ham}(b, b')$ denote the Hamming distance between the binary representations of b and b'. The contribution of neighboring buckets to the smoothing constant diminishes exponentially with the Hamming distance with the neighbor. Let β denote the exponential decay constant. Let n_b^+ denotes the number of instances with $a_i = 1$ in b, Our smoothed estimate for the accuracy of b is

$$\hat{\mu}_b = \frac{n_b^+ + \gamma \sum_{d=1}^{r} \beta^d \sum_{b':\text{Ham}(b,b')=d} n_{b'}^+}{n_b + \gamma \sum_{d=1}^{r} \beta^d \sum_{b':\text{Ham}(b,b')=d} n_{b'}} \tag{7}$$

In the above equation, γ is another parameter that can be used to control the relative importance of the observed counts over the smoothing constant. We used $\gamma = 5$, and $\beta = 0.1$ for our experiments. The method reduces to Lidstone smoothing when $\beta = 1$. A second technique we use to guard against over-fitting is to disjoint partition L two-ways. Use one part for learning $h()$ and the second part for estimating μ_b within a stratum of $h()$.

Estimating weight of each stratum: Next, we need to find the fraction p_b of instances from D that lie in each stratum b. For this we need to hash partition D as per the learned function $h(\mathbf{f})$. With a single sequential pass over the data we can assign each instance in D to a bucket b and compute $p_b = \frac{1}{N} \sum_{i \in D} [\![h(\mathbf{f}_i) = b]\!]$. Later in Section VI when we present techniques for scaling our algorithm we will discuss how this step can be performed without a full scan on D.

Selecting instances for labeling: The optimal method of sampling stratified data is to first select a bucket b with probability proportional to $p_b \hat{\sigma}_b$ and then select an instance uniformly randomly within b (7). Once D has been hash partitioned via the step above, the implementation of such a sampling is easy. Later, in Section VI we show how to perform this sampling without stratifying the entire D.

The above operations are performed in a loop until the accuracy converges or until the labeling budget is exhausted.

[1]Without loss of generality we assume that the feature space includes a constant feature so that we do not need to separately model the bias.

IV. Learning Hyperplanes

Our goal is to learn the r hyperplanes $\mathbf{w}_1, \ldots, \mathbf{w}_r$ of d dimensions each, used to parameterize our stratification function h(**f**) as per Equation 6. We first characterize the optimal objective for finding $\mathbf{w}_1 \ldots \mathbf{w}_r$, then in Section IV-A describe existing algorithms for solving such objectives, and present our algorithm in Section IV-B.

The optimal $\mathbf{w}_{1\ldots r}$ is one which when used to stratify the labeled data L and unlabeled data D minimizes the expected square error of the estimated accuracy $\hat{\mu}_S$. From Section II-C we know that this error is $\sum_b p_b^2 \frac{\hat{\sigma}_b^2}{n_b}$ where p_b is the fraction of D for which $h(\mathbf{f}|\mathbf{w}_{1\ldots r}) = b$, L_b is the subset of L for which $h(\mathbf{f}|\mathbf{w}_{1\ldots r}) = b$, $\hat{\sigma}_b^2$ is the estimated variance over L_b, and n_b is the size of L_b. For simplicity we assume that L and D distribute in similar proportions over the strata, that is, $n_b \propto p_b$. This allows us to express the error in terms of the labeled data L alone as $\sum_b \frac{n_b}{n^2} \hat{\sigma}_b^2$. Further substituting for the value of variance $\hat{\sigma}_b^2$, and a little bit of manipulation we get the optimal values of $\mathbf{w}_{1\ldots r}$ as

$$\mathbf{w}_{1\ldots r}^{\text{opt}} = \text{argmin}_{\mathbf{w}_{1\ldots r}} \sum_b E(\{i \in L : h(\mathbf{f}_i|\mathbf{w}_{1\ldots r}) = b\})$$

$$\text{where } E(S) = \sum_{i,j \in S} \frac{(a_i - a_j)^2}{|S|} \tag{8}$$

This expresses the objective in terms of the sum of square differences in accuracy of pairs of instances within a stratum. The main challenge in optimizing the objective is that the sign function in $h()$ is non-differentiable in $\mathbf{w}_{1\ldots r}$. Even if we upper bound the sign function with a smooth convex function following the practice of classification algorithms, the objective remains non-convex because multiple hyperplanes have to be learned. However, by expressing the objective in this term we can relate it to the recent exciting research on learning distance preserving hash functions (8; 9; 10; 11). We will review the main techniques that have been developed to tackle such objectives in this literature.

A. Review: Hash Function Learning

The existing techniques are all based on using different smooth relaxations of the sign$(\cdot)$ function and different strategies for incrementally solving the resultant smooth, non-convex objective.

We first discuss a large margin technique proposed recently in (12). This technique uses a hinge-like loss function to penalize large Hamming distances between similar points and small Hamming distances between dissimilar points. The corresponding objective function is smoothed by a

piece-wise linear upper bound, much like in max-margin structured learning (16). The resultant smooth, non-convex objective is minimized using stochastic subgradient-descent. The technique is appealing, but when we used it for our problem of minimizing square distance between accuracies of instances within a bucket, we got poor results. The local minimas were so bad that the stochastic gradient optimizer could not progress beyond the initial step with both random and all-zero initialization of the $\mathbf{w}$ values.

We next turn to methods that attempt to learn the hash hyperplanes through optimization strategies with better guarantee of progress. In (10) the hash-function is specified in a kernel form and the parameters to be learned are the coefficients of these kernels. The objective continues to be non-smooth and non-convex. The paper proposes to solve it in a co-ordinate descent loop where each step optimally solves for a single parameter keeping all others fixed. The sequential hyperplane learning method of (11) smooths the sign function with signed magnitude. The resultant objective is solved by sequentially updating one hyperplane at a time while keeping the others fixed. Under their relaxation, a single hyperplane can be found optimally by the first Eigen vector of a transformed data matrix. The paper then proposes to re-weight misclassified pairs from previously learned hyperplanes when learning the current hyperplane.

Our method also follows the policy of updating a single hyperplane at a time as in (11), but we use a very different strategy for smoothing the objective. All existing hash function learning literature seek to approximate either a Euclidean or a black box distance measure. Our distance measure is over a scalar accuracy value, which in the common case takes 0/1 values. We exploit this common case to design a relaxation that is both more accurate and leads to a more efficient algorithm.

We next present our algorithm for learning hash functions that we call Signed Logistic Hashing.

B. Signed Logistic Hashing

We start with an initial set of hyperplanes, and update a single hyperplane at a time on a re-weighted set of instances until a local minima is reached. We first present a novel algorithm for updating a single hyperplane and then present our re-weighting strategies.

Let $[\mathbf{w}_1^t, \ldots, \mathbf{w}_r^t]$ denote the hyperplanes at time t when we are trying to improve $\mathbf{w}_j^t$. Based on the fixed value of the remaining hyperplanes, the training data L is hard partitioned into $B/2$ groups $L_1^t, \ldots, L_{B/2}^t$. Using $\mathbf{w}_j$, our goal is to bisect each group L_k^t into two parts so that the sum of the error over the new B buckets is minimized. Accordingly we can rewrite objective 8 keeping all but $\mathbf{w}_j$ fixed as:

$$\mathbf{w}_j^{\text{opt}} = \min_{\mathbf{w}} \sum_{k=1}^{B/2} \sum_{s=0,1} E(\{i \in L_k^t : \text{sign}(\mathbf{w} \cdot \mathbf{f}_i) = s\}) \quad (9)$$

For solving this objective we consider the common case where accuracy values are binary. For binary values of a_is, $E(S) = |S| f_S(1 - f_S)$ where f_S denotes the fraction of labeled instances i in S with $a_i = 1$. Even for binary values, the objective is non-differentiable and non-convex. However, for this case, we propose a tractable upper bound.

Upper bounding objective 9: Let $\mathcal{E}(\mathbf{w}, L_k^t)$ denote the part of objective 9 within the first summation over k. For ease of notation we rewrite this term as

$$\mathcal{E}(\mathbf{w}, L_k^t) = n_1 f_1 (1 - f_1) + n_0 f_0 (1 - f_0) \quad (10)$$

where n_s denotes the number of instances in L_k^t with $\text{sign}(\mathbf{w} \cdot \mathbf{f}_i) = s$ and f_s denotes the fraction of these with $a = 1$. We next present a relaxation of the objective that allows us to harness the mature area of learning binary classifiers under the loss regularization framework. Let $\text{loss}(a_i, \mathbf{w} \cdot \mathbf{f}_i)$ denote a convex upper bound to the 0/1 loss $[\![\text{sign}(\mathbf{w} \cdot \mathbf{f}_i) \neq a_i]\!]$ where $[\![z]\!]$ is 1 if z is true and 0 otherwise. Examples of such loss functions include the popular Hinge loss and Logistic loss.

Theorem 4.1

$$\mathcal{E}(\mathbf{w}, L_k) \leq \min\Big(\sum_{i \in L_k} \text{loss}(1 - a_i, \mathbf{w} \cdot \mathbf{f}_i), \sum_{i \in L_k} \text{loss}(a_i, \mathbf{w} \cdot \mathbf{f}_i)\Big)$$

Proof:

$$\begin{aligned} \mathcal{E}(\mathbf{w}, L_k) &= n_1 f_1 (1 - f_1) + n_0 f_0 (1 - f_0) \\ &\leq \min(n_1 f_1 + n_0 (1 - f_0), n_1 (1 - f_1) + n_0 f_0) \end{aligned}$$

Consider the first term in the min. The number of instances in $i \in L_k$ with $a = 1$ and $\text{sign}(\mathbf{w} \cdot \mathbf{f}_i) = 1$ is $n_1 f_1$ and with $a = 0$ and $\text{sign}(\mathbf{w} \cdot \mathbf{f}_i) = 0$ is $n_0(1 - f_0)$. Their sum is the 0/1 loss $[\![\text{sign}(\mathbf{w} \cdot \mathbf{f}_i) \neq 1 - a_i]\!]$ which in turn is less than $\text{loss}(1 - a_i, \mathbf{w} \cdot \mathbf{f}_i)$. Similarly we can prove that the second term in the min is less than $\text{loss}(a_i, \mathbf{w} \cdot \mathbf{f}_i)$. ■

The above theorem converts a non-smooth objective to a minimum of two convex objectives. Also, it converts the objective to a form that is additive over single instances rather than over pairs of instances. This provides significant benefits in terms of both running time and memory requirements over existing hash learning methods. We next elaborate on how we solve our relaxed objective.

Optimizing the upperbound: Intuitively, we are claiming that within each group our goal is to use $\mathbf{w}$ to partition the data into a positive and negative side. If we had just one group it would not matter whether we use $a = 1$ to denote the positive or negative side. But since we have multiple groups, we need to allow each group to choose the side which it wants to call positive. Our overall objective now is:

$$\min_{\mathbf{w}} \sum_{k=1}^{B/2} \min\Big(\sum_{i \in L_k} \text{loss}(1 - a_i, \mathbf{w} \cdot \mathbf{f}_i), \sum_{i \in L_k} \text{loss}(a_i, \mathbf{w} \cdot \mathbf{f}_i)\Big) \quad (11)$$

We solve the above objective using a EM-like algorithm. If for each group k, we knew z_k a function on a that is either

the identity function (Pos(a) $= a$) or an inverting function (Neg$(a) = 1 - a$), we can solve this objective

$$\min_{\mathbf{w}} \sum_{k=1}^{B/2} \sum_{i \in L_k} \text{loss}(z_k(a_i), \mathbf{w} \cdot \mathbf{f}_i) \qquad (12)$$

using standard classification techniques. So, we start with some initial guess of z_k = Pos and find a $\mathbf{w}$. For a fixed $\mathbf{w}$, we find the optimal z_k for each group that minimizes loss of that group, and continue until convergence to a local minima as shown in Algorithm 2.

Algorithm 2 Signed Logistic Hashing

Input L, r
$\mathbf{w}_1^0 \cdots \mathbf{w}_r^0$ = Initial hyperplanes
while estimated error reduces **do**
  for $j = 1$ to r **do**
    u_i = calculated weight of instance i (Section IV-B).
    z_k =Pos for all $B/2$ groups of L
    while objective improves **do**
      $\mathbf{w}_j = \text{argmin}_{\mathbf{w}} \sum_{k=1}^{B/2} \sum_{i \in L_k} u_i \text{loss}(z_k(a_i), \mathbf{w} \cdot \mathbf{f}_i)$
      $z_k = \text{argmin}_{z=\text{Pos,Neg}} \sum_{i \in L_k} \text{loss}_k(z(a_i), \mathbf{w}_j \cdot \mathbf{f}_i)$

Initial hyperplanes: The initial set of hyperplanes are chosen via a hierarchical partitioning of the data as follows. Initially, the entire data is in a single group. We find the hyperplane that partitions the group so as to minimize loss() on the group. We then repeat the following in a loop $B - 1$ times. Find the largest m groups and for each group g invoke the binary classifier to minimize loss() on g. Pick the best of these m hyperplanes where best is calculated by summing error over all current buckets in the data.

Weighting instances: We need to ensure that r hyperplanes are as different from each other as possible. We borrow ideas from boosting for re-weighting input instances to achieve this effect. When learning hyperplane $\mathbf{w}_j$ we re-weight instances in each group L_k as follows. Let minor(L_k) be the minority class in L_k and e_k denote the fraction of instances in $i \in L_k$ for which $a_i = \text{minor}(L_k)$. We interpret e_k as the error incurred on L_k from the previous set of hyperplanes that we seek to correct via $\mathbf{w}_j$. As in Adaboost, we assign weights u_i to each instance $i \in L_k$ as $u_i = \frac{1-e_k}{e_k}$ if $a_i = \text{minor}(L_k)$ else $u_i = 1$. Since $e_k \leq \frac{1}{2}$ by definition, this weighs the minority instances higher than majority. Our strategy for re-weighting is similar in spirit to re-weightings used in (11; 9) but one crucial difference is that we re-weight instances and not instance pairs.

In summary, we have developed an algorithm for hash function learning that solves for a single hyperplane at a time using an algorithm that is a small extension of the classification algorithms used in a loss-regularization framework. Although in this section we have restricted to 0/1 values for accuracy, similar reductions to regression models can be used to handle arbitrary real-valued accuracies. We defer the details to an extended version of the paper.

V. Experiments

We first present an overall comparison of our active accuracy estimation method with other methods of selecting instances and estimating accuracy discussed in Section II. We then compare with different data stratification strategies discussed in Section IV-A.

We selected the following set of five datasets covering a wide spectrum of real-life classification tasks.

TableAnnote. Our first dataset is created out of a classification task to annotate columns of noisy Web tables to one of the 250 thousand type nodes in an Ontology (17). Our dataset (D) consists of 12 million Web table columns, and a seed labeled set (L) of 541 table columns obtained from authors of (17). For our experiments in this paper, we needed true labels on all 12 million columns, which was impractical. Therefore, we setup our evaluation task as that of comparing two algorithms: one based on the state-of-art graphical model algorithm of (17) and called these *true label*, and second based on a 'classifier' that assigns a *predicted label* based on majority support. We compared these on 0/1 accuracy. The accuracy on the seed set was 56.4% whereas the accuracy of D was 16.5%. This shows the significant editorial bias in selecting the seed set. $\mathbf{F}(\mathbf{x}, C)$ consisted of 42 features including numerical properties of the column like percentage of text/number cells, size, average width, and the type node in the Ontology, and other features mentioned in (17). The majority rule classifier had no explicit confidence score. To compare with score-based methods, we trained prediction scores using a Sigmoid function with the 0/1 accuracy values as the class label and all 42 features as input.

Spam. This dataset from the LibSVM dataset collection[2] is for classifying Web pages as spam or not. The data consists of 350,000 instances and 16 million features, which we projected using random hyperplanes to 1000 following (18). This dataset included true labels. The classifier evaluated was a linear SVM trained on 5000 instances on LibSVM with default parameters, and class weights inversely proportional to the skew of the two classes in the train set. In this data, a separate seed set was not naturally available. So, we created one by simulating a committee-based active learning process. We created a committee of k SVMs. (We chose $k = 7$) by sampling a different labeled dataset for training each SVM. We then sampled a seed set of 5000 instances by weighting each instance by the disagreement in predictions by the k SVMs. Since the dataset was highly skewed, we evaluated a weighted accuracy measure on the dataset with the weights of 0.1 and 0.9 for predicted majority class and predicted minority class instances respectively. The classifier scores used were the probability scores output by LibSVM.

DNA. This dataset from the Pascal Large Scale Learning

[2]http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html#webspam

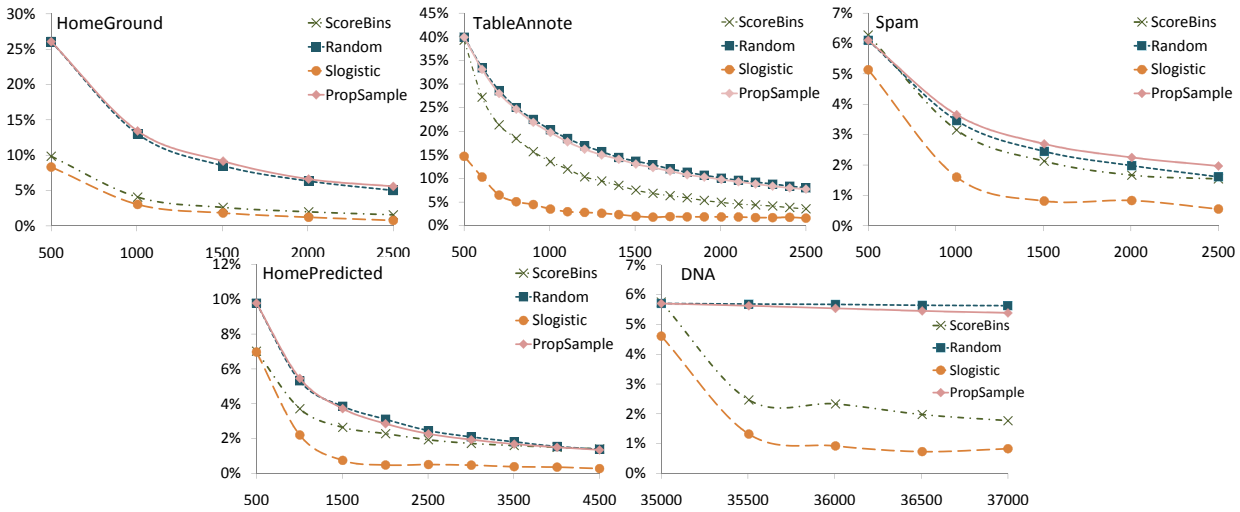

Figure 1: Absolute error (on the Y axis) of different estimation algorithms against increasing number of labeled instances (on the X axis). The five graphs correspond to the five different datasets as shown in the graph's title.

Challenge[3] is for a binary DNA classification task. The data consists of 50 million instances and 800 binary features. Other details about the classifier, the seed selection, accuracy measure, and scores used are the same as that of the Spam dataset; only the sizes of training and seed data are 100K instances each.

HomeGround and HomePredicted. The source of these datasets is Yahoo![4]. Each instance is a (entity, web page) tuple and the classifier has to decide if the web page was a homepage for the entity. $\mathbf{F}(\mathbf{x}, C)$ consisted of 66 numerical features which includes ranking features, entity match features and static web page features. The entire data consisted of 14 million instances but an editorial process selected and assigned labels only to 22 thousand of them. The company used a gradient boosted decision tree (GBDT) classifier for the task. We created two datasets out of this source: the first called **HomeGround** was restricted to the 22 thousand with editorial label as true labels and GBDT labels as predicted, the second called **HomePredicted** over all 14 million with GBDT labels as true labels and the output of a trained linear SVM as predicted labels. For this task precision is more important than accuracy, so in HomeGround only instances with predicted label true were relevant. The final dataset had 514 points in training and 1060 in test. For HomePrediction, we retained all 14 million instances for scalability experiments, therefore we measured weighted accuracy as in Spam. This measure can be treated as an approximation to precision because it gives more importance to instances with predicted label 1 (minority class).

In Table I we present a summary of the five datasets.

[3]ftp://largescale.ml.tu-berlin.de/largescale/dna/

[4]All experiments on these datasets were performed by A. Iyer at Yahoo!

Dataset	# Features	Size Seed(L)	Size Unlabeled(D)	Accuracy (%) Seed(L)	Accuracy (%) True(D)
TableAnnote	42	541	11,954,983	56.4	16.5
Spam	1000	5000	350,000	86.4	93.2
DNA	800	100,000	50,000,000	72.2	77.9
HomeGround	66	514	1060	50.4	32.8
HomePredicted	66	8658	13,951,053	83.2	93.9

Table I: Summary of Datasets

A. *Overall comparison*

We compare our overall algorithm of active evaluation in Section III (Slogistic) with existing alternatives that we discussed in Section II including,

1) Random sampling (Random)
2) Proposal sampling as in (4) (PropSample)
3) Stratified sampling with scores as in (5; 6) (ScoreBins)

All numbers are averaged over 10 random seeds and we compare different methods on the absolute difference between the estimated and true accuracy. We set the default number of strata to 16, that is, $r = 4$.

In Figure 1 we show the absolute difference in accuracy estimates with increasing number of instances labeled. We select $k = 5$ instances in one round, and restratify after 100 labels ($R = 100$). Every dot in the figure denotes the error in estimated accuracy after the retraining. These graphs help us make the following important observations:

1) The estimation error reduces significantly as we select more instances from D and add to the labeled set L. For some cases, example TableAnnote, the difference is quite substantial going from 15% to less than 1% after adding 1000 selected instances. This establishes the practical importance of the problem we are addressing.
2) The best results are obtained with Slogistic. Even the starting seed set error is lowest for Slogistic proving that the stratified estimate $\hat{\mu}_S$ that uses both L and D is more accurate than simple averaging using only L.
3) While stratified sampling based on scores (ScoreBins)

performs better than non-stratified methods like Random and PropSample, ScoreBins is much worse than Slogistic. This establishes the superiority of feature-based learned stratification over classifier score based stratification. The PropSample method is even worse affected by unreliable scores because its accuracy estimate does not involve D unlike the ScoreBins method.

4) Even when the initial accuracies are the same, as in the HomePredicted dataset, with more labeled data Slogistic performs much better than ScoreBins. This establishes the importance of evolving the stratification with increasing labeled data, instead of keeping it fixed as in the ScoreBins method.

B. Evaluating stratification methods

We next compare our Signed Logistic algorithm (Slogistic_hash) for learning hyperplanes with two state-of-art alternatives discussed in Section IV-A: the Sequential projection method of (11) (SeqProj_hash) and the Kernel-based approach of (10) (BRE_hash). For reference, we also compare with simple averaging (NoStratify) and ScoreBins.

Figure 2 plots the absolute difference between the true accuracy and estimated accuracy for different number of hash bits and labeled data sizes. The experiments in this section helped us make the following observations:

1) On all datasets and all combinations of training sizes and hash bits, hyperplanes learned via the Slogistic algorithm provide much better stratification than SeqProj_hash. This shows that our relaxation based on the logistic function that exploits the special nature of 0/1 accuracies is more effective than the signed magnitude relaxation of SeqProj_hash.
2) BRE_hash is worse than SeqProj_hash; this validates the conclusions made in (11), and establishes that the general strategy of solving for a single hyperplane at a time is better than co-ordinate ascent methods.
3) Although we do not separately plot running times here due to lack of space, we found that Slogistic was on average 2 times faster than SeqProj_hash and 3 times faster than BRE_Hash. The main reason is that the objectives in these methods are defined in terms of pairs of points unlike ours. The pairwise objective also blows up the memory requirements of these methods; both of which could not scale to our experiments on the DNA dataset with more than 40000 points.

VI. Scaling up our algorithm

In this section we describe how we scale our entire accuracy estimation process (Algorithm 1) when the unlabeled data D is very large. Since labeled data is expensive to obtain, we assume that L is small and our learning algorithm for $h(\mathbf{f})$ is not a bottleneck. Unlabeled data is accessed in two steps of Algorithm 1: First, in step 5 for calculating the fraction p_b of instances in each bucket b which in turn are used to estimate accuracy in step 6. Second, in step 7 for selecting the k instances to be labeled. An exact method for these steps requires a sequential scan over D every time $h(\mathbf{f})$ is retrained. An easy option is to replace D with a smaller uniform sample of D. We will show that this option is suboptimal compared to the methods we propose here.

We assume that the unlabeled data D is indexed so as to partition the data into disjoint parts $D_1, \ldots, D_U$ where for each partition u we can (a) get its size N_u in terms of number of instances, and (b) generate a uniform random sample of instances within the partition. All indexing strategies that we are aware of can support these capabilities easily. In Sections VI-A and VI-B we show how these capabilities are exploited to intelligently sample from D, and in Section VI-C we show empirically how effective they are in approximating the exact method based on full scan.

A. Assigning bucket weights

We discuss how to sample D so as to accurately estimate $\hat{\mu}_S$ defined in Equation 3. With a slight rewrite of that equation, we express $\hat{\mu}_S$ as

$$\hat{\mu}_S = \frac{1}{N} \sum\nolimits_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)} \tag{13}$$

Our goal is to approximate $\hat{\mu}_S$ without making a full pass of D, which can be extremely large. If we use a uniform random sample U from D to get an estimate $\hat{\mu}_{SU} = \sum_{i \in U} \hat{\mu}_{h(\mathbf{f}_i)}/m$, the expected square difference between $\hat{\mu}_{SU}$ and $\hat{\mu}_S$ can be shown to be $\sum_b p_b(\hat{\mu}_b - \hat{\mu}_S)^2/m$ where m is the size of sample U.

Is it possible to perform better than uniform random sampling given the limited ways in which we can access D? Instead of a uniform sample, suppose we get a sample Q from a proposal distribution $q(i)$ for $i \in D$ and estimate accuracy as:

$$\hat{\mu}_{Sq} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right) \tag{14}$$

Without any restriction on the way unlabeled data can be sampled, an optimal choice is $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$. With this choice, the error between $\hat{\mu}_{Sq}$ and $\hat{\mu}_S$ is zero. But, performing this sampling is impossible without first assigning each $i \in D$ to a bucket of $h(.)$, which is what we are trying to avoid in the first place. The only $q(i)$ we are allowed is the one which assigns the same probability to all instances i within a index partition D_u of D. We next provide an optimal choice of such a $q(i)$.

Theorem 6.1 When $q(i)$ is restricted so that all instances within a partition D_u are sampled with the same probability q_u, the expected squared error between $\hat{\mu}_{Sq}$ and $\hat{\mu}_S$ $E_q\left((\hat{\mu}_{Sq} - \hat{\mu}_S)^2\right)$ is minimized when

$$q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)} \tag{15}$$

where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$.

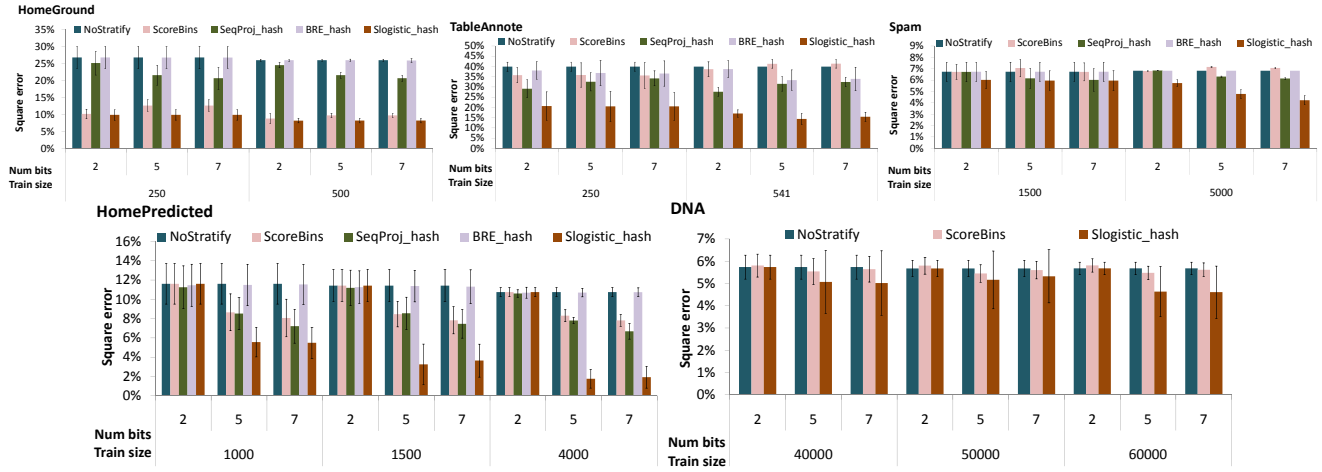

Figure 2: Error of different stratification methods against increasing training sizes and for different number of bits. The five methods compared: No-stratify, ScoreBins, SeqProj_hash, BRE_hash, Slogistic_hash are presented in this order in each group of bars. For the DNA dataset SeqProj_hash and BRE_hash could not be completed because of memory problems.

Proof:

$$
\begin{aligned}
E_q\left((\hat{\mu}_{Sq} - \hat{\mu}_S)^2\right) &= \frac{1}{m} E_q\left(\left(\frac{\hat{\mu}_{h(.)}}{Nq(.)} - \hat{\mu}_S\right)^2\right) \\
&= \frac{1}{m}\sum_i \left(\frac{\hat{\mu}_{h(\mathbf{f}_i)}}{Nq(i)} - \hat{\mu}_S\right)^2 q(i) \\
&= \frac{1}{m}\sum_u \sum_b \left(\frac{\hat{\mu}_b}{Nq_u} - \hat{\mu}_S\right)^2 p(b|u) p_u q_u
\end{aligned}
$$

In order to get a distribution we require $Nq_u p_u = 1$ where p_u is the fraction of instances in data partition D_u. The value of $p_u = \frac{N_u}{N}$ is known to us exactly from the index. From the last expression we get that $E_q\left((\hat{\mu}_{Sq} - \hat{\mu}_S)^2\right)$ is minimized by the following constrained objective:

$$
\min_{q_1,\ldots,q_U} \sum_u \sum_b \frac{\hat{\mu}_b^2}{q_u} p(b|u) p_u \quad s.t. \quad \sum_u N p_u q_u = 1 \qquad (16)
$$

This objective is convex in the q-s and can be solved in closed form to get the optimal solution as in Equation 15. ■

To calculate q_u we need an estimate of $p(b|u)$. We use the following strategy to obtain these estimates. Initially, we depend on the labeled data and a small static sample to estimate $p(b|u)$. As the algorithm progresses and more instances are sampled from any D_u, we use these to continuously refine $p(b|u)$. We show empirically that in spite of depending on an estimate of $p(b|u)$, these values of q_u do better than uniform sampling.

B. Selecting instances

In this step we need to sample k instances from D such that the probability $g(i)$ of including sample i is proportional to $\hat{\sigma}_{h(\mathbf{f}_i)}$; and we have to do this without evaluating $h(\mathbf{f}_i)$ over each $i \in D$. Our only option is to use a proposal distribution $q(i)$ that is restricted to choose the same $q(i)$ for each i in data partition D_u. Using $q(i)$ we sample a set S of size larger than k, and from S we sample the k instances by weighting each instance as $g(i)/q(i)$. The sampling is efficient if $q(i)$ is close to $g(i)$. We find the best $q(i)$ by solving for the unlabeled bucket weights $q_1, \ldots, q_U$ for which the expected L1 distance between $g(i)$ and $q(i)$ is minimized. Under the restriction imposed on $q(i)$, this goal can be formulated as the following linear program:

$$
\min_{q_1,\ldots,q_U} \sum_u \sum_b p_u p(b|u) \left| \frac{\hat{\sigma}_b}{Z_g} - q_u \right| \quad s.t. \sum_u N p_u q_u = 1 \qquad (17)
$$

where Z_g denotes the normalizer for the $g(i)$ distribution which we approximate as $Z_g = \sum_b \hat{\sigma}_b \sum_u p_u p(b|u)$. In the above, $|\frac{\hat{\sigma}_b}{Z_g} - q_u|$ is the difference between the sampling probability of an instance that belongs to D_u and has $h(\mathbf{f}) = b$. The term $p_u p(b|u)$ estimate the fraction of such instances. Thus, the objective above estimates the L1 distance between $g(i)$ and $q(i)$ under the constraint that the q_u values define a distribution. This objective can be easily solved as a linear program using any off the shelf package.

C. Empirical evaluation

Here we show that with well-designed sampling methods we can obtain accuracy estimates that are close to those obtained via the exact method based on sequential scan while reading orders of magnitude less data.

We perform these experiments on the three largest datasets: TableAnnote, HomePredicted, and DNA from Section V consisting of 12 million, 14 million, and 50 million instances respectively. We created indices on each of the data by hashing on a seven bit signature. The hyperplanes for the signature were obtained by first projecting the data onto the top-seven Eigen vectors of a sample of size 0.1 million from the data D. Then sequentially we choose a hyperplane for hashing by finding the bias and Eigen vector that achieves

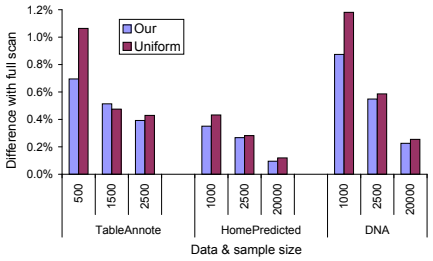

Figure 3: Comparing methods of sampling from indexed data for estimating bucket weights

the greatest reduction in variance while achieving at least a one-third/two-thirds split of the data. The data was thus divided into $2^7 = 128$ partitions. The maximum size of a partition was 5% of the data showing a fairly good balance among partitions.

We evaluate our method of sampling D for estimating accuracy (discussed in Section VI-A) by comparing against a baseline where a uniform random sample on D is used to estimate p_b. We measure error by comparing the estimated accuracy with the estimate obtained via a full scan. In Figure 3 we compare the two methods for varying budgets on the number of instances to be sampled for each dataset. We see that as the amount of data sampled is increased, both estimates get closer to exact estimates from full scan. With just 2500 sampled instances, the estimated accuracy is within 0.5% of the optimal, and with 20,000 it is within 0.2%. Our method performs much better than uniform for small sample sizes. As expected, the difference diminishes as the sample size grows.

We performed similar experiments comparing our algorithm for instance selection in Section VI-B with uniform selection. Due to space limitations we cannot show those graphs. These experiments also support the above finding that that it is possible to perform stratified sampling on indexed unlabeled datasets; and an intelligent sampling strategy provides significant gains over uniform random sampling.

VII. Conclusions

In this paper we addressed an important challenge arising in real-life deployments of classification models — calibrating a classifier's accuracy on large unlabeled datasets given only a few labeled instances and a human labeler. We proposed a method based on stratified sampling theory that provides better estimates than straight averaging and better selection of instances for labeling than random sampling. We proposed a stratification method based on hashing on r learned hyperplanes. We relate our problem to the recent exciting literature on learning distance preserving hash functions, and propose a novel formulation that leads to an efficient learning algorithm. Experiments on a wide spectrum of real datasets show that our estimates achieve between 15% and 62% relative reduction in error compared to existing approaches. We make our algorithm scalable by proposing optimal sampling strategies for accessing indexed unlabeled data directly. We show that our strategies achieve close to optimal performance while reading three orders of magnitude fewer instances on datasets of upto 50 million instances.

Future work include estimating accuracy of structured learning tasks such as sequential labeling where predicted labels within a sequence are not i.i.d, and handling accuracy measures like F1 where the denominator depends on the unknown true label.

Acknowledgment: This work was partly supported by research grants from the Indo-German Max Planck Centre for Computer Science (IMPECS) and from Yahoo! Research.

References

[1] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti, "Collective annotation of wikipedia entities in web text," in *SIGKDD*, 2009.

[2] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu, "Recovering semantics of tables on the web," *PVLDB*, vol. 4, no. 9, 2011.

[3] D. Davidov, O. Tsur, and A. Rappoport, "Enhanced sentiment learning using twitter hashtags and smileys," in *COLING (Posters)*, 2010, pp. 241–249.

[4] C. Sawade, N. Landwehr, S. Bickel, and T. Scheffer, "Active risk estimation," in *ICML*, 2010.

[5] P. N. Bennett and V. R. Carvalho, "Online stratified sampling: evaluating classifiers at web-scale," in *CIKM*, 2010.

[6] G. Druck and A. McCallum, "Toward interactive training and evaluation," in *CIKM*, 2011.

[7] W. G. Cochran, *Sampling Techniques*, 3rd ed. Wiley, 1977.

[8] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008.

[9] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu, "Complementary hashing for approximate nearest neighbor search," in *ICCV*, 2011.

[10] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *NIPS*, 2009.

[11] J. Wang, S. Kumar, and S. Chang, "Sequential projection learning for hashing with compact codes," in *ICML*, 2010.

[12] M. Norouzi and D. Fleet, "Minimal loss hashing for compact binary codes," in *ICML*, 2011.

[13] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *ICML*, 2005.

[14] K. Weinberger and L. Saul, "Fast solvers and efficient implementations for distance metric learning," in *ICML*, 2008.

[15] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon, "Information-theoretic metric learning," in *ICML*, 2007.

[16] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *JMLR*, vol. 6, pp. 1453–1484, 2005.

[17] G. Limaye, S. Sarawagi, and S. Chakrabarti, "Annotating and searching web tables using entities, types and relationships," in *VLDB*, 2010.

[18] P. Li and A. C. König, "b-bit minwise hashing," in *WWW*, 2010.