Balancing Exploration and Exploitation: A New Algorithm for Active Machine Learning

Thomas Osugi, Deng Kun, and Stephen Scott
Dept. of Computer Science
256 Avery Hall
University of Nebraska
Lincoln, NE 68588-0115
{tosugi,kdeng,sscott}@cse.unl.edu

Abstract

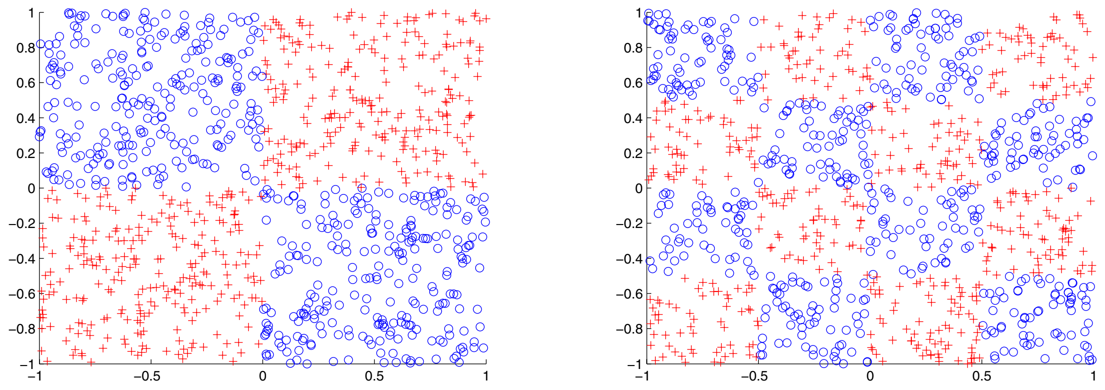
Active machine learning algorithms are used when large numbers of unlabeled examples are available and getting labels for them is costly (e.g. requiring consulting a human expert). Many conventional active learning algorithms focus on refining the decision boundary, at the expense of exploring new regions that the current hypothesis misclassifies. We propose a new active learning algorithm that balances such exploration with refining of the decision boundary by dynamically adjusting the probability to explore at each step. Our experimental results demonstrate improved performance on data sets that require extensive exploration while remaining competitive on data sets that do not. Our algorithm also shows significant tolerance of noise.

1. Introduction

In active machine learning (also known as active sampling or selective sampling), the learning algorithm A has access to a large set U of unlabeled examples and some oracle O that A can query to get the label of an individual example $x \in U$ (x is then moved from U to a set L of labeled examples). As A's queries to O are assumed to be expensive (e.g. consulting a human expert), it is only feasible to label a small subset of U. Thus the goal is for A to actively select examples from U such that a good hypothesis is learned while using as few (carefully chosen) labeled examples as possible. Applications of active learning include areas in which there is so much data available that it's infeasible to manually label it all, so one needs to manually label a small subset to train a classifier. Successful applications of active learning include include text classification [20], image classification [12], speech recognition [7, 8], and software testing [3].

Many conventional active learning algorithms choose to label points that are near the decision boundary of the current hypothesis. This can work well if the active learner is aware of all the important regions of the instance space, i.e. there are no large "pockets" of examples that the learner's hypothesis will misclassify since it hasn't seen labeled examples from them. Such active learners are good at "exploitation" (labeling examples near the boundary to refine it), but they do not conduct "exploration" (searching for large regions in the instance space that they would incorrectly classify). An example of a problem requiring exploration is the exclusive OR problem (Figure 1(a)), where the negative examples are in the upper-left and lower-right regions (respectively numbered 1 and 4) and the positives are in the other two regions (numbered 2 and 3). If all the labeled points are from regions 1–3 and none from 4, then an active learner that is not designed to explore (e.g. one that always chooses from near the decision boundary) may never discover that region 4 contains negatively-labeled examples and thus never adjust its hypothesis to accommodate it. The result is that all new negative examples that appear in region 4 would be misclassified, which could lead to high generalization error. However, one must not focus too much on exploration. Algorithms that do nothing but explore will require many calls to the labeling oracle to refine its decision boundary.

Our algorithm addresses this problem by randomly choosing between exploration and exploitation at each round, and then receives feedback on how effective the exploration phase was, measured by the change induced in the learned classifier when an exploratory example is labeled and added to the training set. Like a simple reinforcement learning algorithm, our active learner updates the probability of exploring in subsequent rounds based on the feedback it received. In this way, our algorithm is similar to Baram et al.'s [1] "COMB" algorithm, except that ours is simpler.



(a) (b)

Figure 1. (a) An example of the exclusive OR problem, aka a 2×2 "checkerboard." (b) 4×4 checkerboard data ($d = 2$ and $n = 4$).

We found that in synthetic data with many regions that require exploration, our algorithm showed a distinct advantage over others in the literature, including COMB. Further, on data when extensive exploration was unnecessary, our algorithm performed competitively with the others despite its desire to explore. Finally, we discovered that even on data when extensive exploration was unnecessary, our algorithm showed improvements over the others when the labels were corrupted by noise.

In the next section we summarize related work in active learning. Then in Section 3 we describe our algorithm. Section 4 presents experimental results on noise-free and noisy data. Finally, we conclude in Section 5.

2. Related Work

An active learner tries to select the most informative example from the unlabeled pool U. One basic idea is to select examples that effectively shrink the current version space (the set of hypotheses consistent with the training data). As discussed by Tong and Koller [20], such a heuristic would be probabilistically optimal if it could always exactly halve the version space each time A makes a query. Strong theoretical results using this idea are yielded by the Query by Committee algorithm [6, 19]. The "closest sampling" method [5, 18, 20] (sometimes called "uncertainty sampling") can be thought of as a heuristic to shrink the version space. It greedily selects points that are closest to the current decision boundary. The intuition behind this is that points that are closest to the current decision boundary are points that A is most uncertain about. By labeling these, A can have better knowledge about the correctness of its decision. Tong and Koller explain why this method often works: in a large margin classifier such as a support vector machine (SVM), the current hypothesis lies approximately "in the center" of the version space and by choosing an example (a hyperplane in version space) that is closest to it, it also cuts the version space approximately in half. Although this method is elegant and easy to implement, focusing on examples near the decision boundary prevents exploration of the feature space for regions of examples that the current hypothesis misclassifies [1].

Another approach [6, 9, 11, 17] is to select examples that are helpful in building up confidence in low future error. It is impossible to know the exact future error without knowing the target concept, but approximations make this method feasible. For example, Roy and McCallum [17] suggest to directly minimize the expected error on the dataset. One challenge to this approach is that a naïve implementation results in prohibitive computational complexity (though some heuristics are available to mitigate this). Further, a fair amount of initial labeled data is needed to establish a good approximation of the prior class distribution in order for this method to work, and if labeled examples are not chosen randomly, the class probability estimation may not be statistically valid. In line with this work, Nguyen and Smeulders [14] chose examples that have the largest contribution to the current expected error: they built the classifier based on centers of clusters and then propagated the classification decision to the other samples via a local noise model. During active learning, the clustering is adjusted using a coarse-to-fine strategy in order to balance between the advantage of large clusters and the accuracy of the data representation. Yet another approach [13] is specific to SVM learning. Conceptually, in SVM learning if we can find all the true support vectors and label all of them, we will guarantee low future error. Mitra et al. assigned a confidence factor c to examples within the current decision boundary and $1 - c$ to examples outside each indicating the confidence of whether they are true support vectors, and then chose those examples probabilistically according to this confidence. It is noteworthy that their approximation of c is done using a separate validation set (with known labels) and c indicates balance of number of positive and negative examples within the neighborhood of current support vectors.

The third category contains active learning algorithms that try to quickly "boost" or "stabilize" an active learner. Active learning is unstable, especially with limited labeled examples, and the hypothesis may change dramatically each round it sees a new example. One way to boost active learning algorithms is simply combining them in some way. For example, Baram et al.'s [1] algorithm COMB combines three different active learners by finding and fast-switching to the one that currently performs the best. These three algorithms are "Simple" [20] (which selects the example nearest the decision boundary), "Self-Conf" [17] (which attempts to directly minimize generalization error), and, to better explore the feature space, their own heuristic "Kernel Farthest First" (KFF). Given a set L of labeled examples, the next example x chosen by KFF to be labeled is the one farthest from L in the feature space induced by the SVM's kernel K:

$$\operatorname*{argmax}_{x \in U} \left(\min_{y \in L} \|\Phi_K(x) - \Phi_K(y)\| \right) , \qquad (1)$$

where

$$\|\Phi_K(x) - \Phi_K(y)\| = \sqrt{K(x,x) + K(y,y) - 2K(x,y)}$$

is the Euclidean distance from $\Phi_K(x)$ to $\Phi_K(y)$, which are the images of x and y in the feature space induced by the kernel K.

The COMB algorithm works by having each "expert" (active learner) in a pool score each candidate unlabeled example (higher score implies more suitable for labeling). These scores are exponentially weighted and normalized to probabilities. After low-probability instances are filtered out, the remaining probabilities are combined into a single probability vector, which is used to randomly select the next point x from U to be labeled. After x is chosen, it is labeled and added to L, which is used to train a new classifier C. Next, the unlabeled examples in U are labeled by C and a "reward utility" of x is computed based on the classification entropy maximization (CEM) score: $H\left(|C^+(U)|/|U|\right)$, where $C^+(U) \subseteq U$ is the subset of examples of U that C labels positive and $H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$ is the entropy function. Given the previous CEM score H (before x was chosen) and the new one H' (after x was chosen), the reward utility of x is

$$r(x) = \frac{e^{H'} - e^{H} + e - 1}{2e - 2} .$$

The reward utility is then used to update the weights of the active learners in the pool. While it is possible to prove guaranteed relative loss bounds on the performance of this algorithm, it is complex to implement.

Related to Baram et al.'s approach, Pandey et al. [15] proposed an algorithm based on Kalai and Vempala's [10] "follow the perturbed leader" algorithm. Pandey et al.'s algorithm chooses between Simple and randomly selecting from U. As with COMB, their algorithm updates the weights of the "experts" (active learners) based on their effectiveness. Pandey et al. measure effectiveness by the change in classification error from the current hypothesis to the new one. A drawback to this approach is that the classification error is measured on the labeled training set. Measuring error on the training set can yield a biased estimate of the error and requires a larger starting set of labeled examples.

3. Our Algorithm

We now describe our algorithm for exploratory active learning. As with Baram et al. [1], the problem we address is how to balance the exploitation of labeling examples that are near the current decision boundary and the exploration of searching for examples that are far from the already-labeled points and labeled inconsistently with how the current hypothesis would predict.

Our work addresses this problem by, at each step, randomly deciding whether to explore or exploit. If an exploratory step is taken, then our algorithm considers the "success" of the exploration to adjust its probability of exploring again. Our algorithm is similar to that of Baram et al., but is simpler to implement since its decision-making process is more straightforward and its probabilistic model is easier to update. Further, our algorithm measures the impact of a selected example on the hypothesis via a simple empirical measure rather than Baram et al.'s classification entropy maximization score.

Formally, our algorithm flips a biased coin with probability p of coming up heads. If the result is heads, then we apply the KFF algorithm to select the next example to label. If tails, then we choose the next example with Simple [20], which chooses a point near the current decision boundary. After training using this new example, we get a new hypothesis h' (let h be the hypothesis before the new example was labeled). To determine how successful an exploration step is, we look at the change induced from h to h', denoted $d(h, h') \in [-1, +1]$. If this is positive (implying significant change from h to h'), we assume the exploration was successful and we want to keep the probability p high to encourage further exploration. If $d(h,h')$ is negative, we reduce p. Formally, we update the exploration probability p to p' as follows:

$$p' = \max(\min(p\,\lambda \exp(d(h,h')), 1-\epsilon), \epsilon) \ , \qquad (2)$$

where ϵ is a parameter that upper- and lower-bounds the value of p (so there's always a chance of exploring and exploiting) and λ is a learning rate for updating p.

We now define the function $d(h,h')$ that we use[1]. Let $S = \{x_1,\ldots,x_m\} = L \cup U$ be the set of labeled and unlabeled training examples that we have. Then for each of the two real-valued hypotheses $h(\cdot), h'(\cdot)$, we define the vectors $H = (h(x_1), h(x_2), \ldots, h(x_m))$ and $H' = (h'(x_1), h'(x_2), \ldots, h'(x_m))$, i.e. vectors of the real-valued predictions of h and h' on S. Now we define

$$s_1(h,h') = \frac{\langle H, H' \rangle}{\|H\|\,\|H'\|} \ ,$$

i.e. the inner product of H and H' normalized by the product of their lengths. Thus $s_1(h,h') \in [-1,+1]$ is the cosine of the angle between H and H'. Despite the fact that mathematically, $s_1(h,h')$ could be as small as -1, in our experiments, we found that $s_1(h,h')$ was always[2] in the interval $[1/2, 1]$. Thus we rescaled and translated s_1 as follows:

$$d_1(h,h') = 3 - 4s_1(h,h') \ .$$

Now mathematically, d_1 could be as large as 7, but in practice $d_1(h,h') \in [-1,+1]$. Negative values of d_1 correspond to large values of s_1, i.e. negative values of d_1 mean that h and h' predicted similarly on S, implying that relatively little change was induced by exploration.

The time complexity of our algorithm obviously depends on the active learners used as subroutines (KFF and Simple in our case). For each round of active learning, our algorithm takes constant time to choose between KFF and Simple, then we add the time complexity of the chosen algorithm, and we add time linear in $|S|$ to update p. Specifically, our algorithm's expected time complexity in the current round is $p\,T_{KFF} + (1-p)\,T_{Simple} + |S|$, where T_{KFF} is the time to run KFF and T_{Simple} is the time to run Simple.

The time complexities of KFF and Simple are similar. Given w as the weight vector of the current hypothesis h, Simple computes the dot product $\langle w, \Phi_K(x) \rangle$ for every $x \in U$. This requires computing the kernel $K(x,v)$ for every support vector $v \in V_h$. This takes time $\Theta(|U|\,|V_h|\,T_K)$, where T_K is the time to evaluate kernel K. Similarly, computing the argmax (1) for KFF takes time $\Theta(|U|\,|L|\,T_K)$. Since $V_h \subseteq L$ and $S = L \cup U$, we have that our algorithm's time complexity is $O(|U|\,|L|\,T_k)$ per round.

In the worst case, our time complexity each round asymptotically matches that of COMB. However, an examination of the full page of COMB's pseudocode [1] reveals that it requires more steps in practice. E.g. COMB executes all of its active learning subroutines each round, whereas we choose a single one to execute. It also performs significant bookkeeping, which takes time to run and is more complex to implement.

[1]Other possible measures are proposed later.

[2]This is because h and h' come from training on very similar data sets (only differing in a single example). For $s_1(h,h')$ to be -1, h and h' would have to predict the exact opposite of each other on *every* example in S, which is very unlikely.

4. Experimental Results

We implemented our algorithm in Matlab using the Spider[3] package and SVM^{light} 4.0. We used d_1 as the measure between hypotheses and set the parameters $\epsilon = 0.01$ and $\lambda = 0.1$. We evaluated our algorithm on noise-free and noisy data.

4.1. Noise-Free Data

The data we used in our first experiment was a generalization of exclusive OR: a d-dimensional, $n \times n \times \cdots \times n$ "checkerboard" (e.g. Figure 1(b)). We randomly generated 250 points per square in the checkerboard and put them in a set T (so $|T| = 250n^d$). The results in Figure 2 are based on 50 runs of the following experiment: We randomly selected 100 points from T to serve as the test set and placed the remaining points into U, the unlabeled set. Each curve in each figure is the average of these 50 experiments. Our results ("exploration") are contrasted against Simple, KFF, random sampling (each point labeled is chosen uniformly at random from U), and COMB, which is Baram et al.'s [1] combination of Simple, Self-Conf, and KFF. In each plot, we see that our approach starts out roughly even with the others, but then surpasses them in generalization accuracy after 50–250 examples have been labeled[4]. Our conclusion is that our method starts by emulating KFF then switches to Simple once exploratory learning is no longer fruitful, i.e. after it has sufficiently sampled all n^d regions.

To determine if our algorithm performs worse on on data that does not require extensive exploration, we tested it on three data sets from the UCI repository [2]: Landsat Satellite (SAT), Waveform, and Image Segmentation. We also tested it on data extracted from the Corel CD Image Collection, with features based on color histograms, color coherence, and moment invariants. Table 1 describes these

[3]http://www.kyb.tuebingen.mpg.de/bs/people/spider/index.html

[4]Our simulations of COMB yielded slightly worse results than those of Baram et al. on similar data. A possible explanation of this is that the squares of our checkerboards are closer together than those in Baram et al.'s XOR data; thus Simple is needed more often to refine the decision boundaries.

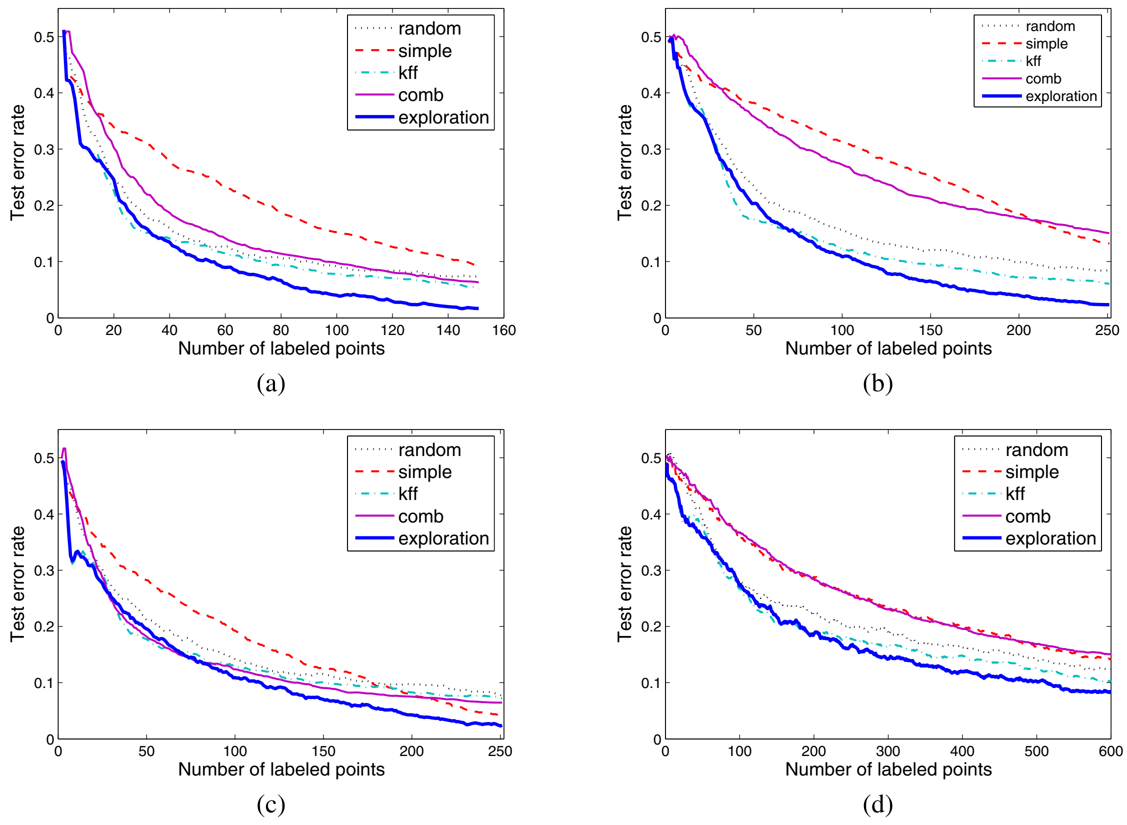


Figure 2. Checkerboard results for (a) $d = 2$ and $n = 3$, (b) $d = 2$ and $n = 4$, (c) $d = 3$ and $n = 2$, and (d) $d = 3$ and $n = 3$. "Exploration" is our algorithm, "COMB" and "kff" are from Baram et al. [1], "simple" is from Tong and Koller [20], and "random" is uniform random sampling.

four data sets and also gives the parameters γ (for the Gaussian RBF kernel) and C (for the soft-margin SVM) that we used for all active learners in each experiment. Since each of these data sets is multi-class, we ran our experiments in a one-versus-rest fashion (for brevity, only a subset of the results are shown).

Figure 3 summarizes results of 20 runs of each experiment using random, Simple, KFF, and our algorithm[5]. These along with Figure 4(a) show that our algorithm quickly learns when to stop exploring and keeps pace with Simple. Thus on these data sets, using Exploration does not hurt, even when extensive exploration is unnecessary.

4.2. Noisy Data

We also evaluated our algorithm on noisy data, using the Image Segmentation dataset from the UCI repository [2]. We performed 25 trials, where each trial started with two randomly chosen labeled examples (one positive and one negative), and each algorithm was run until it chose 150 examples to label. When an active learner chose an example to label, the label was randomly flipped with probability $\eta \in \{0, 0.05, 0.10, 0.20\}$.

In Figure 4, we see that COMB initially fares better than our algorithm for all noise rates, but only until 20–70 examples are labeled. After that point, for noise rates $\eta > 0$, our algorithm shows improved performance over the others, and the amount of improvement increases with the noise rate.

It is not obvious why our algorithm is so effective in the presence of high noise rates. One possible explanation of the advantage of our algorithm over Simple is as follows. If Simple gets a noisy label early on in learning (when $|L|$ is small), that will induce a large change in the hypothesis, potentially moving it far from the optimal hypothesis. Simple labels points near the decision boundary since it assumes that its current hypothesis is near the optimal one. When this assumption is violated due to an early noisy point, it

[5]Experiments using COMB are pending.

Table 1. UCI data set characteristics and experimental parameters.

Data Set	Number of Attributes	Number of Instances	Test Set Size	γ	C
SAT	36	6435	500	4	100
COREL	310	1000	100	2	100
Waveform	21	5000	1000	2	128
Image Segmentation	19	2310	300	1	1000

can take many small steps to reach the optimum[6]. In contrast, our algorithm starts with frequent exploration before it begins to sample near the boundary (the presence of noise will likely prolong the exploratory phase). Thus by the time our algorithm is making small steps due to closest sampling, it has a more diverse set of labeled points available, resulting in a more optimal "starting hypothesis" to refine (i.e. it starts its small steps closer to the optimal hypothesis).

Further, while COMB also chooses KFF early on, it appears that it over-commits to KFF by letting KFF's weight far exceed those of Simple and random (this is evident in Figure 4 where COMB's curves tend to follow those of KFF). Since COMB's weight updates are done multiplicatively, such an over-commitment can be difficult to reverse. In contrast, our algorithm's use of an ϵ bound on p in (2) helps to avoid this. Further experimentation (including running COMB with bounds on the ratios of the algorithms' weights) is necessary to confirm this.

4.3. Effect of Alternative Measures

Other measures of hypothesis change can be used in our algorithm. One that we have started to evaluate is specific to linear threshold units, e.g. support vector machines. Let w be the weight vector for hypothesis h and w' be the weight vector for hypothesis h'. Then we can use

$$s_2(h, h') = \frac{\langle w, w' \rangle}{\|w\| \, \|w'\|} \ , \tag{3}$$

which for a support vector machine can be computed using

$$\langle w, w' \rangle = \sum_{x_i \in V_h} \sum_{x_j \in V_{h'}} \alpha_i \, \alpha'_j \, y_i \, y_j \, K(x_i, x_j) \ ,$$

where V_h ($V_{h'}$) is the set of h's (h''s) support vectors, y_i is the $\{-1, +1\}$ label of x_i, α is the representation of SVM h, and $K(\cdot, \cdot)$ is the kernel used by the SVM. We complete the computation of (3) by noting that $\|w\| = \sqrt{\langle w, w \rangle}$. Finally, we set $d_2(h, h') = -s_2(h, h') \in [-1, +1]$.

We have completed preliminary experiments with 2×2 checkerboard, COREL, and SAT, with and without noise. So far we have not found any significant difference between d_1 and d_2, but more experiments are necessary.

[6]This is not unlike optimizing a function using gradient descent with a small step size. Eventually a near-optimal point will be reached, but convergence is slow.

5. Conclusions and Future Work

In some active learning applications, it is necessary to early on emphasize exploration of the instance space in order to familiarize the learner with the "pockets" of examples that it would otherwise misclassify. We presented a fast, simple alternative to Baram et al.'s [1] COMB algorithm. In experiments, our algorithm fared better than others in the literature on synthetic checkerboard data, and did not fare worse on data sets that did not require extensive exploration. Finally, we found that our algorithm is robust against high rates of classification noise.

Another measure of hypothesis change that we plan to evaluate is a variation of the measure d_1 from Section 3. This variation uses the symmetric difference of the subsets of S that h and h' label as positive. I.e. if P (similarly P') is the subset of S that h (h') labels positive, then we define

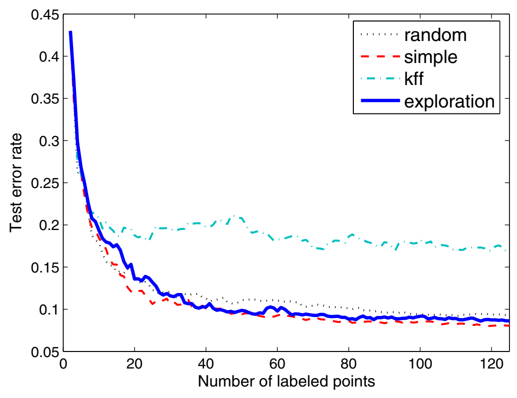
$$d_3(h, h') = \frac{2\,|P \Delta P'|}{m} - 1 \ ,$$

which has range $[-1, +1]$. This measure is similar to a special case of d_1 (if $h(\cdot)$ and $h'(\cdot)$ are $\{0, 1\}$-valued), the main difference being a change in the normalization factor.

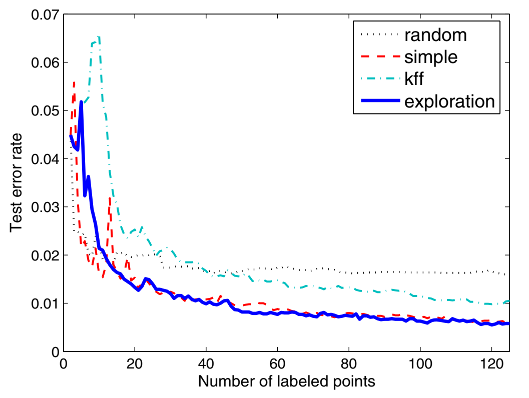
Our algorithm's update of p via (2) is sensitive to the change measured between h and h' via d. As the set of labeled examples L grows, each new labeled example added to L has less impact on h'. Thus $d(h, h')$ could be small even if the new point was a valid exploration. We did not witness this phenomenon, but it is conceivable that if L is large, then p could be reduced even if an exploratory labeling of x discovered a new region but was disregarded as noise since it was overwhelmed by the other examples in L. One way to remedy this would be to dynamically weight d by the size of L. I.e. when L is large, even small changes from h to h' should result in a positive value of $d(h, h')$.

Acknowledgments

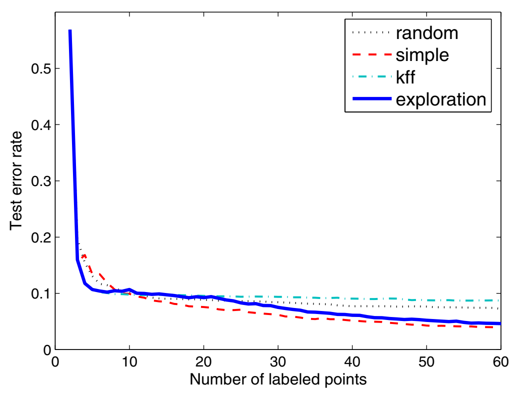
The authors used the implementation of COMB from Ran El-Yaniv's web site. This research was funded in part by



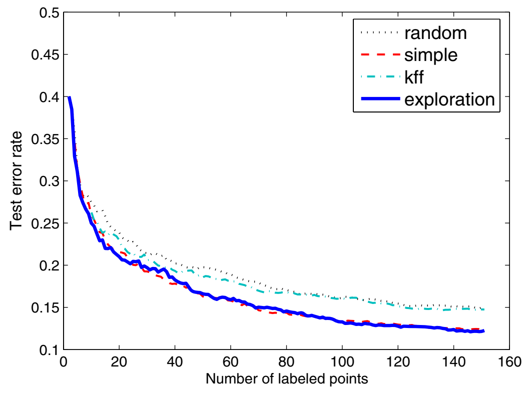
(a)



(b)



(c)



(d)

Figure 3. UCI results for (a) SAT class 7 vs. rest, (b) SAT class 2 vs. rest, (c) COREL class 3 vs. rest, and (d) Waveform class 0 vs. rest. "Exploration" is our algorithm, "kff" is from Baram et al. [1], "simple" is from Tong and Koller [20], and "random" is uniform random sampling.

NSF grant CCR-0092761 and a grant from the NU Foundation. It was also supported in part by NIH Grant Number RR-P20 RR17675 from the IDeA program of the National Center for Research Resources.

References

[1] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5(Mar):255–291, 2004.

[2] C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases, 2005. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[3] J. F. Bowring, J. M. Rehg, and M. J. Harrold. Active learning for automatic classification of software behavior. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 195–205, 2004.

[4] K. Brinker. Incorporating diversity in active learning with support vector machines. In *Proc. of the 20th International Conference on Machine Learning*, pages 59–66, 2003.

[5] C. Campbell, N. Cristianini, and A. Smola. Query learning with large margin classifiers. In *Proceedings of the 17th Intl. Conf. on Machine Learning*, pages 111–118, 2000.

[6] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–168, 1997.

[7] D. Hakkani-Tür, G. Riccardi, and A. Gorin. Active learning for automatic speech recognition. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 3904–3907, 2002.

[8] D. Hakkani-Tür, G. Tür, M. Rahim, and G. Riccardi. Unsupervised and active learning in automatic speech recognition for call classification. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 429–432, 2004.

[9] V. S. Iyengar, C. Apte, and T. Zhang. Active learning using adaptive resampling. In *Proceedings of the Sixth ACM*

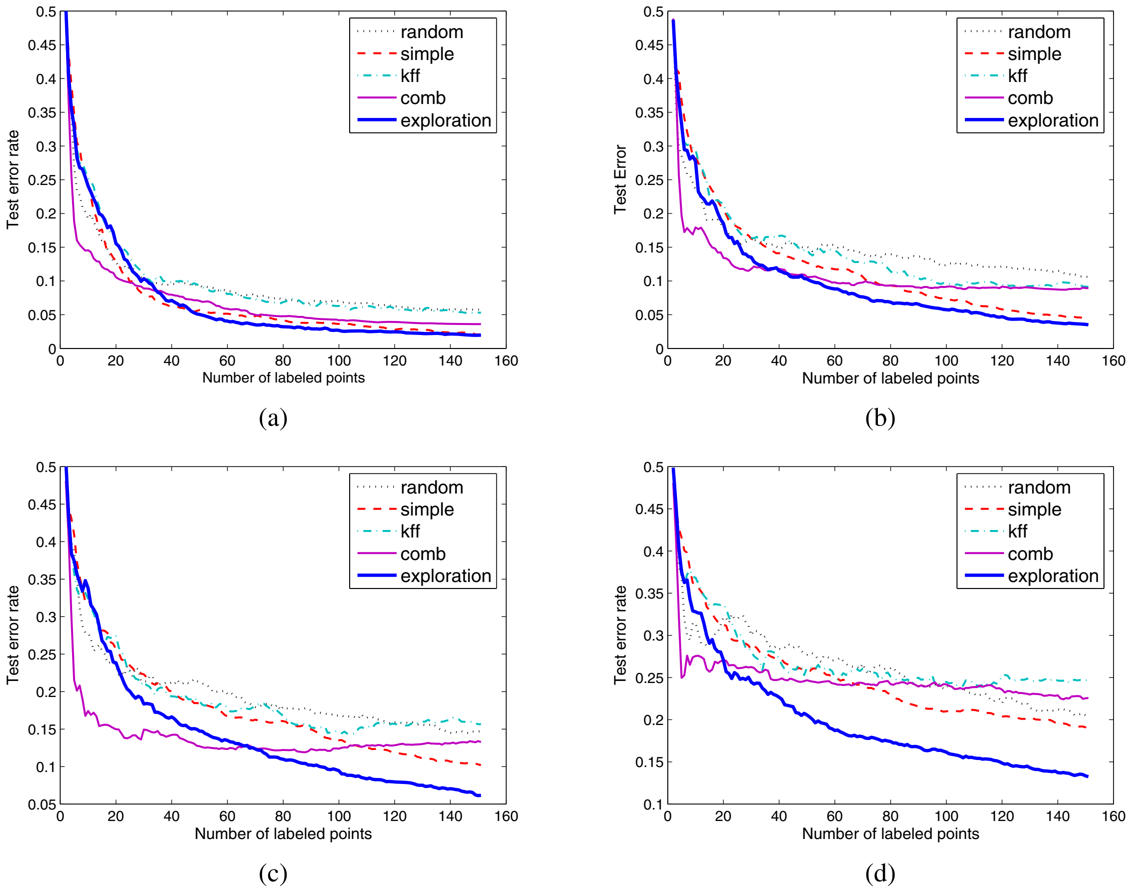


Figure 4. Image segmentation data results for (a) no noise, (b) noise rate $\eta = 0.05$, (c) noise rate $\eta = 0.10$, and (d) noise rate $\eta = 0.20$. "Exploration" is our algorithm, "COMB" and "kff" are from Baram et al. [1], "simple" is from Tong and Koller [20], and "random" is uniform random sampling.

SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 91–98, 2000.

[10] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. In *Proceedings of the 16th Annual Conference on Learning Theory*, pages 26–40, 2003.

[11] M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine Learning*, 54:125–152, 2004.

[12] T. Luo, K. Kramer, D. B. Goldgof, L. O. Hall, S. Samson, A. Remsen, and T. Hopkins. Active learning to recognize multiple types of plankton. *Journal of Machine Learning Research*, 6(Apr):589–613, 2005.

[13] P. Mitra, C. A. Murthy, and S. K. Pal. A probabilistic active support vector learning algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(3):413–418, 2004.

[14] H. T. Nguyen and A. Smeulders. Active learning using preclustering. In *Proceedings of the 21st International Conference on Machine Learning*, pages 623–630, 2004.

[15] G. Pandey, H. Gupta, and P. Mitra. Stochastic scheduling of active support vector learning algorithms. In *Proc. of the ACM Symp. on Applied Computing*, pages 38–42, 2005.

[16] J.-M. Park. Convergence and application of online active sampling using orthogonal pillar vectors. *IEEE Trans. on Patt. Anal. and Mach. Intell.*, 26(9):1197–1207, 2004.

[17] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the 18th International Conference on Machine Learning*, pages 441–448, 2001.

[18] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the 17th Intl. Conf. on Machine Learning*, pages 839–846, 2000.

[19] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 287–294. ACM Press, New York, NY, 1992.

[20] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2(Nov):45–66, 2001.