Active Class Selection

R. Lomasky[1], C.E. Brodley[1], M. Aernecke[2], D. Walt[2], and M. Friedl[3]

[1] Computer Science Department, Tufts University, Medford, MA
{rlomas01,brodley}@cs.tufts.edu
[2] Chemistry Department, Tufts University, Medford, MA
{matthew.aernecke,david.walt}@tufts.edu
[3] Dept. of Geography, Boston University, Boston, MA
friedl@bu.edu

Abstract. This paper presents *Active Class Selection* (ACS), a new class of problems for multi-class supervised learning. If one can control the classes from which training data is generated, utilizing feedback during learning to guide the generation of new training data will yield better performance than learning from any *a priori* fixed class distribution. ACS is the process of iteratively selecting class proportions for data generation. In this paper we present several methods for ACS. In an empirical evaluation, we show that for a fixed number of training instances, methods based on increasing class stability outperform methods that seek to maximize class accuracy or that use random sampling. Finally we present results of a deployed system for our motivating application: training an artificial nose to discriminate vapors.

1 Introduction

Active Class Selection (ACS) addresses the question: if one can collect n additional training instances, how should they be distributed with respect to class? We recognized this new class of supervised learning problems when working with chemists to train an artificial nose. In this domain, creating more data requires chemists to conduct experiments where vapors are passed over a sensor (the nose). Thus, the new data is labeled at the same time as it is generated. This is in contrast to a domain, such as the Reuters articles [15], in which data can be collected independent of the labeling process.

One might assume that ACS is a subclass of active learning rather than its complement [5]. Both iteratively grow the training data. However, active learning requests labels for *existing* instances [1] or explicitly queries the feature space by creating instances for an expert to label [5]. ACS requests that instances be *generated* for a particular class.

Successful methods for ACS can be grounded by recent results in stability and generalization [3,13,10], which show that one can predict expected error based on empirical error with a stable learning algorithm that satisfies certain constraints [13]. The goal of ACS is to minimize the number of new training examples needed in order to maximize learning performance. In our case, given the ability to choose class proportions for data collection we are interested in

J.N. Kok et al. (Eds.): ECML 2007, LNAI 4701, pp. 640–647, 2007.

assessing for each class whether empirical error is converging to expected error; i.e., to determine when we can do no better for this class. Uniform sampling works best if the error rates of all classes are converging at the same rate. If this is not the case, then one heuristic is to sample in proportion to the inverse of the convergence rates for each class. Using this intuition, we propose two methods for ACS that base class proportions on heuristic assessments of class stability. We compare these two methods to uniform and random sampling (sampling in proportion to the distribution specified as best by the domain expert), and to a method that uses error rate directly.

Section 2 outlines five methods for ACS. Section 3 presents a comparison of methods for ACS on both a land cover task and on the motivating problem of this research, building an artificial nose, including results of ACS deployed in the laboratory. In Section 4 we discuss the relationship of ACS to active learning and instance weighting methods. Finally, in Section 5 we conclude with a discussion of the open problems in this new research area.

2 ACS Methods

In this section, we present several methods for determining the class proportions for the generation of new training instances. We assume there are no limits on generating instances of a particular class. All of our methods begin with a small set of labeled training data T_1 of size $b[1]$, where $b[r]$ is the number of instances to add in round r. The choice of $b[1]$ and the class proportions of T_1 are domain specific issues. We perform a f-fold cross validation (CV) over T_1 (in our experiments, f=10). From the CV, we obtain class predictions for T_1. Our methods differ in how they use these predictions to specify the class proportions ($P_r[c]$, $c \in classes$) for the next round of data generation. Specifically, on round r, we generate a new set of examples, T_{new}, a set of $b[r]$ examples generated using the class proportions $P_r[c]$. We add this data to the existing data to create a new set $T_r := T_{r-1} + T_{new}$. We next describe five methods for generating $P_r[c]$ followed by a discussion of the choice of batch size and stopping criteria.

1) Uniform: Sample uniformly from all classes. $P_r[c] := \frac{1}{|classes|} * b[r]$

2) Inverse: Select class proportions $P_r[c]$ inversely proportional to their CV accuracy on round $r-1$. Thus, we obtain more instances from classes on which we have a low accuracy. This method relies on the assumption that poor class accuracy is due to not having observed sufficient training data. Although this may be true initially, our results show that this method does not perform well. $P_r[c] := \frac{\frac{1}{acc[c]}}{\sum_{i=1}^{|classes|} \frac{1}{acc[i]}} * b[r]$

3) Original Proportion: Sample in proportion to the class proportions in T_1. The idea is that domain knowledge led to these proportions, perhaps because they are the true underlying class distribution or because of the creator's intuition as to which classes are more difficult. $P_r[c] := n_c * b[r]$, where n_c is the proportion of class c found in the collected data T_r.

4) Accuracy Improvement: Sample in proportion to each classes' change in accuracy from the last round. If the change for class c is ≤ 0, then $P_r[c] = 0$. The intuition is the accuracy of classes that have been learned as well as possible will not change with the addition of new data and thus we should focus on classes that *can* be improved. This method looks for stability in the empirical error of each class. $P_r[c] := max(0, \frac{currAcc[c]-lastAcc[c]}{\sum_{i=1}^{|classes|} currAcc[i]-lastAcc[i]} * b[r])$

5) Redistricting: The idea behind redistricting is that instances from T_{r-1} whose classification changes when classified by a new classifier trained on $T_{r-1} \cup T_{new}$ are near volatile boundaries. Thus, we strive to assess which classes are near volatile boundaries in order to sample from these these "unstable" classes. The pseudocode is shown in Algorithm 1. We begin with a CV over T_1, the initial sample of the data. We obtain a prediction for each $x_i \in T_1$. In the second round, we collect T_2 of size $b[2]$. We next perform a CV over *all* of the data collected thus far and create a classifier for each fold. Note that on subsequent iterations, we keep the data from T_{r-1} in the same folds, and stratify only the newly generated data T_{new} into the existing folds. For each fold f, we compare the classification results of $C_{r,f}$ and $C_{r-1,f}$ on each instance $x_i \in T_{r-1}$. If the labels are different, then the counter for the class specified by the true label y_i, $redistricted[y_i]$ is incremented. We conclude by generating predictions of the new batch of data T_{new} and increment r.

After the second round we add instances using the formula in Step 12, where c is a class from the set of all classes in the dataset, $P_r[c]$ is the number of instances of c to add, n_c is the proportion of c in T_{r-1} and $b[r]$ is the number of new training instances. We divide $redistricted[c]$ by n_c to keep small classes from being ignored and large classes from being overemphasized.

We note the special cases here rather than in pseudocode. First, for any round r the next batch is added uniformly if instances were not redistricted in round $r-1$. Second, if instances of class c have not been redistricted, then instances of c will not be generated in the next round. Thus, resources are not wasted generating instances of a class in which the accuracy is not changing. Redistricting may temporarily blacklist c, but request c later. Empirical results show a class may be removed from the blacklist upon adding instances of neighboring classes.

Because redistricting seeks to measure stability of the class boundaries, it cares whether the prediction for instances are different than prediction in the previous iteration, not whether it is correct. Ideally, we would like new instances to be near the volatile class boundary. However, for many domains, there is no control over whether data from a particular class is near a classification boundary.

Before moving to our experimental section, we discuss the issues of batch size and stopping criteria. Batch size depends on the cost of generating instances. If too few instances are added, the method may be impractical for domains in which data is generated in large batches. If the batch size is too large, then potentially less instructive training data may be gathered. Note that a different batch size can be specified for each round. Stopping criteria depend on domain-based constraints. Data collection terminates if the accuracy is acceptable to the

Algorithm 1. Redistricting Algorithm (b)

Require: b, array of the number of instances to add in round r
Generate a sample T_1 of size $b[1]$
Divide T_1 into 10 stratified folds $T_{1,1}, T_{1,2}....T_{1,10}$
for $f = 1$ to 10 **do**
  Build Classifier $C_{1,f}$ from $\{T_1 - T_{1,f}\}$
  for all instances x_i in $T_{1,f}$ **do** $label_1[x_i] := C_{1,f}(x_i)$ **end for**
end for
$r := 2$
while instance creation resources exist and stopping criteria not met **do**
  if $r = 2$ **then**
    T_{new} := "random" sample of size $b[2]$
  else
    T_{new} := sample of size $b[r]$ where the number of instances for class c is computed as: $P_r[c] = \frac{\frac{redistricted[c]}{n_c}}{\sum_{i=1}^{|classes|} \frac{redistricted[i]}{n_c}} * b[r]$
  end if
  $T_r := T_{r-1} + T_{new}$
  Initialize $redistricted[c], \forall c \in$ classes
  Divide T_{new} into 10 stratified folds $T_{new,1}$,$T_{new,2}....T_{new,10}$
  for $f = 1$ to 10 **do**
    $T_{r,f} := T_{r-1,f} \cup T_{new,f}$
    Build Classifier $C_{r,f}$ from $\{T_r - T_{r,f}\}$
    for all instances x_i in $T_{r-1,f}$ **do**
      $label_r[x_i] := C_{r,f}(x_i)$
      if $label_r[x_i] \neq label_{r-1}[x_i]$ **then**
        $redistricted[y_i]$++ /* y_i is the true label of x_i */
      end if
    end for
    for all instances x_i in $T_{new,f}$ **do** $label_r[x_i] := C_{r,f}(x_i)$ **end for**
  end for
  r++
end while

domain expert, data generation resources are exhausted, or given the available features one is unable to wring more accuracy from the data. Investigation of stopping criteria is an open problem.

3 Experiments

Our experiments compare the proposed methods on two domains for which ACS is applicable. ACS can be applied with any supervised learning algorithm. In our experiments, we report results run with (SVMs)[1] and k-NN.

[1] We used the SMO [12] with pair-wise classification to assess accuracy [7]. Using the default parameters in Weka [21], the complexity parameter is set to one, and we use a linear kernel.

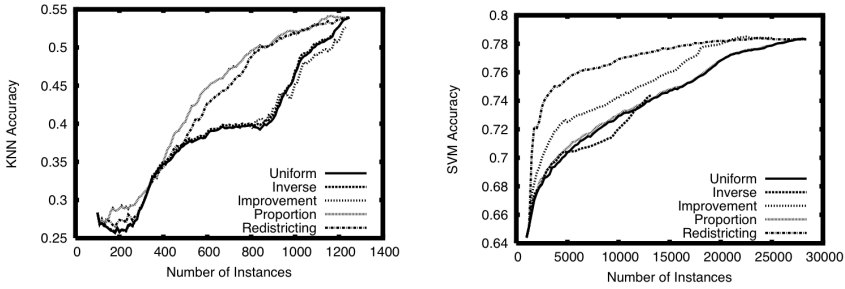
Experimental Method for Static Datasets: We simulate ACS using pre-existing static datasets. We used a uniform distribution for the test set because it is the default choice without knowledge of the class distribution of the underlying population. We performed experiments where T_1 had the same class distribution as the entire dataset, but found no significant difference in any method and thus present the results for T_1 collected uniformly. Because the data are pre-existing static datasets, we may run out of instances from a particular class. In this case, we make the large classes uniform and include all instances from the smaller classes. Similarly, when running the proposed ACS methods, we may not have sufficient data of class c on round r as dictated by the proportion $P_r[c]$. In such cases, we sample the remainder of T_{new} uniformly with respect to the classes that still contain data. In our experiments, we used a 5-fold CV to assess accuracy. This is distinct from the 10-fold CV that redistricting uses on training data.

Artificial Nose Dataset: Gathered in the Walt Laboratory at Tufts University [19], the "artificial nose" dataset consists of experiments in which vapors were passed over an array of sensors (the nose) [2]. The nose is a general purpose device that can be trained to discriminate the k vapors of interest. The accuracy is a function of how well the sensors can differentiate among the vapors. We first present results on a static dataset, and then at the end of this section, we present results from deploying redistricting in the lab.

Prior to our collaboration, the chemists on our team generated a dataset of sixteen classes[2]. The dataset is not uniformly distributed according to class because the chemists used their intuition to collect data from "needed" classes. The nose software currently uses 3-NN and we continued this practice. We repeated the experiments using SVMs, with no significant difference in the results. We started with 100 instances because the chemists believe it is sufficient to to make the initial redistricting decisions. We then add ten instances at a time. As more data is added, all methods have seen the same data, and the results converge.

Figure 1(a) shows the results for the nose. The x-axis represents the number of instances collected and the y-axis shows accuracy. We see that redistricting rivals the scientists' intuitions ("proportion" in the graph), and performs over 10% better than uniform sampling for parts of the graph. The reason that proportion performs comparably to redistricting for this data set is because it was collected before we began to work with the chemists. Prior to our collaboration, the chemists were performing ACS by manually examining the results after each new data collection to see where they were doing poorly and which classes were confused. Inverse performs poorly because as mentioned in Section 2, it skews data collection to favor classes that are "less learnable." Results for "Improvement" for this dataset are inconclusive because this method tends to focus on just a few classes at a time, improving their accuracy, and then moving on to

[2] Toluene, Dimethylmethlphosponate, ethanol, heptane, p-cymene, Isopropenyl acetate, combinations of these vapors, water, and air For some vapor classification problems, the artificial nose can achieve almost 100% accuracy [2].



(a) Nose (b) Land Cover

Fig. 1. Results show that the stability-based method, Redistricting, outperforms other methods of ACS

the next set of classes. The small number of instances in the dataset does not allow the full cycle.

Land Cover Data: The land cover dataset consists of a time series of globally distributed satellite observations of the Earth's surface [4]. ACS is applicable because geographers know where on the Earth's surface one can expect to find a given land cover type. We used an existing static dataset to illustrate the benefits of ACS for this domain. Figure 1(b) shows the results for the land cover data. The x-axis represents the number of training instances. The y-axis is the accuracy obtained by applying a set of pair-wise SVMs on the test data that vote based on the confidence assigned by the SVMs. T_1=5000 and $b[r]$=250, a reasonable number to be hand-labeled at one sitting. We exhaust the entire static dataset, causing the same ultimate accuracies for all of the methods because each method has seen the entire dataset. This is a function of the experimental method, not redistricting or ACS, nor limitations in the land cover types of the Earth. Redistricting outperforms uniform, inverse and proportional (for land cover, class proportions are dictated by the geographers' intuition, which is a highly skewed distribution). Accuracy improvement outperforms inverse, proportional and uniform, but is worse than redistricting. We conjecture this is because it looks only for *improvements* in accuracy rather than stability.

ACS in the Field: We deployed our software in the Walt Laboratory and the chemists have begun gathering data guided by redistricting. An initial dataset was gathered using the chemists' intuition of the eight vapors of interest[3]. We compute accuracy using k-NN with k=3 and 5-fold CV. Because of time limitations in the lab, we only deployed uniform and redistricting.

The CV accuracy on the initial 168 instances was 74%. On each iteration, we collected 49 additional instances. After one iteration, uniform achieves an

[3] The vapors are benzaldehyde, benzene, butyraldehyde, chloroform, decanol, ethyl propionate, and n-butanol.

accuracy of 74% and redistricting leads to an accuracy of 83%. The second iteration gives 85% and 86% accuracy, respectively. On the second iteration, both uniform sampling and redistricting obtain similar performance. We conjecture that we are close to the maximum obtainable accuracy for this set of vapors and beads. To test this conjecture we re-ran the k-NN with all of the data (325 instances in total) and obtained an accuracy of 86%, supporting our conjecture.

4 Related Work: Active Learning and Instance Selection

Little attention has been paid to determining whether more accuracy can be achieved when a practitioner can select the classes from which to generate training data. One notable exception is Jo and Japkowicz's method for collecting more examples of the minority class with the goal of increasing accuracy [9].

Active learning, like ACS, adds training examples to the dataset after examining properties of the classifier learned thus far. However, it assumes an existing set of unlabeled data or that one can query the feature space explicitly [14,11]. For active learning, requesting more data is actually requesting a label [18]. ACS does not draw from unlabeled data, rather it guides the generation of new data. Indeed, situations in which both ACS and active learning are applicable translate into situations where both the features' values and the labels exist. In this case, instance selection or weighting are applicable.

ACS is similar to methods for constructing a training set, either by instance selection [17,11], or by weighting methods such as boosting [8]. Instance selection methods focus on removing instances from a dataset that hinder classification accuracy, e.g. [17,4,20,16]. Other methods change the training set's class distribution by sampling or instance replication to handle misclassification costs or minority class issues (see [8] for an overview).

5 Conclusion

This paper identified a new class of problems called *Active Class Section*. ACS answers the question: If given the opportunity, from which classes should you generate additional training data? We evaluated several methods for ACS, each of which can be applied in conjunction with any supervised learning algorithm. We deployed redistricting as part of a real-world system in the Walt Laboratory.

Many open problems remain. First, a deeper analysis of the relationship between the proposed methods and results on stability may lead to even better methods. Second, determining when all of the available "structure" has been learned from the data, and additional learning will lead to over-fitting, is a critical objective for any learning problem. Thus, a more thorough analysis of stopping criteria is needed.

References

1. Baram, Y., El-Yaniv, R., Luz, K.: Online choice of active learning algorithms. JMLR 5, 255–291 (2004)
2. Bencic-Nagale, S., Walt, D.: Extending the longevity of fluorescence-based sensor arrays using adaptive exposure. Anal. Chem. 77(19), 6155–6162 (2005)
3. Bousquet, O., Elisseeff, A.: Stability and generalization. JMLR 2, 499–526 (2002)
4. Brodley, C., Friedl, M.: Identifying and eliminating mislabeled training instances. JAIR 11, 131–167 (1999)
5. Cohn, D.A., Ghahramani, Z., Jordan, M.l.: Active learning with statistical models. In: Advances in NIPS vol. 7, pp. 705–712 (1995)
6. Freund, Y., Schapire, R.: Experiments with a new boosting algorithm. In: ICML, pp. 148–156 (1996)
7. Hastie, T., Tibshirani, R.: Classification by pairwise coupling. In: NIPS, pp. 507–513 (1998)
8. Japkowicz, N., Stephen, S.: The class imbalance problem: A systematic study. Intel. Data Anal. 6(5), 429–449 (2002)
9. Jo, T., Japkowicz, N.: Class imbalances versus small disjuncts. In: KDD, pp. 40–49 (2004)
10. Kearns, M., Ron, D.: Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. In: COLT, pp. 152–162 (1997)
11. Lewis, D.: A sequential algorithm for training text classifiers: Corrigendum and additional data. SIGIR 29(2), 13–19 (1995)
12. Platt, J.: Fast Training of Support Vector Machines Using Sequential Minimal Optimization. MIT Press, Cambridge, MA, USA (1999)
13. Poggio, T., Rifkin, R., Mukherjee, S., Niyogi, P.: General conditions for predictivity in learning theory. Nature 428(6981), 419–422 (2004)
14. Raskutti, B., Ferra, H., Kowalczyk, A.: Combining clustering and co-training to enhance text classification using unlabelled data. In: KDD, pp. 620–625 (2002)
15. Sanderson, M.: Reuters Test Collection. In: BSC IRSG (1994)
16. Sebban, M., Nock, R., Lallich, S.: Stopping criterion for boosting-based data reduction techniques: From binary to multiclass problem. JMLR 3, 863–885 (2003)
17. Srinivasan, A., Muggleton, S., Bain, M.: Distinguishing exceptions from noise in non-monotonic learning. In: Int. Workshop on ILP (1992)
18. Tong, S., Chang, E.: Support vector machine active learning for image retrieval. Multimedia, 107–118 (2001)
19. Optical sensing arrays. White paper, Tufts University (2006), `ase.tufts.edu/chemistry/walt/research/projects/artificialnosepage.htm`
20. Wilson, D., Martinez, T.: An integrated instance-based learning algorithm. Comp. Intel. 16(1), 1–28 (2000)
21. Witten, I., Frank, E., Trigg, L., Hall, M., Holmes, G., Cunningham, S.: Weka: Practical machine learning tools and techniques with java implementations (1999)