CMS/CS 159 **Spring 2016**

Active Learning Notes

April 26, 2016 Daniel Gu

1 Motivation

A lot of machine learning algorithms involve *passive learners*: learners who passively receive labeled training data (typically in an offline fashion) and then train on this supervised learning set. By contrast, an *active learner* receives *unlabeled* training examples (typically in an online fashion), and gets to choose which examples it will have labeled and learn from. The motivation for considering active learning is that of reducing the *label complexity*: the number of labeled examples needed to obtain a learner of a given accuracy. In the real world, labeled training examples and difficult and expensive to obtain, so learning could be done much more cheaply and efficiently if we extract the maximum amount of information from each labeled example.

A simple example where an active learner achieves and exponential advantage over his passive counterpart is in the problem of learning a 1D threshold. An active learner needs logarithmically many samples as a passive learner to achieve the same accuracy, since he can pick his examples in a "binary search" fashion.

We define *sample complexity* (a passive analogue of label complexity) in the PAC model. In this model, our learner is given learning examples $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$ with f (the *target function*) drawn from a class F given to the learning algorithm and the x_i are chosen independently at random from f's domain according to a probability distribution D. The learning algorithm produces a hypothesis h. We first define error in the PAC model.

Definition 1 ***Error in the PAC Model***
The error with respect to the produced hypothesis h is defined as

$$E(h) = \sum_{x \in h \Delta f} D(x)$$

where Δ is the symmetric difference; that is, the error is the probability that h and f will disagree on a sample randomly drawn from D.

Definition 2 (ϵ, δ)-***PAC Learning***
A learner achieves (ϵ, δ)-PAC Learning if the produced hypothesis h if with probability $1 - \delta$ (with respect to D) h achieves an error $E(h) \leq \epsilon$.

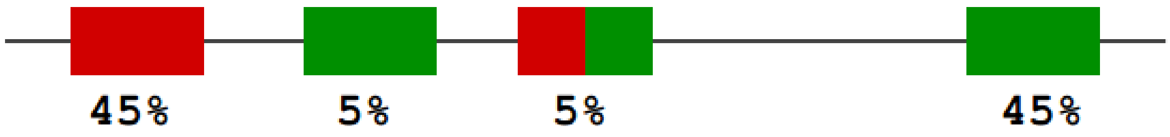
We then want to study the number of samples m that it takes for our algorithm to produce a (ϵ, δ)-learner. For passive learners, results exist which lower bound the number of samples needed; for example, in the problem of learning a halfspace against the uniform distribution on the unit ball in $\mathbb{R}^d$ has been proven to be upper bounded by $O(\frac{1}{\epsilon}(n + \log(\frac{1}{\delta})))$. In general, such upper bounds are a function of ϵ, δ, and the VC dimension, which is the largest number of points that the model f can shatter. In the label complexity setting, we want to do the same sort of analysis, except in the active learning setting. Our goal is to try to get label complexities which are much lower (say, exponentially lower) than their corresponding sample complexities.

There are 3 main settings in the active learning literature. They are *membership query synthesis*, *stream-based selective sampling*, and *pool-based sampling*. Each of these settings assumes that we make queries regarding an unlabeled training example to an oracle, which then labels it for us. They will be discussed in more detail in Sections 3 and 4.

2 Realizable vs. Agnostic Learning

In general, in a *realizable* or *noise-free* setting, we assume that our hypothesis class H contains a hypothesis h^* which makes no mistakes; that is, it optimally classifies all training examples. By contrast in the *agnostic* or *noisy* setting, we have no such assumption; we have no idea whether our hypothesis class is capable of zero loss.

This distinction is especially important in active learning because many algorithms most algorithms are only consistent in the realizable case. When there is noise, these algorithms do not converge to an optimal predictor even given infinite time and training examples. None of the algorithms we will survey in these notes have any theoretical guarantees in the agnostic setting, and many perform poorly in the agnostic case, even though they perform well in the realizable case.



In the picture above, green represents + labels and red represents - labels. If the initial set of labeled examples does not contain any representatives from the second group from left, and the algorithm picks points it is most uncertain about, on even an infinite amount of training data it will converge to a predictor in-between the third and fourth groups, which has 5% error. But the optimal classifier in fact lies between the first and second groups, and has 2.5% error. This means that in the inseparable case, the algorithm is inconsistent (a common situation in active learning), and this is due to the sample bias we introduce by our querying strategy.

See [4] for more discussion of this and other examples.

3 Sampling Models

At a high level, our basic framework for our two common sampling models (stream-based and pool-based) is as follows:

Algorithm 3 ***Sampling Framework***

```
START with a pool of unlabeled data $S \subseteq X$ (and possibly some labeled data as well)
PICK points from $S$ and query for their labels
REPEAT:
    FIT a classifier $h \in H$ to the labels seen so far
    QUERY an unlabeled point in $S$ with some local optimality
```

Sampling models are prone to *sampling bias*. Each time we greedily pick a point with some sort of local optimality (say, closest to the boundary of our current model), we depart from the actual distribution of the training examples. This often means that our active learning algorithms are *inconsistent*: even given infinite training examples, they converge to a suboptimal predictor.

3.1 Stream-Based Selective Sampling

In this model, our learner receives training examples from a stream of data. At each time step, it receives a single training example, and can choose whether or not to ask the oracle to label the sample. These approaches typically work by maintaining the space of hypotheses which are still consistent with the data gathered so far, and then outputting a hypothesis still left in the space when the algorithm terminates. However, calculating and maintaining this space of possible hypotheses is generally computationally intractable, so in implementations approximations are used.

Algorithm 4 ***Stream Sampling Framework***(H, S)

```
INITIALIZE $V = H$
FOR $t = 1, 2, \ldots$
    SEE stream element $s \in S$
    CHOOSE whether to query oracle for label
    IF QUERY
        TRAIN classifier on current labeled examples
        UPDATE $V$
RETURN $h \in V$
```

3.2 Pool-Based Sampling

In pool-based sampling, we have a small amount of labeled training examples and a large amount of unlabeled training examples. We are then allowed to look at the entire pool of training data before deciding which examples to query and label. Usually, we first preprocess the data using unsupervised learning techniques such as clustering, and then pick training examples to query via some greedy metric.

Algorithm 5 ***Pool Sampling Framework***(L, U, H)

```
TRAIN classifier on labeled examples $L$
PREPROCESS the unlabeled examples $U$
REPEAT:
    PICK an unlabeled example $u \in U$ to label greedily
    UPDATE classifier with new example
```

4 Membership Query Synthesis

In this model, the learning algorithm is allowed to ask for the labels of training examples it itself generates, whether it has been sampled or not. More formally, let our domain be X. Define a *concept* to be a subset of X; a *concept* class is then a set of concepts. We often represent the domain X and concept class C by a matrix whose rows are indexed by concepts and whose columns are indexed by domain elements; the (c_i, x_j)-entry is 1 if $x_j \in c_i$, and 0 otherwise. Our goal is to learn a concept from the concept class.

Algorithm 6 ***Membership Query Framework***(X, C, Q)

```
    INITIALIZE $P := C$
    WHILE $|P| > 1$:
        MAKE a query from $Q$
        UPDATE $P$
    RETURN $c \in P$
```

We are allowed to make two types of queries to an oracle to learn more about a concept c. We can make a *membership query*, in which we give a concept c and an element x to the oracle, and it returns 1 if $x \in c$ or 0 otherwise. We can also make an *equivalence query*, in which we give a two concepts c and c' to the oracle, and it outputs "the same" if $c = c'$, or a non-deterministically chosen counterexample $x \in c \Delta c'$, where Δ represents the symmetric difference. We further distinguish between *proper* queries, in which we compare our unknown concept c' to a concept $c \in C$, and *extended* queries, in which we compare it to an arbitrary subset of X (not necessarily in C).

A learning problem is given by the domain X, the concept class C, and types of permitted queries. The goal is to learn some $c \in C$ with the permitted queries. We can then think of a learning algorithm as a rooted tree with two types of internal nodes. A *membership query node* is labeled by some $x \in X$ and has two outgoing edges, labeled 0 and 1. A *equivalence query node* is labeled by a concept $c \subseteq X$ and has $|X| + 1$ outgoing edges, one labeled 1 (for "yes") and the others labeled by the elements of X (which may be produced as counterexamples).

The tree T is evaluated on a concept class C as follows. The root is assigned the entire hypothesis class C. At a membership query node with label x, if we have the set C' assigned to it, we then assign the 0-child the subset of C' such that $x \notin C$, and the 1-child the subset of C' the subset such that $x \in C$. If we are instead at an equivalence query node with label c', we assign the yes-child the set $\{c'\}$ if $c' \in C'$ or the empty set otherwise; at every x-node, we assign the subset of C' which consists of concepts c such that the symmetric difference of c and c' contains x. A learning algorithm is then *successful* on C if in the evaluation of T on C, every leaf l is assigned to at most one concept in c, and every concept is assigned to some leaf.

The minimum over all successful trees for C of the maximum depth of the tree is then the label complexity of the concept class C.

It turns out that in general, it is **NP**-complete to decide whether, given a binary relation representing a concept class C and a depth bound, whether that relation has a membership query algorithm with depth at most the depth bound. Nonetheless, query synthesis tends to be efficient for finite problem domains. Furthermore, the framework can be extended to regression problems.

In the case that our oracle is a human, membership query synthesis algorithms may often fail, because the algorithm may give the human oracle outputs to classify with no semantic meaning.

5 Query Strategies

The heart of active learning algorithms is the way in which they evaluate the "informativeness" of unlabeled training examples. We survey 3 common methods used here.

5.1 Uncertainty Sampling

The idea behind uncertainty sampling is to request the labels of training examples which the learning algorithm is least certain about classifying correctly. The simplest way to measure uncertainty is via maintaining probabilities; we then pick the training example whose best prediction $\hat{y} = \text{argmax}_y \Pr_\theta(y|x)$ has the lowest probability, i.e.

$$x^* = \text{argmax}_x (1 - \Pr_\theta(\hat{y}|x))$$

This can be interpreted as choosing to label the training example with the highest expected 0/1 loss.

This naive approach doesn't take into account the full distribution of labels that our model can make. We can take into account more of the label distribution by instead looking at the margin, which is the difference between our best prediction $\hat{y_1}$ and our second best prediction $\hat{y_2}$, with the intuition that the smaller the margin, the more difficult it will be to correctly distinguish between the labels for that example.

$$x^* = \text{argmin}_x \Pr_\theta(\hat{y_1}|x) - \Pr_\theta(\hat{y_2}|x)$$

However, this still throws away most of the label distribution. The most popular way to measure uncertainty is via the entropy of the label distribution; the training example whose distribution has the highest entropy holds the most uncertainty with regard to the true value of the label.

$$x* = \text{argmax}_x - \sum_i \Pr_\theta(\hat{y_i}|x) \log \Pr_\theta(\hat{y_i}|x)$$

5.2 Query by Committee

At any time, there is a set of hypotheses which are consistent with the (labeled) data seen so far by our algorithm. Query-by-committee (QBC) strategies aim to reduce this space (often called the *version* space) as much as possible. At each point in time, we maintain a list of competing hypotheses; when we need a point to label next, we have all of the hypotheses classify all of the training examples, and then choose the examples on which the models most disagree.

Studying the optimal committee size is still a matter of research, but even committee sizes of 2 or 3 work well in practice. As for a measure of the disagreement, vote entropy

$$x^* = \text{argmax}_x - \sum_i \frac{V(y_i)}{C} \log V(y_i) C$$

where $V(y_i)$ denotes the number of votes for label y_i, or the average Kullback-Leibler divergence of each model from the consensus

$$x^* = \text{argmax}_x \frac{1}{C} \sum_{c=1}^{C} D(P_{\theta^{(c)}} || P_C)$$

where P_C is the consensus probability distribution, are used. The two previously described schemes are hard voting schemes; we could also use soft voting schemes in which we use the posterior probabilities instead.

5.3 Reducing the Generalization Error

We have two approaches to picking training examples to reduce the generalization error. We can directly minimize the expected loss, by choosing to label the example with the smallest expected future loss. This is a very computationally expensive task for which closed form solutions are generally not known. An alternate way to try to reduce the generalization error is by minimizing the variance. It can be shown that the expected squared loss of a model can be decomposed into a noise term, a term which depends on the model class, and a variance term. Taking advantage of this, we choose the example whose addition to the set minimizes the expected variance. This can be done in a closed form format. This strategy is more efficient than directly reducing the generalization error, but still slower than methods like uncertainty sampling.

6 The Disagreement Coefficient

For a passive learning algorithm using a hypothesis class of VC dimension d, we need about $\frac{d}{\epsilon^2}$ samples to get an ϵ-error predictor. For active learning algorithms, we are not only governed by the VC dimension as the leading coefficient d, by another parameter called the *disagreement coefficient*. Let H be a hypothesis class and h, h' be hypotheses in H. Then we can define a metric on H which is the probability that the two hypotheses differ:

$$d(h, h') = \Pr[h(X) \neq h'(X)]$$

Now imagine we are in a *version space* $V \subseteq H$, which we can imagine to represent the hypotheses which are still consistent. Then the disagreement region of V is all points of the domain in which some hypotheses of V disagree:

$$\mathrm{DIS}(V) = \{x \in X : \exists h, h' \in V \text{ such that } h(x) \neq h'(x)\}$$

Given two definitions, we can now define the disagreement coefficient:

Definition 7 ***Disagreement Coefficient***
Given a minimum error hypothesis h^*, the disagreement coefficient is

$$\theta = sup_{r>0} \frac{\Pr[DIS(B(h^*, r))]}{r}$$

Analogously to the sample complexity case, it can be shown that we need to query about $\theta d \log \frac{1}{\epsilon}$ labels to get an ϵ-error learner. This quantity appears in our known general-purpose label complexity bounds.

Disagreement coefficients are known or upper bounds are known for several simple cases. For example, thresholds on a line in $\mathbb{R}$ have a disagreement coefficient of 2, which naturally corresponds to a label complexity of $O(\log \frac{1}{\epsilon})$, as we would expect from our 1D threshold case discussed earlier. For linear separators passing through the origin in $\mathbb{R}^d$ under a uniform data distribution, it can be shown that the disagreement coefficient is upper bounded by $\sqrt{d}$, so we have the corresponding label complexity bound of $O(d^{3/2} \log \frac{1}{\epsilon})$.

References

[1] B. Settles. Active Learning Literature Survey. *Computer Sciences Technical Report 1648*, University of Wisconsin-Madison. 2009.

[2] P. M. Long. An Upper Bound on the Sample Complexity of PAC Learning Halfspaces with Respect to the Uniform Distribution. *Information Processing Letters*, 87(5): 229-234, 2003.

[3] D. Angluin. Queries Revisited. *Proceedings of the International Conference on Algorithmic Learning Theory*, pp. 12-31. Springer-Verlag, 2001.

[4] S. Dasgupta. Two Faces of Active Learning. *Theoretical Computer Science*, 412(19): 1761-1781, 2011.

[5] S. Hannecke. A Bound on the Label Complexity of Agnostic Active Learning. *Proceedings of the 20th Annual Conference on Learning Theory (COLT)*, 2007.

[6] S. Dasgupta and J. Langford. A Tutorial on Active Learning. *Presentation at the 26th International Conference of Machine Learning*, 2009.