Sean Kelley

Machine Learning Final

1.

**Bagging** To use bagging, one produces new training sets sampled from the entire training set and of any size, as long as they are equal. A classifier is produced for each new set. To classify new instances, the instance is run through every classifier, the results are tallied, and the most popular class is chosen.

Bagging doesn’t have much of a problem with low levels of noise as the noise will only be present in some of the trained classifiers. Other classifiers can reduce skew when all the classifiers’ results are used in tandem. High levels of noise will likely still be a problem, since it will be present in high levels in all the classifiers. However, they may still be able to work together to reduce overall error below what it otherwise would be.

**Boosting** Boosting first produces a set of different classifiers for the training data. Misclassified examples are then given higher weights and the classifiers are retrained with the weighted training data for some specified number of iterations. Classifiers are scored based on how well they classify the weighted data; hence the algorithm improves at each step by placing more emphasis on misclassified examples. To classify new instances, each classifier’s weighted vote for a class is totaled and the class with the highest total weighted vote wins.

Boosting weights misclassified instances more heavily, so noise and outliers can drag down the performance as working classifiers are retrained again and again to correctly classify noise. This is especially problematic at 50% noise.

2.

Multiple instances of a binary classifier may be used in combination to produce a k-class classifier.

**One vs. All** For each class, train a corresponding classifier that classifies instances as members of that class or not. To classify new instances, run them through each classifier and choose the classifier that classifies it as a member of its own class with the greatest confidence.

**Binary Partition** Create k classifiers that successively partition the instances into two categories. The first classifies instances as being in classes C1 through Ck/2 or Ck/2+1 through Ck. The next two classifiers divide these two spaces into two parts each again to generate four spaces. Continue until all necessary binary classifiers (the base case) are generated. To classify new instances, perform a binary-search-like series of classifications (ending in a binary classifier).

**Pairwise** Construct binary classifiers for every pair of classes. To classify new instances, run it through all classifiers and select the class that has the most classifications in its favor. It may be necessary to implement an “abstain” option for classifiers so that they may not skew the vote when their vote is irrelevant (i.e. testing an instance that is class A with the B vs. C classifier) or to have classifiers report a confidence value.

3.

Uncertainty sampling may be applied to any number of algorithms as long as they have a metric for producing a certainty value for the classification of any given instance. The least confident should of course be labeled at each step.

**Decision Trees** The most straightforward certainty metric is the difference between different class labels at leaves that have more than one class label. Leaves with only one class may return positive infinity (i.e. they are completely certain) or they may produce a value based on the number of instances at that node. In some sense, a leaf that has two classes, one of which has significantly more instances than the other, is more confident in favor of the more common class than a leaf with one class present but very few instances.

**kNN** The certainty of a particular classification can be computed by the method we used in the last assignment: each neighbor yields a weighted vote, and the certainty of the classification is the difference of the two greatest votes.

**SVM** SVMs implicitly produce a certainty value when classifying instances: the distance of the instance from the dividing hyperplane. Greater distances are more certain to belong to the chosen class; hence there is a direct correlation between distance to the hyperplane and certainty.

**Naïve Bayes** Certainty may be easily computed by taking the difference of the two most likely classifications for the given instance. This is similar in concept to the way kNN computes certainty.

4.

The solution to this problem necessarily includes some concept of either discrete eras (i.e. fixed time frames in which one variant of the concept is dominant such as the “roomy era” or the “gas mileage era”) or a point in some continuous space. If the user specifies era lengths, any classifier can be used – one must simply produce a classifier for each era using only training data from that era, and then feed the unclassified instances to the appropriate classifier. However, this places a burden on the user to specify a good era length (or, in a more complex case, decide arbitrary era boundaries manually), which may be difficult.

**SVM** SVMs can be modified to utilize a continuum of hyperplanes to match the timeline of the input data. To train the SVM, a sliding window of time frames (i.e. overlapping eras) is used to produce a number of different hyperplanes. The hyperplanes can be interpolated so that the hyperplane appropriate to any given timestamp may be calculated and used to classify data with that timestamp. Different methods of interpolation can be used to model different or changing rates of concept drift.

This method assumes that concept drift manifests in a continuous way – i.e., not sudden jumps between different concepts but rather gradual change where interpolation can perform well. Different chosen interpolations correspond to different assumptions about the behavior or rate of the concept drift.

**Decision Tree** Unfortunately, decision trees can’t be easily modified to produce some type of continuous classifier. Instead, they can be changed to discover era boundaries for non-overlapping eras of varying sizes automatically. If timestamps are treated as a feature, they can be used to partition the data like any other feature. The key is to allow repeated partitions with timestamps along any path. In this way the algorithm will invoke timestamps as many times as it sees fit to create the appropriate era divisions. In a three-era example with A, B, and C, the tree may first divide between A and B/C, and then in the B/C branch use timestamps again to differentiate between eras B and C.

One way to empirically observe the rate of concept drift would be to build a classifier for some number of the chronologically earliest instances and then calculate the accuracy for a sliding window of instances. In theory, the accuracy should drop with later instances in a way that reflects the concept drift.