DTAD-20: Advanced Artificial Intelligence: Final Project

Purpose:

- Learn to combine several AI systems into one.
- Understand Ensemble Learning by implementing ADABOOST.

1 Assignment

1. Implement the ADABOOST algorithm on page 667 of the textbook.
2. Include at least 3 types of classifiers in your booster:
   (a) A naive bayesian classifier (NBC)
   (b) A decision-tree classifier (DTC)
   (c) A feedforward neural network classifier (FNC)
3. Test your boosted classifiers on two large data sets.

2 ADABOOST Overview

As described in the textbook and lecture notes for chapter 18, ADABOOST combines the hypotheses generated by other learning algorithms in a voting-based classification process. In the field of Machine Learning, the many sources of these hypotheses include, but are not restricted to, NBCs, DTCs and FNCs.

ADABOOST takes in a data set and divides it into training and test sets, as done in homeworks 2 and 3. Then, the training data is used to build each of the M hypotheses. During testing, each test case is classified by all M hypotheses, with a weighted vote among their classifications being the final answer returned by the booster.

Weighting plays a key role in boosting and is applied in two ways:

- Each classifier (is weighted according to its accuracy on the **training** data. Thus, after the learning system uses the training data to build a classifier (e.g. a decision tree, an NBC, etc), it runs the classifier on the entire training set and sums the weights of the instances that were incorrectly classified. That sum is used to compute the weight of the classifier as:
  $\log \frac{1-error}{error}$
- Each training instance is weighted according to the **previous classifier's** ability to classify it correctly.

Each call to a classifier generator is done with a **differently weighted** training set. The instances are the same, but their weights will vary. Essentially, instances that were poorly classified by a previous classifier will have higher weights when sent to the next round of classifier generation, so that (hopefully) the next classifier will be built so that can handle the difficult instance.

It is important to note that even though each classifier may be generated by the same algorithm, such as a decision-tree learner or a bayesian network builder, and the same training instances, each call to the learner/builder will involve a different set of weights for the training instances. In many cases, the different instance weights will lead to the production of different decision trees or conditional probability tables, respectively.

What follows is a brief description of the 3 classifier types (if not already described in a previous homework) and how each should be tailored to work with ADABOOST.

3 The Naive Bayesian Classifier

Naive Bayesian Classifiers compute the most likely class for a set of attributes using:

- Bayes rule to compute probabilities of classes given attributes by using the probabilities of attributes given classes, and
- the naive assumption that all attributes are conditionally independent, given the class.

The basic idea is easier to understand by looking at diagnosis, where the relationship between causes (diseases) and effects (symptoms) mirrors the relationship between classes and attributes in learning, since classes are can be viewed as causes of the attributes.

For diagnosis:

$$\begin{aligned} & P(Cause \mid Effect_1, \ldots, Effect_n) \\ = \; & \alpha P(Effect_1, \ldots, Effect_n | Cause) P(Cause) \\ = \; & \alpha P(Cause) \prod_i P(Effect_i | Cause) * \end{aligned}$$

In Naive Bayesian diagnosis, the effects are assumed conditionally independent of one another, given the cause.

For classification learning:

$$\begin{aligned} & P(Class \mid Attribute_1, \ldots, Attribute_n) \\ = \; & \alpha P(Attribute_1, \ldots, Attribute_n | Class) P(Class) \\ = \; & \alpha P(Class) \prod_i P(Attribute_i | Class) * \end{aligned}$$

Here, the attributes are assumed conditionally independent, given the class.

Given a set of attributes, the Naive Bayesian Classifier finds the class hypothesis with the maximum a-posteriori probability, h_{MAP}, i.e. the class that maximizes $P(Class) \prod_i P(Attribute_i | Class)$.

3.1 Building a Naive Bayesian Classifier

Computing h_{MAP} is a simple operation when you have data concerning $P(Class_i)$ for all classes (i.e. the a-priori probabilities of each class) and $P(Attribute_j | Class_i)$ for all classes and attributes. These values

come from prior experience, which, in our case, is a simple set of atomic events, such as the one used to build the Bayesian Network in homework 1.

In this case, each atomic event is a training instance consisting of a set of attribute values plus the correct class for that instance.

When running an NBC **without** BOOSTING, the basic approach is the following:

- Read a data set of atomic events and assumed parent-child relationships between the variables.
- Separate the data set into a training set and a test set, with a ratio of 3:1 or 4:1, i.e. most of the instances can be used for training.
- Use the **training set** data to build the a-priori and conditional probability tables in your Bayesian Network. Make sure that each training instance has the same weight. Differential weighting will only be used during boosting.
- For each instance (I) in the **test set**, which consists of attributes(I) and class(I):
  - Use attributes(I) to compute the h_{MAP}.
  - Compare h_{MAP} to class(I).
  - If h_{MAP} = class(I), record a hit, else record a miss.
- Return the test accuracy, which is simply: $\frac{hits}{hits+misses}$

3.2 Weighted Training Instances in Naive Bayesian Classification

In a Boosted Naive Bayesian Classifier, the NBC uses the training set to build conditional probability tables, which then provide the basis for computing the maximum a-posteriori hypothesis, h_{MAP}, as described above. In normal non-boosted NBC, the weights of all training instances are assumed equal.

For example, consider the following simple data set for deciding whether or not to take another card (Hit?) when playing the card game of blackjack (21) and when you have at least 13 points already:

Table 1: Unweighted training-instance set

Hit?	Over 15?	Opponents Remaining	> 10 face cards remaining?
Yes	No	2	No
Yes	Yes	4	Yes
No	Yes	1	Yes
No	No	0	No
Yes	Yes	2	No

Using this table to generate conditional probabilities, consider $P(Over15? = true \mid Hit? = true)$. First we gather all instances in which Hit? = true. Among those 3 instances, two have Over 15? = true, so $P(Over15? = true \mid Hit? = true) = 0.666$ is one entry in the conditional table for Over 15?, given Hit?.

Now consider the same training set, but with uneven weights:

Table 2: Weighted training-instance set

Hit?	Over 15?	Opponents Remaining	> 10 face cards remaining?	Weight
Yes	No	2	No	0.4
Yes	Yes	4	Yes	0.1
No	Yes	1	Yes	0.15
No	No	0	No	0.25
Yes	Yes	2	No	0.1

To compute $P(Over15? = true \mid Hit? = true)$ in this case, gather all instances with Hit? = true, as before. But now, the estimated probability becomes:

$$P(Over15? = true \mid Hit? = true) = \frac{0.1 + 0.1}{0.4 + 0.1 + 0.1} = 0.333 \quad (1)$$

Notice that now we calculate fractions based on the weights of the instances, not simply the counts.

This general idea is used to generate all conditional tables for the NBC. Since each NBC will be based on a differently-weighted training set, each will have different probabilities in its conditional probability tables. Thus, different NBCs can give different answers to the same classification problem. And, of course, if you need to generate k NBCs, then you will need k sets of conditional probability tables.

4 Decision-Tree Classifiers

For DTCs, the weights of the instances will affect the information-gain calculations used to select next-best attributes. Previously, we assumed that all instances were of equal weight when doing entropy calculations. For boosting, we use the instance weights to bias these calculations. So even though an attribute may split the instance set into k positive and k negative examples, the entropy could still be low if, for example, all the positive examples have low weights but all the negative examples have high weights.

In Figure 1, consider a branch of the decision tree from the Chapter 18 lecture notes. In this figure, the weight of each positive or negative instance is written below it. The total weight of all examples along the *shape* branch is 0.35.

One alternative is to use the instance weights only as weights of the 3 branches, but not as parts of the individual-branch entropy calculations:

- Circle: $\frac{0.1}{0.35}(-0.666 \log_2 0.666 + -0.333 \log_2 0.333) = .262$
- Star: $\frac{0.2}{0.35}(-0.5 \log_2 0.5 + -0.5 \log_2 0.5) = .571$
- Diamond: $\frac{0.05}{0.35}(-0.666 \log_2 0.666 + -0.333 \log_2 0.333) = .131$
- Total: 0.964

For example, the $\frac{0.1}{0.35}$ in the circle equation stems from the fact that the ratio of weights along the *circle* branch to the sum of all weights along the *shape* branch is:

$$\frac{.05 + .03 + .02}{.05 + .03 + .02 + .1 + .1 + .01 + .02 + .02} = \frac{0.1}{0.35} \quad (2)$$

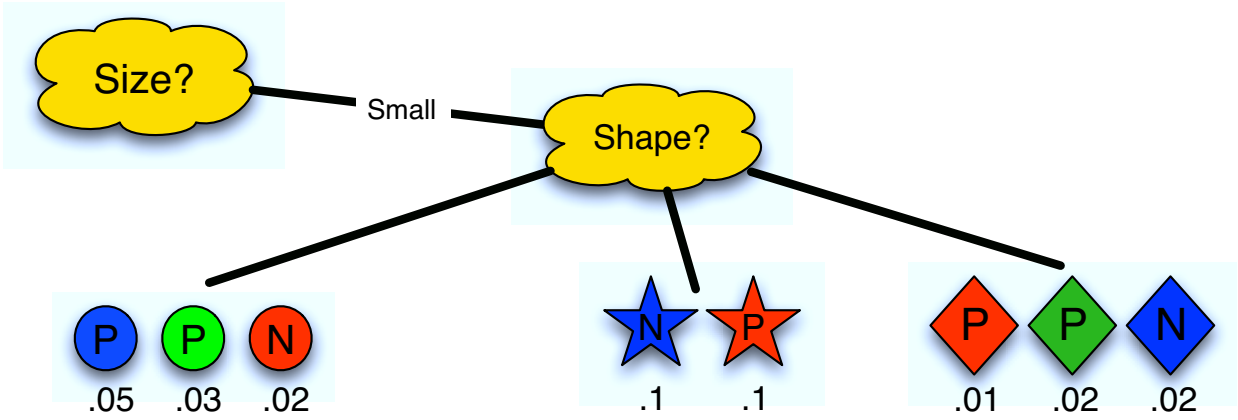

Figure 1: Using instance weights for building boosted decision-tree classifiers

Another option includes the weights in the individual-branch entropy calculations as well.

- Circle: $\frac{0.1}{0.35}(-0.8\log_2 0.8 + -0.2\log_2 0.2) = .206$
- Star: $\frac{0.2}{0.35}(-0.5\log_2 0.5 + -0.5\log_2 0.5) = .571$
- Diamond: $\frac{0.05}{0.35}(-0.6\log_2 0.6 + -0.4\log_2 0.4) = .139$
- Total: 0.916

For example, the 0.8 in the entropy calculation for the circle branch is based on the fact that the ratio of the positive-instance weights to the total weight of the circle branch is:

$$\frac{.05 + .03}{.05 + .03 + .02} = .8 \tag{3}$$

Clearly, then, as the instance weights vary during boosting, different DTCs can be built from the same data set, as long as the instance weights vary between calls to the decision-tree builder.

5 Feed-Forward Neural Network Classifiers

Possible uses of instance weights during backpropagation learning are not obvious, although it might be feasible to adjust the learning rate to correspond to the instance weights. For example, by using a higher learning rate for the higher-weighted examples, the net might be more biased toward them. However, as you have probably found out, the success of backpropagation learning is often very sensitive to the learning rate, so messing with it too much during a run could be a bit risky. But if you define a reasonably safe range for the learning rate and then vary within that range during booster training, you will probably get a decent result.

However, note that a) the building of NBC conditional probability tables (with or without instance weights) is a deterministic process, and b) decision-tree learning is deterministic except for the occasional arbitrary

choice of next-best attributes. Thus, we needed instance weights to insure that different NBCs and DTCs were built from the same training data set. Conversely, feed-forward neural network classifiers (FNCs) are built by a very stochastic process, since the initial arc weights (i.e., weights on the connections between nodes) are determined randomly. Thus, even though backpropagation itself is deterministic, the weights of the starting network are randomly generated. Hence, k calls to an FNC builder will create k different FNCs, even without instance weighting.

For this assignment, the innate diversity of these k FNCs is quite sufficient. So you do not need to include the instance weights in the neural-network building/training process. However, you still need to use the weights in evaluating the success of each FNC. That is, to compute the accuracy of each FNC, you must first train up the network using backpropagation and then:

1. Turn off learning
2. Run each training example through the network one last time.
3. For each example, record whether or not it was classified correctly.
4. Compute the accuracy of the classifier based on the weights of the correctly and incorrectly-classified instances (as described in the standard ADABOOST algorithm).

6 Boosting with all 3 Types of Classifiers

Your ADABOOST algorithm should accept (at least) the following inputs:

1. The number of NBCs.
2. The number of DTCs.
3. The number of FNCs.
4. The percentage of the data set to be used for testing - the rest is used for training.
5. Other key parameters for the individual boosters, such as the learning rate and number of epochs for all the FNCs, the variable topology for the NBCs, etc.

The data sets should be loaded into a data structure that can be used by all the classifier-builder routines. The FNCs may require a special translation of all values into legal input-layer activation levels, so you may need a parallel data structure for these. Use at least TWO different data sets from the UC Irvine Machine-Learning Repository (http://www.ics.uci.edu/ mlearn/MLSummary.html). or some other source. The only constraints on the data set are that it contains at least 100 instances with at least 5 attributes (in addition to the classification) and that the set is based on REAL data, not hand-generated fantasy data.

For each of the two data sets, use a 4:1 training-test ratio and run using several different numbers of classifiers:

1. 2 NBCs
2. 5 NBCs
3. 10 NBCs

4. 2 DTCs
5. 5 DTCs
6. 10 DTCs
7. 2 FNCs
8. 5 FNCs
9. 10 FNCs
10. 2 NBCs + 2 DTCs + 2 FNCs
11. 5 NBCs + 5 DTCs + 5 FNCs
12. 10 NBCs + 10 DTCs + 10 FNCs

The individual parameter settings for the different classifiers should be based on values that worked most successfully in homeworks 1-3.

For each trial, show:

- The training errors for each of the individual classifiers, e.g., the 6 errors the 6 classifiers in case 10 above.
- The test error for the boosted set of all classifiers.

All of these results must be documented in a short report, to be shown to the instructor during your demonstration of the ADABOOST code.

7 A Note on the Irvine Data Sets

In many of the UCI data sets, several of the attributes have quantitative values. If your chosen data set includes such attributes, you may want to preprocess the file and convert all quantitative values into simple qualitative values such as (high, medium and low) or a small set of integers (1, 2, 3 and 4).

For example, if the values of a particular attribute (A) are any real numbers in the range from 3.5 to 9.5, then you can simply convert each value to a qualitative value according to the following simple linear conversion factor:

- $a \in [3.5, 5.5) \rightarrow$ LOW
- $a \in [5.5, 7.5) \rightarrow$ MEDIUM
- $a \in [7.5, 9.5] \rightarrow$ HIGH

All files in the UCI Repository include a .names file, which contains information about the meanings of attributes, the ranges of their values, and even some historical background.

Also, be aware that some data sets in the repository have missing data. This is also specified on the summary page of the repository. If you use a file with missing data, you will have to take special measures to insure that your system does not crash on these cases. For this assignment, it is acceptable to go through the files and just fill in the missing data with made-up (but reasonable) values. All data sets are text files in unix format. I have included a short unix script to convert text files from unix to dos format. Simply type:

```
dirunix2dos data
```

This will convert all .data files in the current directory to DOS format.