CS 440/ECE448: Introduction to Artificial Intelligence

Assignment: Machine Problem 2 **Due date:** April 28, 2011 12:30pm

General instructions

1. Programming Your task is to implement Naive Bayes, Decision Trees, and a Perceptron in `Classify.java`. You need to download this file from

`http://www.cs.illinois.edu/class/sp11/cs440/Classify.java`

Your code will train these three learning algorithms on 2000 training examples and evaluate their performance on 100 testing examples.

You will implement:

```
1. void NaiveBayes()         boolean TestNaiveBayes(int[] datum)
2. void DecisionTrees()      boolean TestDecisionTrees(int[] datum)
3. void Perceptron()         boolean TestPerceptron(int[] datum)
```

The first function will create and train a model while the second one will produce a binary prediction for a given data point. **You may introduce new functions.**

Your code needs to go into the places that we have indicated in the file. You are not allowed to change any other methods. All necessary libraries have already been imported. If you choose, you may *only* import and use other standard Java libraries. All code must be able to be compiled and run on the departmental machines. Any general questions about the assignment or the Java programming language should be addressed to the newsgroup (`class.sp11.cs440`).

Compiling and running the code To compile the code, open a terminal in the directory of `Classify.java` and execute:

```
$ javac Classify.java
```
[1]

To run the code, you need to tell Classify which algorithm to train with, which should be the first argument: *NB, DT, P*.

Please test your code on test set and see how well they perform. Not all classifications will necessarily be correct. Specifically you should give serious thought to how the three algorithms perform on the data and why they don't have perfect performance. Also, if the performance varies, why different algorithms are better at different components of the task.

The second and third arguments (*Train.txt, Test.txt*) specify the input files which we provide:

[1] If the command javac returns a "command not found" error, you need to ensure you have installed the Java Developers Kit (JDK) which can be found at `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

```
http://www.cs.illinois.edu/class/sp11/cs440/Train.txt
http://www.cs.illinois.edu/class/sp11/cs440/Test.txt
```

To run the code execute the following command in the terminal

```
$ java Classify {NB|DT|P} Train.txt Test.txt
```

2. Code comments In your own words, you must clearly comment every part of the algorithms. You will be graded on whether we understand your source code without too much additional effort or not.

3. Explanations of the algorithms These three algorithms all analyze the training data and make predictions on unseen examples. They generalize from previous data. In the header for an algorithm, explain:

- How the algorithm works.
- What assumptions the algorithm make about the data.
- Do Training and Test accuracy differ? Why?
- Does performance change when training with `Train.small`[2]? Is this what you expected?

Data

The data referenced earlier is organized in seven columns (binary features):

1. Class (1000 = crew, 0100 = first, 0010 = second, 0001 = third)
2. Age (1 = adult, 0 = child)
3. Sex (1 = male, 0 = female)
4. Survived (1 = yes, 0 = no)

We provide the text parsing code. All the training data is stored in a global ArrayList. Each data point is an integer array of length seven. Your algorithms will read the global ArrayList for training and a 100x6 int array for testing. Our code will evaluate and print the system's performance including a breakdown by variable of the number of predicted survivors vs the number of actual survivors.

Submission

You will submit to compass the edited *single java file*. **Reminder: Please ensure you have tested your code thoroughly, have documented all code you have written and have provided the proper analysis of both algorithms. Please ensure your code compiles.**

You will submit your code on compass.

[2]`http://www.cs.illinois.edu/class/sp11/cs440/Train.small`

1. Login to `https://compass.illinois.edu`
2. Click Assignments
3. Choose MP1
4. Upload `Classify.java` only.

Task 1: Decision Trees - 3pts

You will write a decision tree whose leaves are binary values indicating if the person survived. You will implement ID3 for evaluating splitting criteria in the tree. This task also requires building an actual tree. We have provided the skeleton from our solution. You are not required to follow this. The only thing that matters structurally is that we can call `DecisionTrees()` which will result in filled prediction arrays.

Smoothing: When you decide which attribute to split a node on, some attributes might have values that do not occur in our data set. Since you don't want to take the log of zero, you should therefore always add a count of 1 to each possible value (even when it does occur), and compute the probabilities accordingly. So if you have n data points and are testing an attribute with v values, the probability of a node with m data items should be $\frac{m+1}{n+v}$.

Feat	Count	Freq	Smoothed	Freq
0	5	100.0	6	85.7
1	0	0.0	1	14.3

Task 2: Naive Bayes - 3pts

You will write a Naive Bayes classifier which has a single output node. Your code will be called from the function: `NaiveBayes()`.

Task 3: Perceptron - 4pts

The perceptron is a binary classifier which takes a vector x and produces an output $f(x) = w^T \cdot x$. Your implementation will be called from the function: `Perceptron()`. Your perceptron should include a **bias** term. This means that the weight vector should be one element longer and the data points should have an extra "1" added to the end of each before taking the product or performing an update. We have left a comment in the code for how to do this and take a dot product.

Debugging

For debugging you may find it useful to print intermediary states of variables in arrays. Java comes equipped with a library `Arrays` which includes a function `toString` which can be called on any primitive array to return a nicely formatted string.