Assignment 4
CSE 473 Autumn 2010

November 24, 2010

The assignment is graded out of 100 points and is due December 10 by email to `guillory@cs.washington.edu` before class. Attach to this email a zip file containing your code and your writeup. Put your writeup in a PDF file titled "writeup.pdf"

Implement Decision Tree Learning (50 points)

Problem 1. (50 points) Implement decision tree learning as described in Section 18.3 of the text book. Your implementation should work for data sets with binary features and binary labels (it can work for other kinds of features and labels too, but we only require that it works for binary features and binary labels). Implement the recursive training procedure given in Figure 18.5 with two extensions:

- Your code should take in a parameter $depth$ which specifies the maximum height of the tree. If $depth$ is 0 then the tree produced should be a single node. If $depth$ is 1 then the tree should be at most a node whose children are leaf nodes (this is sometimes called a decision stump). More generally, if depth is n then there should be at most n non-leaf nodes along any path from the root to a leaf node.
- Your code should also take in a parameter $splittingRule$ which is either $INFO\_GAIN$ or $RANDOM$. If this value is set to $INFO\_GAIN$ then your training procedure should use the information gain heuristic described in the book to pick which feature to use at each node. If this value is set to $RANDOM$ then it should use a randomly selected feature at each node.

Also implement a procedure for predicting the label of an unlabeled test example.

To help you out, we've provided some skeleton code `http://www.cs.washington.edu/education/courses/cse473/10au/a4.zip`. Here is a (non exhaustive) list of things we'll be looking for when grading your implementation:

- Data structures for storing the decision tree
- Correct stopping criteria in your recursive training procedure
- Correct recursive calls in your recursive training procedure

- Correct information gain computations
- Correct prediction procedure
- Reasonably efficient implementation (In training, each call should take time $O(n * d)$ where n is the number of examples and d is the number of features. Prediction should take $O(d)$ for a single example.)

Here are some implementation hints:

- When calculating the entropy of a boolean random variable, make sure to correctly handle the cases where $p = 0$ and $p = 1$.
- Similarly, watch out for division by zero when computing probabilities
- Stick to binary valued features. This will simplify your code.
- If you're getting strange results, try printing out your information gain values. These should all be between 0 and 1.

Test Your Implementation (50 points)

Problem 2. Test your implementation on the data set provided at `http://www.cs.washington.edu/education/courses/cse473/10au/a4.zip`. In this zip file you'll find a file "train.txt" and "test.txt" containing training and test data respectively. Each file is in CSV (comma separated value) format. The first 38 values on each line are binary features and the last value on each line is the binary label. The task in this data set is to predict the winner in a chess end game involving King and Rook versus King and Pawn. The features are derived from the chess positions and the label is 1 if white can win and 0 if white cannot win. The data set is take from the UCI Machine Learning Repository and you can find the original data set here: `http://archive.ics.uci.edu/ml/datasets/Chess+(King-Rook+vs.+King-Pawn)`. We've modified the data set to convert all features to binary and split the data into train and test sets. There are 2109 examples in the training set and 1087 examples in the test set (about a 2/3 train/test split).

Write a brief (no more than 3 pages) discussion of your implementation and results. This writeup should include one or more plots showing the training and test set accuracy on the provided data set for all values of $depth$ between 0 and 30 and for both values of $splittingRule$. When computing accuracy values for $splittingRule = RANDOM$, average over 100 runs. A simple way to plot all this would be as a line graph with $depth$ on the x axis and accuracy on the y axis. You can then include a training accuracy and test accuracy line for both values of $splittingRule$ (4 lines total).

Also discuss in your writeup

- A brief overview of your code (where to find what parts of your implementation)
- Which values of $depth$ and $splittingRule$ gave the best test set accuracy and why

- Which values of $depth$ and $splittingRule$ gave the best training set accuracy and why
- Which values of $depth$ and $splittingRule$ gave the largest difference between training and test set accuracy and why
- Any problems you encountered, possible bugs in your implementation

Extra Credit: Random Forests (10 points)

Problem 3. You probably found that random splitting did not give as good test set accuracy as information gain splitting. Try the following: instead of training a single random decision tree, train m random decision trees. Then to predict the label of an unlabeled example, take the majority prediction of all m decision trees. A reasonable value for m is 100. Variations of this method are sometimes called “Random Forests” or “Random Decision Forests”. Test this method on the provided data set and include this method in your experiments and in your writeup.