Homework 1, Due April 11

MPCS 53111 Machine Learning, Spring 2017
University of Chicago

1. In 3-4 paragraphs describe an efficient algorithm to find the best split in a continuous attribute when building a decision tree. Ensure that —

    ◦ the algorithm takes $O(n \lg n)$ time, where n is the number of examples, and

    ◦ it only tests for splits between examples with different classes.

    See Section “Splitting of Continuous Attributes” on page 162 of “Introduction to Data Mining” by Tan, Steinbach, and Kumar for further details. Please submit the answer in a file `hw1-1.pdf`.

2. **Decision Trees.** Write Python[1] code to implement a decision tree and plot a validation curve. Specifically, write a class `DecisionTree` with the following methods—

    ◦ `__init__(max_depth)`: Initializes a decision tree object where `max_depth` is the maximum depth of the tree that will be built.

    ◦ `fit(X, y)`: Builds a decision tree using information entropy as the splitting rule. `X` is a `pandas` dataframe consisting of the independent variables. `y` is a `pandas` series object consisting of the output, dependent variable. Further details on how to build the tree are given below.

    ◦ `predict(T)`: returns the predicted classes for a dataframe `T` of test examples. Assume `T` has the same columns as `X`, which was used in training.

    ◦ `print()`: prints the current decision tree using BFS order. For each node it prints the depth of the node and whether it is an internal node or a leaf. If it is an internal node, it also prints the column being tested and the test threshold (E.g., “column 45 <= 6.5?” tells that column 45 is being tested against the threshold 6.5). If it is a leaf node, it also prints the class that will be predicted for any example reaching it.

[1] In this course we'll use Python 3.x unless otherwise specified

Building the tree. Assume the variables are either binary or continuous, and make only binary (two-way) splits. Make your code as efficient as possible; use the algorithm developed in 1 above. You may make additional assumptions or use additional arguments for the member functions above. But describe clearly the assumptions you are making in an accompanying discussion file.

The `fit` function should also fill in missing values. There are several sophisiticated mechanisms for filling in missing values, but you should simply replace missing values with the most common value for that variable.

Plotting the validation curve. Test the performance of your decision tree on the Arrhythmia dataset. This consists of 279 independent variables, measuring the age, sex, height, etc., of a patient with Arrhythmia, and several variables derived from an Electrocardiogram of the patient (such as duration of the QRS complex, etc.). Understanding the medical significance of these variables is not required. The target output variable is the last column in the dataset.

You need to plot the accuracy of the training set and the accuracy of the test set as you vary the maximum depth of the tree. Specifically—

- Randomly shuffle the examples in the dataset.
- Divide the examples into three partitions of roughly equal size.
- Vary the maximum depth from 2 to 20 in steps of 2, and for each do the following:
  - Build a decision tree (with the specified maximum depth) using the first two partitions as the training set.
  - Using `predict` find the fraction of examples in the training set in which the tree predicts the class correctly. This is the accuracy on the training set.
  - Now find the accuracy on the test set by using `predict` on the remaining partition.
  - Repeat these steps (building tree, finding training accuracy, finding test accuracy) by choosing the last two partitions as the training set and the first partition as the test set. Repeat

again, but with the first and last partitions as the training set. The average accuracy of the training set for the given maximum depth is the average value over all 3 possible training set combinations. Similarly the average accuracy of the test set is the average of the 3 possibilities. This is called *3-fold cross-validation.*

Use `matplotlib` to plot the values. All this code for plotting the validation curve should be in a function `validation_curve()`, which when called should read in a file from the current directory called `arrhythmia.csv`, do all the processing and create a file `validation.pdf` in the currect directory with the required plot.

Tips. When developing test your code using only 2-3 independent variables and 10-50 examples. Depending on the efficiency of your code you may not be able to plot the validation curve using the entire dataset. In that case, use as many attributes you can such that `validation_curve()` returns in at most 10 minutes. The number of attributes you use should be specified in the discussion.

What you need to submit. You need to submit the following:

(a) A file `hw1.py` with the class `DecisionTree` and the function `validation_curve()`.

(b) A file `discussion.pdf` describing your implemention, any assumptions you made, or anything interesting you discovered about the data.