EECS 349 Problem Set 2

Due 11:59PM Friday, May 1

Overview

In this assignment you will work in groups of 2 or 3 to implement a decision tree learning algorithm and apply it to a synthetic dataset. You will also implement a pruning strategy in your algorithm. You will be given labeled training data, from which you will generate a model. You will be given labeled validation data, for which you will report your model's performance. You will also be given individualized unlabeled test data for which you will generate predictions.

Submission Instructions

Each student should turn in their own copy of the homework. You can work together on the code, but you should write-up your answers to the questions independently. Here is how James Bond would submit the homework. Please adjust for your own name:

1. Create a single PDF file with your answers to the questions below. Name this file `PS2-James-Bond.pdf`.
2. Create a directory (i.e. a folder) named `PS2-James-Bond.code` that contains your source code.
3. Create a file named `README` that explains how to build and run your code.
4. Run your code on test test file and output a file in the same format, but with your predicted labels in the last column. Name this file `PS2-James-Bond.csv`.
5. Create a ZIP file named `PS2-James-Bond.zip` containing:
   - `PS2-James-Bond.pdf`
   - `PS2-James-Bond.code` (directory)
   - `README`
   - `PS2-James-Bond.csv`
6. **Ensure that the zip file contains all of your source code.** You may have to tell the ZIP utility explicitly to include the contents of the subdirectory containing your code.
7. Turn in your code under Problem Set 2 in Canvas.

Download the Dataset

The dataset files are here:

- btrain.csv (labeled training set, 50000 instances)
- bvalidate.csv (labeled validation set, 10000 instances)
- btest.csv (unlabeled test set, 10000 instances)

The dataset is from a synthesized (and therefore fictitious) database of 70,000 baseball games played between two rival teams. Each line of text contains the following information about the game:

- Winning percentage of one team : numeric
- Winning percentage of the opposing team : numeric
- Weather : nominal
- Temperature: numeric
- # of injured players on one team: numeric
- # of injured players on the other team: numeric
- Starting pitcher: nominal
- Opposing starting pitcher: nominal
- Days since one team's last game: numeric
- Days since the other team's last game: numeric
- Whether it's a home or away game (for the first team): nominal
- Run differential for first team: numeric
- Run differential for second team: numeric
- Winner : binary (0 or 1)

The class label is given by the *group* attribute. This is a *binary classification* problem with *numeric* and *nominal* attributes. Some attribute values are missing (as might happen in a real-world scenario). These values are indicated by a "?" in the file. In the test files the class labels are missing, and these missing labels are also indicated by a "?". The test sets are all drawn from the same distribution as the training and validation sets.

If you want, you can imagine that the task is to predict the winner of a baseball game that will be played under the conditions described by each line. However, it's not advised to read too much into the meaning of each attribute, since the data is fictional.

Implementation

For this assignment you will *implement* a decision tree algorithm in the language of your choice. In particular, you should not use Weka or any other existing framework for generating decision trees. You are free to choose how your algorithm works. Your program must be able to:

1. Read the training data file and generate a decision tree model.
2. Output the generated decision tree in disjunctive normal form.
3. Read the validation data file and report the accuracy of the model on that data (i.e. the percentage of the validation data that was classified correctly).
4. Read a test data file with missing labels (question marks) in the last column and output a copy of that file with predicted labels in the last column (replacing the question marks).

Note: your algorithm must handle missing attributes.

A Note About Design

The data files are provided to you in CSV format so that it will be easier for you to read them in. One drawback of the CSV format is that it does not contain metadata (as ARFF does, for example). This means that it is not possible from the data alone to know which attributes are nominal and which are numeric. For example, `zipcode` and `car` are actually nominal attributes that are represented as integers, as described above. Therefore you need to represent this information somewhere. You can either put this information directly in the code that reads in the input files, or you can generate a metadata file of your own and write code that interprets the input file based on the contents of the metadata file.

Regardless of how you translate the input file into an internal representation, **write your decision tree algorithm to handle a general binary classification problem**. The algorithm should be able to handle another binary classification problem with a different composition of numeric and nominal attributes. For example, the algorithm itself should not assume that each example contains exactly 12 attributes, nor for example should it assume that there is an attribute named "elevel" with 5 categories.

Pruning

Add a pruning strategy to your decision tree algorithm. You are free to choose the pruning strategy, but you SHOULD use the validation set for pruning. Note that you don't, for example, iteratively greedily select the one *best* node to prune, as this might be computationally prohibitive. So feel free to choose an approximation (e.g. any node that improves accuracy on the validation set).

Be sure you can run your algorithm both with and without pruning.

Re-using attributes

In your code, in contrast to the decision tree pseudo-code in the lecture notes, you may want to split on the same attribute more than once (for numeric attributes). As a result, you do not want to remove attributes when split and recurse, and you don't need to check if the attribute set is empty. You *should*, however, add a base case in your code to stop when no new split yields non-zero information gain.   Further, setting some limit on the number of splits on a given numeric attribute may be wise, to prevent your trees from growing too large.

Learning curve

As discussed in class, in general, the more training data your algorithm has available, the better it will perform. To illustrate this, you'll be creating a graph showing the performance of your decision trees on the validation set while restricting its training set to .1x, .2x, …, .9x, 1.0x the size of the training set, with finer gradations if desired. For each amount of training data (except the last, which uses all the data), take the average over multiple runs, using a different subset of the training data each time.

Common-Sense Guidelines

1. Write your program so that you do not have to modify code when switching from one task to another or when turning pruning on or off. For example, you might use command-line parameters to enable or disable pruning and to distinguish between the model generation

task, the validation task, etc. An acceptable alternative is to follow the style of LIBSVM and have separate programs for each task, e.g. `model-train`, `model-validate`, `model-predict`, etc.

2. Do not hardcode the names of input or output files in your program. It should be possible to run your program on another input file.
3. Document the usage of your program in the README.
4. While it is not required for this assignment, you may find it useful to have your program be able to output the generated decision tree in a human-readable format similar to that produced by J48 in Weka.

Questions

Put answers to the following questions in a PDF file, as described in the submission instructions.

Answer concisely. You may include pseudocode or short fragments of actual code if it helps to answer the question. However, please keep the answer document self-contained. It should not be necessary to look at your source files to understand your answers.

1. How did you represent the decision tree in your code?
2. How did you represent examples (instances) in your code?
3. How did you choose the attribute for each node?
4. How did you handle missing attributes in examples?
5. What is the termination criterion for your learning process?
6. Apply your algorithm to the training set, without pruning. Print out a Boolean formula in disjunctive normal form that corresponds to the *unpruned* tree learned from the training set. For the DNF assume that group label "1" refers to the positive examples.   NOTE: if you find your tree is cumbersome to print in full, you may restrict your print-out to only 16 leaf nodes.
7. Explain in English one of the rules in this (unpruned) tree.
8. How did you implement pruning?
9. Apply your algorithm to the training set, with pruning. Print out a Boolean formula in disjunctive normal form that corresponds to the *pruned* tree learned from the training set.
10. What is the difference in size (number of splits) between the pruned and unpruned trees?
11. Test the unpruned and pruned trees on the validation set. What are the accuracies of each tree? Explain the difference, if any.
12. Create learning curve graphs for both unpruned and pruned trees. Is there a difference between the two graphs?
13. Which tree do you think will perform better on the unlabeled test set? Why? Run this tree on the test file and submit your predictions as described in the submission instructions.
14. What aspects of the feature set (if any) are a good fit for decision trees, and what aspects aren’t a good fit?
15. Which members of the group worked on which parts of the assignment?
16. BONUS: Define 1-3 features to address any problems you noted in 14. Calculate the feature(s) for each example (the features should be functions of one or more elements from the original feature set), modify btrain.csv to include them, and report the results of running pruned and unpruned 10-fold CV on your new and improved feature set.

Grading Breakdown

This assignment is worth 15 points, broken down as follows:

- Algorithmic Design (Questions 1-5)
  - 5 points
- Disjunctive Normal Form (Questions 6-7)
  - 2 points
- Pruning (Questions 8-11)
  - 3 points
- Learning Curve (Question 12)
  - 1 point
- Output of Algorithm (Question 13)
  - 3 points
- Domain Suitability (Question 14)
  - 1 point
- Feature Design (Question 16 - Bonus)
  - 1 point

It is possible to get up to 12 points of credit without implementing pruning. (If you do not implement pruning, Questions 11-12 can still receive full credit based on the output of the algorithm without pruning.)