Assignment 2: CNF and Decision Trees

CS486/686 – Winter 2020

Out: Feb 1, 2020
Due: Feb 14 at 5pm, 2020
LATE ASSIGNMENTS ACCEPTED until February 18th, 2020 at 5pm with no penalty

Note: The following exercises are to be done individually. All code you write **must be your own**. No late assignments will be accepted. Please hand in your assignment on the LEARN site for CS486/686 `https://learn.uwaterloo.ca`

Question 1 [40pts]: CNF

There are three boxes with three labels on them. Under one of the boxes is a pile of money. Only one box has money under it, the other two have nothing. One and only one of the labels is true. The labels are as follows

- box 1: *This box is empty*
- box 2: *This box is empty*
- box 3: *The money is under box 2*

1. **5 pts** Which box is the money under? Give a simple proof in plain language
2. **10pts** Axiomatize the domain (define a knowledge base (KB) of variables and clauses such that the interpretation above is a model of the KB)
3. **10pts** Convert your KB to a statement in Conjunctive Normal Form (CNF)
4. **15pts** Prove which box the money is under using resolution

Question 2 [60pts]: Text Categorization with Decision Trees

Text categorization is an important task in natural language processing and information retrieval. For instance, news articles, emails or blogs are often classified by topics. In this assignment, you will implement (in the language of your choice) a decision tree algorithm to learn a classifier that can assign a discussion board topic to an article.

Train and test your algorithm with a subset of Reddit posts sourced from `https://files.pushshift.io/reddit/` and processed it using Google BigQuery. The dataset includes the first 1500 comments of August 2019 of each of the r/books and r/atheism subreddits, cleaned by removing punctuation and some offensive language, and limiting the words to only those used more than 3 times among all posts. These 3000 comments are split evenly into training and testing sets (with 1500 documents in each). To simplify your implementation, these posts have been pre-processed and converted to the *bag of words* model. More precisely, each post is converted to a vector of binary values such that each entry indicates whether the document contains a specific word or not.

Each line of the files trainData.txt and testData.txt are formatted "docId wordId" which indicates that word wordId is present in document docId. The files trainLabel.txt and testLabel.txt indicate the label/category (1=`atheism` or 2=`books`) for each document (docId = line#). The file words.txt indicates which word corresponds to each wordId (denoted by the line#). If you are using Matlab, the file loadScript.m provides a simple script to load the files into appropriate matrices. At the Matlab prompt, just type "loadScript" to execute the script. Feel free to use any other language and to build your own loading script for the data if you prefer.

Implement a decision tree learning algorithm. Here, each decision node corresponds to a word feature, and the leaf nodes correspond to a prediction of what newgroup the article belongs in. You will experiment with two feature selection mechanisms as follows:

1. average information gain (evenly weighted across the leaves)

$$I_G = I(E) - [\frac{1}{2} * I(E_1) + \frac{1}{2} * I(E_2)]$$

2. information gain weighted by the fraction of documents on each side of the split, as we discussed in class

$$I_G = I(E) - [\frac{N_1}{N} I(E_1) + \frac{N_2}{N} I(E_2)]$$

where $I(E)$ is the information content in the N examples before the split, and $I(E_i)$ is the information content of the N_i examples in the i^{th} leaf after the split. The first method does not deal well with splits that put a lot of examples on one side, and very few on the other. For example, suppose you have evenly distributed data across the target class (so $I(E) = 1$), and N examples, and 2 features. The first feature splits one example off from the other $N - 1$, so has an information gain using method (b) of $I(E_1) = 0$ and $I(E_2) \approx 1$ so $I_G \approx 1 - [\frac{1}{N} * 0 + \frac{N-1}{N} * 1] = \frac{1}{N}$. Method (a), on the other hand, will make I_G look better than it should be, since it will compute $I_g \approx 1 - [\frac{1}{2} * 0 + \frac{1}{2} * 1] = \frac{1}{2}$. Use $I(\{\}) = 1$ (the entropy of the empty set is maximal).

As discussed in class, build each decision tree by maintaining a priority queue of leaves, sorted by the maximum value of the feature (the information gain of the best performing next feature to split on). At each iteration, split the leaf at which the split will give the largest information gain (according to each of the measures above), and do the split on the word feature that gives that highest gain. Grow the tree one node at a time, up to 100 nodes, by training with the training set only. The point estimate should be the majority class at the leaf.

Report the training and testing accuracy (i.e., percentage of correctly classified articles) of each tree (from 1 to 100 nodes) by producing two graphs (one for each feature selection method) with two curves each (one curve for training accuracy and one curve for testing accuracy).

What to hand in:

- A printout of your code. Your code must be your own, and you *must* use the iterative priority queue model. If your code is recursive, or doesn't expand nodes by order of information gain throughout the tree, 20/60 marks will be subtracted.
- A printout (or hand drawing) showing two decision trees (one tree for each feature selection method) with the first 10 word features selected and their information gain measure. Show the prediction at each leaf.

  Each printed or drawn tree should have 10 nodes and 11 leaves.
- Two graphs (one for each feature selection method) showing the training and testing accuracy as the number of nodes increases.