MP2 Project Report

**Architecture:**

The graph below depicts the architecture of my project:

Stemmer.java

Unit of document:

Document.java

It contains: label, list of word it contains, vector of the document.

DecisionTreeClassifier.java

Input training data

Preparation: Prepare.java

Functions：split, filter, stop word, stem

Input：The original data read from txt files

Output：Format data set, dictionary

Input testing data

Prepare the data and can initialize the format data set and dictionary

Unit of the tree: TreeNode.java

It contains: data set, node name, parent, children

Create Tree: DecisionTree.java

Functions：creating the tree

Input：format data set, dictionary, parent node

Output：the root of the tree

Create decision tree according to the format training data and the dictionary

test testing data

The unit of each word:

wordUnit.java

This class records the usesul information of each word.

It contains: word, its frequency, its weight

test training data

**Preprocessing：**

The preparation of data in my program includes the steps below:

1. Read data from the files and turn them into ArrayList<String> format.
2. For each document(String), split them into words and delete all the non-letter characters. After that, each document is represented by a list of words and each word only includes lower case letters.
3. For each initialized word, we do the stemming and then check if they are in the set of stop words.
4. Using the initialized documents, we can choose a set of words which is useful to be in the dictionary—I chose those words whose frequency is above 0.5% of the total word number. And then I calculate the weight of each word in the dictionary.
5. For each initialized document(which contains only words), we calculate the vector according to the dictionary and get the value of each attribute.

**Model Building:**

1. For matrix:

I firstly create the dictionary and format document so the dictionary contains useful words which are the attributes for matrix. And then according to the dictionary to figure out whether the document contains those words and set 1 for yes and 0 for no.

1. For pruning:

I use recursion to build the tree. For each node, I will find the best word (lowest conditional entropy) and then calculate the gain information. I set a lowest information gain limit so then the node has the lowest information gain, I will let it to be the leaf and set the leaf to be 0 or 1 according to the number of positive documents.

1. Constructing the tree:

For root node, I put all the data into the root. For extension, I will traverse the dictionary and calculate the conditional entropy of each word and finally choose the best word (lowest entropy). Then split the data set into two parts according to the best word and calculate the information gain. If the information gain is above the limit, do the extension. Otherwise, let the node to be a leaf.

**Results:**

(1)Accuracy:

For train.txt and sample\_testing.txt, my program accuracy is 0.72

1. Running time:

For building a decision tree for train.txt, the time can be around 10 second. And for testing sample\_testing.txt, the time can be 2 second.

(3)Pruning performance (on accuracy and time): I change the limit of pruning for different experiment and finally found out that pruning the node when information gain is below 0.001 can get the best result.

**Challenges:**

There were lots of challenges that I faced when I begin the program.

Firstly of all, we need the data preparation. I choose to read the document line by line and put them into ArrayList<String>. After that, for each String, I then delete all the non-letter characters and split them by space char. After deleting the space of each word, I them use stemming to change their form and also check of they are stop word.

Second in line is the organization of the tree. So I start to build the tree first and during the process I can figure out what other functions or components I need to finish the tree.

After building the tree, how to get the useful values is another big b problem like how to find the best word for next splitting, how to get the dictionary and how to calculate the vector of each word.

**Weakness:**

There are two weaknesses in my program.

The first is the selection of dictionary. I just intuitively select the word which frequency is above the certain property of the total words. Thus, there might be some words that appear in one document for many times but have low values for the whole data.

The second problem is when I use tf-idf to calculate the weight of each word and also the vector value of each word, the result is not good. Instead, when I just split the words by whether it exists or not, the result is better. So I finally chose the basic method. The reason might be that I didn’t use tf-idf efficiently or correctly.

Thus, the next step for me to develop the program is to solve these two problems.