Lab 3

In this lab, we have investigated the use of decision tree and adaboost i.e. boosted decision stumps to classify text as one of two languages - English or Dutch.

To Run the Code

Execute following commands to run the code:

To compile - javac lab3.java
Run - java lab3 <given parameters>

To implement this lab we have collected data and trained decision tree/stumps using the two approaches - Decision Tree and Adaboost. The data has then been categorized depending on the features.

Features for classifying Dutch and English

1. Letter 'Y' - If any word in the sentence contains letter 'Y' then it is presumed that this is an English sentence since Dutch language rarely contains letter 'Y'.
2. Word 'de' - It is a dutch word which means 'the' in english. This word is only used in the Dutch language and not in english.
3. Word length - If any of the words in the sentence has length equal to or greater than 10, then this sentence is presumed to be a dutch sentence as English words rarely have these big words.
4. Letter repeat - If any word in the sentence has more than 2 occurrences of a letter, then this sentence is considered as a dutch sentence.
5. Words 'een', 'en' and 'aan' : In Dutch language, een means 'a', en means 'and' and aan means 'on' and thus when a sentence has these words, it is classified as Dutch. These words are not used in english.
6. Word ‘the’ - This is an article which is frequently used in the English language but not in Dutch.

For training the dataset, each input line is classified as 'true' or 'false' for each feature.

Training Class

1. Reads training data from a file specified in the command-line argument.
2. Implements a decision tree learning algorithm if learning type specified as dt (decisionTreeLearningAlgorithm()), or an adaptive boosting algorithm if provided as ada (adaLearningAlgorithm()).
3. The decision tree is built recursively using a divide-and-conquer approach.

4. AdaBoost is built creating a forest of stumps which in this case is stored in the ArrayList
5. Once the model is trained, it is serialized and saved to the specified file (hypothesisOut).

Tree Node

1. Represents a node in the decision tree or adaboost.
2. Contains information such as feature ID, feature value, probability, information gain, and references to child nodes.
3. Implements methods for evaluating stumps (decision points) and evaluating the tree node based on sample data.

Decision Tree

Training

1. First, I had to find the TreeNode or the feature with the least gini impurity that becomes my root node in the tree.
2. Then, I recursively attach new nodes to the parent node by performing the same steps again using the remaining features.
3. Finally, all the data of the node i.e, the gini impurity, featureId, featureValue is stored in all the nodes.
4. This process continues until a stopping point is reached when all the features are exhausted to add to my decision tree.
5. After this another function is executed to label each node in the tree that corresponds to the maximum label for that node.
6. This trained model is now stored in a serialized object and is used for prediction.

Testing and Results

1. The serialized object is deserialized and the root node is taken out to evaluate the test data.
2. The test data is passed to the tree and it evaluates each feature recursively to arrive at a decision.
3. The accuracy for this model is between 88%-95%.

<u>ADABoost</u>

Training

1. In this algorithm, several one - level decision trees are made that are called stumps. Initially, each data is given a sample weight which is assigned as 1/(number of samples). Each decision stump made has some errors i.e. labels that have been wrongly classified. From both the left and right child of these decisions, the total error is calculated.
2. if Label[j] 6= P[j], for all j = 1, 2 . . . N: error = error + wj
3. Based on this error an amount of say for a stump is calculated. It can be done by using below formula:

$$\alpha = 1/2 \cdot \log(1-error/error)$$

4. Lesser the errors, higher the amount of say is. Higher the amount of say, more accurate the decision stump is. The decision stump with a high amount of say has more accurate predictions or has more say in the whole algorithm then other stumps. After calculating the amount of say, the sample weights of the examples with errors is increased and the weight of correctly classified examples is decreased. This is called normalization.
5. if it is an error: $wj=wj \cdot e^\wedge\alpha$ else:

$$wj = wj \cdot e^\wedge(-\alpha)$$

6. When the sample weights are updated, then the weights are normalized so that its value lies between 0 and 1. For normalizing the weights:

$$wj = wj/Summation\ (k=1\ to\ N)\ wk$$

7. Now, this stump along with its amount of say and predicted values is saved in a node and all the nodes are saved in an arraylist. After the first stump is created, the example set is updated with new values. These new values are randomly selected and the one with the error is expected to be selected more often to reduce the error rate. In this particular implementation, I have created 6 stumps that are equal to the number of features or attributes. All the stumps along with its amount of say and prediction is saved in a file.
8. All those stumps created are stored in an ArrayList and is serialized

Testing and Results

1. The serialized object is deserialized and the stumpList is taken out to evaluate the test data.
2. The test data is run iteratively over all the stump list and finds whether which language either en or nl has the largest amount of say combined after taking all the stumps into consideration
3. The accuracy for this model is between 85% - 90%.