Intro to IS Lab 2: Wikipedia Language Classification

For this lab, I implemented Decision Tree and Adaboost Algorithms to classify a given 15-word fragment as either Dutch (‘nl’) or English (‘en’). The program is trained using either algorithm, which is used by the predicter to predict the expected class label.

Features:

To classify the sentence fragment as either English or Dutch, I found the following 13 features to be useful:

1. *beginning_with_ge*: I found that many words begin with ‘ge’ in Dutch language which is not as common in English.
2. *occurrence_of_aa*: Many words in Dutch language contain ‘aa’ as a substring in them. Ex. Amerikaanse.
3. *containing_sch*: Many words in Dutch language contain ‘sch’ as a substring in them, which is not common for English.
4. *containing_cht*: words containing ‘cht’ are common in Dutch
5. *containing_lijk*: words containing ‘lijk’ are common in Dutch.
6. *containing_en*: ‘en’ is a common word in Dutch.
7. *english_articles*: Commonly used English articles like ‘the’, ‘an’, ‘a’
8. *dutch_articles*: Commonly used Dutch articles like ‘een, ‘de, ‘het’
9. *ending_with_ig*: Dutch words end with ‘ig’
10. *average_word_length*: Average word length for Dutch words is greater. I have used 8 as a threshold to distinguish amongst Dutch and English words.
11. *dutch_common_words*: Common Dutch words like "aan", "dat", "en", "te", "voor"
12. *english_common_words*: Common English words like "the", "of", "to", "for", "at", "and", "so", "as".
13. *containing_kt*: words containing ‘kt’ are common in Dutch.

For creating training data (training_data.txt), I used Python’s Wikipedia library to randomly pick 2000 pages each for English and Dutch and split those sentences into 15-word fragments without any filtering any punctuations. The ‘train.py’ file reads these files and extracts above features using function in ‘getAttributes.py’, which is then passed onto training model. Similarly I created a testing file (testing_data.txt) which is used for testing the learning models.

Decision Tree: A supervised learning algorithm to classify data based on its attributes. The tree consists of internal node and leaf nodes, where leaf nodes hold the class label and the internal nodes hold attributes of the data.

For the lab, the decision tree takes as input labelled examples and builds a decision tree based on the entropy and the information gain of each attribute.

Entropy measures the impurity of an attribute in the dataset.

Entropy of an attribute V can be calculated as follows:

$$E(V) = -\sum_i (p_i \log_2 p_i)$$

where i is the unique values of attribute V.

p_i is the probability of the that variable.

Information Gain tells us the importance of an attribute with respect to the target class label. The information gain for an attribute tells us how good that attribute will be for classifying the data. Thus, we choose the attribute that maximizes the information gain.

Information Gain for an attribute V can be calculated as follows:

$$Gain(X, V) = E(X) - \sum_i ((\frac{X_i}{|X|}) E(X_i)$$

where i is the unique values of attribute V.

X is set of samples of dataset.

At each level, a best attribute is chosen based on maximum information gain and the data is split according to its 'True' or 'False' values. The decision tree stops when either maximum depth has been reached or there is no attribute or rows left to split the data on.

For the purpose of this lab, I found depth = 5 to be the best, which gave me about 99% accuracy.

Adaboost: Adaboost is an ensemble learning algorithm which takes a learning algorithm (ex. Decision Tree) as its weak classifier and creates one strong classifier. Initially, the algorithm assigns same weight (1/n) to each sample in the data. Then, it creates a stump using a learning algorithm (in this case a decision tree with depth = 1) and sums the error in prediction from the weights of the samples which were misclassified. The weight of correctly classified samples is decreased by giving more importance to the ones misclassified as follows:

$$\boldsymbol{w}[j] \leftarrow \boldsymbol{w}[j] \cdot error/(1 - error)$$

where $\boldsymbol{w}[j]$ is the weight of sample.

$error$ is the total sum of weights of misclassified samples.

The algorithm creates k hypothesis and assigns weights to each hypothesis as follows:

$$\boldsymbol{z}[k] \leftarrow log\ (1 - error)/error$$

where $\boldsymbol{z}[k]$ is the weight of k^{th} hypothesis.

$error$ is the total sum of weights of misclassified samples.

Where each hypothesis along with its weight votes to classify one data sample. The weighted majority of the set of all hypothesis is taken for classification.

Adaboost can be used to boost the performance of weak learning algorithms by learning from the errors made in previous stages. The number of decision stumps to be created by the algorithm can be controlled. For the purpose of this lab, I have used K = 8 which gave me about 97% accuracy.

It can be noted that since decision tree is performing well on its own, Adaboost algorithm gives about the same level of accuracy.