# Lab 2 Documentation and Implementation

## 1) Feature Description and reasoning:

English and Dutch are both West Germanic, and descend further back from the common ancestor language Proto-Germanic. Their relationship however, has been obscured by the lexical influence of Old Norse as a consequence of Viking expansion from the 9th till the 11th century, and Norman French, as a consequence of the Norman conquest of England in 1066. Because of their close common relationship - in addition to the large Latin and French vocabulary both languages possess - many English words are essentially identical to their Dutch lexical counterparts, either in spelling (*plant*, *begin*, *fruit*), pronunciation (*pool = pole*, *boek = book*, *diep = deep*), or both (*offer*, *hard*, *lip*) or as false friends (*ramp = disaster*, *roof = robbery*, *mop = joke*).

I strongly felt that checking the presence of certain substrings, checking the average length of words, checking of certain special characters (á é í ó ú à è ë ï ö ü) play a vital role in differentiating between both the language as based on text analysis. My feature set involves checking if or not there is a presence of a substring in a given text, for my feature set I took a reasonable size of a substring say 2 to 3 words and to keep it fair I took the top 5 most used words in English and Dutch respectively and the list is as follows:

1)If or not “the” is present? **(ENGLISH)**

2)If or not “of” is present? **(ENGLISH)**

3)If or not “an” is present? **(ENGLISH)**

4)If or not “to” is present? **(ENGLISH)**

5)If or not “his” is present? **(ENGLISH)**

6)If or not “van” is present? **(DUTCH)**

7)If or not “het” is present? **(DUTCH)**

8)If or not “oo” is present? **(DUTCH)**

9)If or not “ik” is present? **(DUTCH)**

10)If or not “ij” is present? **(DUTCH)**

Moreover, only using 4 attributes out of the following a reasonable amount of accuracy (85) can be achieved, so that clearly justifies the checking of most commonly occurring words/substrings in the given language.

A problem with this approach was that we could often get false positives, consider a frequently occurring Dutch substring found in an English sentence, thus the words had to have a high frequency but also be exclusive to only one language that is Dutch or English.

## 2) Data collection and processing:

I collected data from Wikipedia and stored in my dat.txt file which is a 200-line text file (can be found in submission). Then from the file I skip the first 3 words (label) and store in a list of lists. My processing function works on this and checks the presence of a substring in order to assign a true or false value and the result is then stored in a data frame, then this data frame is stored in a .csv file which will be input for our decision tree.

**3) Decision Tree Implementation:**

We now pass the .csv as input to the tree which calculates the maximum information gain of each attribute by subtracting the entropy of each attribute from the target attribute. Here is a sample run for each of the attributes:

| Attribute name | Information Gain |
|---|---|
| the | 0.4869832481739663 |
| of | 0.23865214301164772 |
| an | 0.019048872732746025 |
| to | 0.06923954987809289 |
| his | 0.1440649987816801 |
| van | 0.4390710068293102 |
| het | 0.1771888658831342 |
| oo | 0.14027825708321084 |
| ik | 0.07743756714254924 |
| je | 0.09686559222031232 |

The decision tree is built in a nested dictionary with attribute name as key and result stored in pair, here the value with the largest information is passed and the function is called recursively and chooses the value with the highest information gain as the best split and here is a sample run:

```
{'the': {False: {'van': {0.0: {'ij': {0.0: {'oo': {0.0: {'het': {0.0: 'en',
                                                  1.0: 'nl'}},
                                         1.0: {'an': {0.0: 'nl',
                                                  1.0: {'het': {0.0: {'to': {0.0: 'nl',
                                                                             1.0: 'en'}},
                                                                1.0: 'nl'}}}}}},
                                  1.0: 'nl'}},
                           1.0: 'nl'}},
     True: {'het': {0.0: 'en',
                    1.0: {'van': {0.0: {'ij': {0.0: 'en', 1.0: 'nl'}},
                                  1.0: 'nl'}}}}}}
```

We split the train and test data based on our requirements by slicing (100 and 100 in my case) it and passing it to our test and train functions.

The predicted output is then stored in a text file with the label for each line and the accuracy of the run is printed.