- Feature List Description
    - Average Word Length: This is the average word length of a sentence. This is a range of Numbers from 3~11. The highest frequency was on range in about 3-7. Hence that was the threshold selected for this attribute.
    - Dutch Articles: Dutch articles, as presumed proved to be a good classifier. The words ‘een’, ‘det’ etc are calculated for each sentence.
    - English Prepositions: A list of english language prepositions. If the number was greater than 3 in a sentence, that was considered true.
    - Punctuations: Number of punctuations in a sentence. Some language sentences have more punctuation.
    - Vowel Pairs: Vowel pairs in dutch language are frequent. Hence this was selected

- Decision Tree Learning:

  The decision tree created has the following stopping criteria.
  Probability of the classifications in a node
  Number of classifications in the node
  And till the point there is data in the dataset

  This is due to the following reasons:
  If the probability of a particular class in a node is more than 89%. I.E majority nodes are of one class, we stop because there is very less chance of further classification of that dataset
  If the number of nodes in a class is less than 10, there’s less chances of it being classified further since the training data is huge.
  At every point, we check if there’s data and only then we enter.
  The decision tree is built recursively and it also writes the decisions to the classifier.

  Since we eliminate the column the data is split on, the max depth wouldn’t be more than 4.

  Now, all these decisions are taken based on information gain and entropy values. We evaluate all Columns based on the highest info gain returned. Highest info gain implies that the dataset is supposed to be split on this particular column.

  Now, once the decision tree runs, we write a classifier method. The decision tree method recursively writes rules to this classifier file and the rest of the code is written in this classifier method.

  This python file also has a method that writes the final classifications of the testing data on a csv file called **dt_classifications.csv** and also prints the decisions

  Note: When we perform the training, it makes the classifier file. When we run the command for it, that runs quickly. Although, it takes about 2 minutes for the classifier.py file to be made. In the below case, I am talking about the file ‘classifierdt5.py’. So I request you to wait for that. I have also added the txt files copy pasted from wikipedias.

    - Instructions to run the decision tree mode with the help of an example:

```
(base) sneharohra@Snehas-MBP AIHomeworks % python3 train.py train_input.dat class.py dt
(base) sneharohra@Snehas-MBP AIHomeworks % python3 predict.py class.py test.dat
```

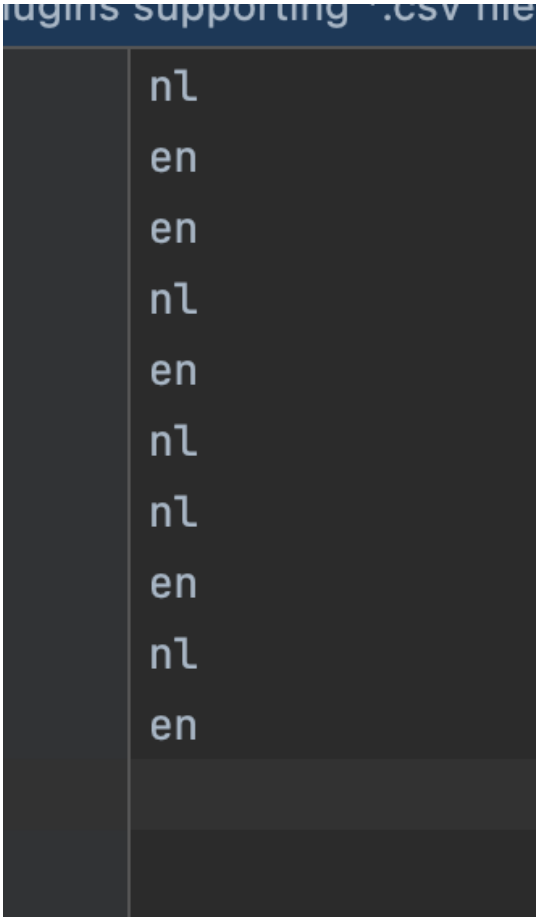
        - For training: <train> <training_data> <classifier_file_name> dt

- <train> : “train.py” (python file that calls the methods associated with training preprocessing)
- <training_data> : dat file of the training data which will get preprocessed
- <classifier_file_name> : Python file that gets created with the decision tree rules which ultimately writes a CSV for decisions and also prints the decisions
- dt: Mode decision tree

■ For predicting: <predict> <classifier_file_name> <test_data>

- <predict> : “predict.py” (python file that calls the methods associated with prediction preprocessing)
- <classifier_file_name> : Python file that gets created with the decision tree rules
- <test_data> : dat file of the testing data which will get preprocessed

○ To test the decision tree classifications, I used the ‘train.dat’ file(It’s in the zip) and it gives the following predictions.(It also prints). It has a 90% accuracy.



○ Best Tree: My best tree decisions will be in the classifier.py file. For the training data I used, the following were the decisions made. This is a screenshot of the classifier file my original file made.

```
def classify(data_list):
    classifications = []
    for i in range(len(data_list)):
        if data_list.iloc[i][1] == 0:
            classifications.append([False])
        else:
            if data_list.iloc[i][4] == 0:
                if data_list.iloc[i][0] == 0:
                    if data_list.iloc[i][2] == 0:
                        if data_list.iloc[i][3] == 0:
                            classifications.append([False])
                        else:
                            classifications.append([True])
                    else:
                        classifications.append([True])
                else:
                    classifications.append([True])
            else:
                classifications.append([False])
    return classifications
```

The classification result is based on the true and false result. If the classification is true, it is an English Sentence. Dutch if false.

- Adaboost technique:

  Adaboosting is done by performing a few calculations on the first level of the decision tree. This is done using the amount of say, which determines how strongly a particular line is of a particular language. This is done 5 times in a loop, since we have 5 attributes to consider.

  - Instructions to run the decision tree mode with the help of an example:

    ```
    (base) sneharohra@Snehas-MBP AIHomeworks % python3 train.py train_input.dat class2.py ada
    (base) sneharohra@Snehas-MBP AIHomeworks % python3 predict.py class2.py test.dat
    ```

    - For training: <train> <training_data> <classifier_file_name> ada
      - <train> : “train.py” (python file that calls the methods associated with training preprocessing)
      - <training_data> : dat file of the training data which will get preprocessed
      - <classifier_file_name> : Python file that gets created with the decision tree rules which ultimately writes a CSV for decisions and also prints the decisions
      - dt: Mode decision tree
    - For predicting: <predict> <classifier_file_name> <test_data>
      - <predict> : “predict.py” (python file that calls the methods associated with prediction preprocessing)
      - <classifier_file_name> : Python file that gets created with the decision tree rules
      - <test_data> : dat file of the testing data which will get preprocessed

  - To test the adaboost classifications, I used the ‘train.dat’ file(It’s in the zip) and it gives the similar predictions as decision trees.(It also prints).
  - I have used all stumps for better accuracy and the technique is run for all attributes. And I get the split columns in the following order:
    - Dutch Articles
    - Vowel Pairs

- Punctuations
- Average Word Length
- Prepositions