**How to use the code:**

First a model will need to be generated, start by calling the script with the train option along with the training data, the features file, the output path for the model, and the algorithm. To choose which algorithm to use simply add ‘dt’ for decision tree or ‘ada’ for adaboost to the end of the script call. This will generate a pickle model for the data provided utilizing the algorithm chosen and the features of English and Dutch languages.

* **Example 1: python3 lab3.py train training.dat features.txt best.model dt**
* **Example 2: python3 lab3.py train training.dat features.txt best.model ada**

Once the model has been generated, call the script again but with the predict option. This will need a prediction file, the same features file, and the model generated. The script will evaluate each line in the prediction file and utilize the model with the features to determine whether the line is in English or Dutch. The script will then print out whether the line in the prediction file is English, ‘en’, or Dutch, ‘nl’, for all lines in the prediction file to the command window.

* **Example: python3 lab3.py predict predict.dat features.txt best.model**

**Features:**

The features I choose for training and predicting uses common articles and words from both languages along with specific characteristics. These features provided me with the best accuracy for identifying sentences. The Dutch common substrings do not occur in English.

* English Common Words:
  + of
  + to
  + for
  + at
  + and
  + so
  + as
* English Articles:
  + an
  + a
* English Specific Feature:
  + the - both a common word and an article
* Dutch Common Words:
  + aan
  + dat
  + en
  + te
  + voor
* Dutch Articles:
  + een
  + de
  + het
* Dutch Common Substrings
  + ge
  + aa
  + sch
  + lijk
  + cht
  + ig
  + kt

**Decision Tree:**

The decision tree uses a class structure for building the tree itself: attributes representing the feature index, true representing the left child node, and false representing the right child node. The decision tree is trained through recursive splitting that finds the best gain for each feature and based on the feature is split into subsets and then performing recursion of both of those subsets. The infoGain function and the entropy function both have weight checks, this is for when boosting is being used, and are used to retrieve the probabilities of each parent and child node and does that recursively until a leaf node is found. For prediction, the predictDecisionTree function determines if a node is a leaf or if it is a branch. I was able to determine that a max\_depth of 6 yielded the best accuracy results based on my training data and the features. This was done using 2 functions, testParameters and calculateAccuracy, that tested for the best max\_depth value and the best number of trees in a range from 1 to 10. My test results, based on GradeScope, showed an accuracy of 0.9835.

**AdaBoost:**

The boosting learning uses a class structure which stores the h representing the weak learner (decision stump) and z representing the weight assigned to the weak learner (alpha). The training of the boosting is done through iterating the number of trees values. For each value, it calculates a stump and then makes a prediction using the stump and the examples from the data. This is used for error calculations and computing the alpha. Weights are then updated, and the stump and the alpha are stored in the class. The prediction for boosting uses each weak learner to predict either English or Dutch, this is then added together to determine the language of a sentence and prints either en or nl. Much like with the decision tree, I used the 2 functions I created to determine how many trees to use. I found that a value of 9 worked the best in a range of 1 to 10. My test results, based on GradeScope, showed an accuracy of 0.9783.

**Additional Notes:**

The 2 functions I created for determining max depth and how many trees to use had some inconsistency when ran multiple times. It is not a perfect function, but I found that it was much better than passing my script through GradeScope with 100 different combinations. I have it commented out in the script so that it does not interfere with any result testing.