CS 677 Final Project

# Title: English And Dutch Text Classifier

# Author: Han Xu

# Email: [han10@bu.edu](mailto:han10@bu.edu)

# Date: August 11, 2020

# Summary

GitHub repository: <https://github.com/hanCodeHub/NLP-classify-english-dutch>

## Objective

The goal of this project is to train a machine learning model that can analyze and predict the language of a given text message. A model is trained on a given dataset with text messages in either English or Dutch. The model is then used to predict the language of new message instances to be either English or Dutch.

For instructions on how to run the program, please refer to *README.md*, which can also be viewed from the GitHub repo linked above. The following sections explain the purpose of each python module and the methodology of the program.

## Dataset

This program uses a public dataset found [here](https://www.kaggle.com/mdhrumil/english-dutch-text-classification-nlp-beginner) on Kaggle. There are 1070 text records in either English or Dutch that the original contributor sourced from Wikipedia. Each record is saved as a separate row within the text file, and each one is classified as either ‘en’ for English or ‘nl’ for Dutch. Note from the contributor: “All the words of a record belong only to one class. There is no record which contains some words belonging to ‘en’ and others belonging to ‘nl’.”

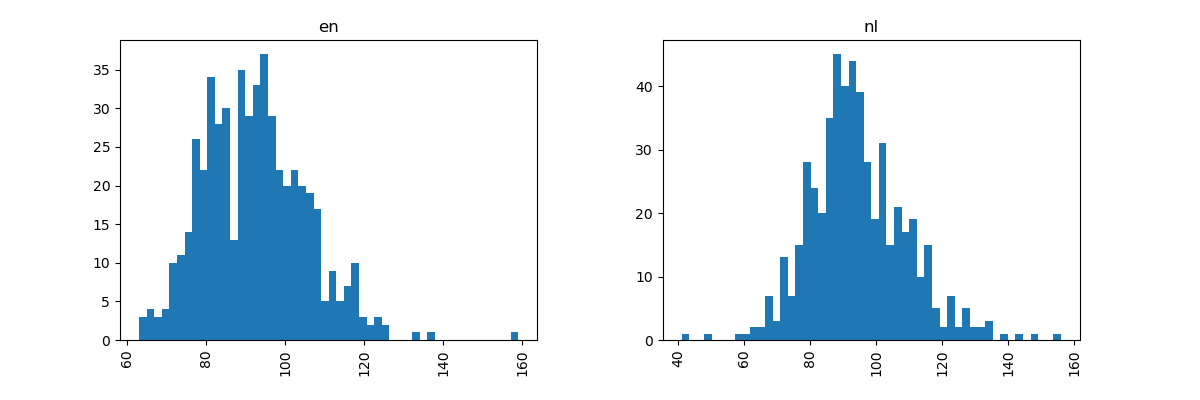
First step is to parse the data from the text file into a pandas dataframe. That’s what the *data.py* file is for. It contains utility functions that process data, which other python modules need. For example:

* *read\_data()* reads a given text file into a dataframe
* *split\_data()* splits a given dataframe into testing and training sets
* *clean\_message()* extracts only the words from a given string and returns them in a list

The section on modeling will explain why cleaning messages this way is necessary.

## Exploratory Analysis

Running *explore.py* will show a sample of the dataframe in the terminal. This way we can see what the messages look like in both languages. It also shows a total count of 535 Dutch messages and 534 English messages, which means that the dataset is evenly split between the two classes. Finally, this script plots a histogram of message length for messages belonging to each language:



*txt\_len\_hist.png*

From this plot we can see that length of Dutch messages are normally distributed, whereas the length of English messages have a few outliers. However, if we discount the few outliers in both histograms, then both English and Dutch message lengths are centered around 80-100 characters. This means that message length is **not** a good differentiator for a model to classify the language label. The following section explains a different method of counting.

## Modeling

Training the model occurs in *model.py*. In order to create a model, the program must first convert textual data into numerical data. In theory, the bag-of-words model allows the program to convert each word of a message into the count of its occurrence in that message (also known as Term Frequency). Only then can it be used as a feature for training the classifier. Here is an example from [Wikipedia](https://en.wikipedia.org/wiki/Bag-of-words_model):

1. John likes to watch movies. Mary likes movies too.
2. "John","likes","to","watch","movies","Mary","likes","movies","too"
3. BoW1 = {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1};

Note that step 2 extracts the words from a message without any punctuations, and step 3 creates a ‘bag’ of word counts. To do this, the program uses the *CountVectorizer* method from *sklearn* to create a matrix of unique word count by message for the given dataset, while passing in the *clean\_message* function to strip out anything that’s not a letter. This matrix is considered sparse, because many messages will display 0 for many words.

Next, each word count needs to be weighted and normalized so that the classifier can determine the statistical importance of a word. The program does this with the *TfidfTransformer* method from *sklearn*, which transforms the word count values based on ‘Term Frequency-Inverse Document Frequency’. Term Frequency increases the importance value of a word based on its frequency in a message, and Inverse Document Frequency decreases the value based on its frequency in the entire document (all messages).

Finally it’s time to train the model. The program uses both *RandomForestClassifier* and *MultinomialNB* from *sklearn* to compare results from two popular classifiers. Running *model.py* will display the performance metrics of each one in the terminal. Naïve Bayes in particular is commonly used for text classification, because it uses an efficient way to determine the label for each new message M: What is the probability of M given it’s a Dutch message vs the probability of M given it’s an English message.

The program then uses a *Pipeline* from *sklearn* to combine all of the above steps into a single stream of events, which can be fitted onto the training set just like any regular classifier. From there, it’s another simple method call to predict the label on a given message. The results from using the Naïve Bayes classifier is 99.81% accuracy, and similar for Random Forest (though less efficient). Please refer to the terminal for more performance metrics.

## Production

Truly productionizing the ML model is not within scope of this project, so instead a *main.py* file is provided as a demo of its capabilities. Again, please refer to README.md for details. Running this file directly allows a user to input a message for prediction. It does this via a wrapper function that parameterizes the functions from *model.py* and returns the language.

# References

Kaggle dataset: <https://www.kaggle.com/mdhrumil/english-dutch-text-classification-nlp-beginner>

Bag-of-words model: <https://en.wikipedia.org/wiki/Bag-of-words_model>

TF-IDF: <http://www.tfidf.com/>

Text classification: <https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html>