**Library overview:**

The project was written in Python 3. To easily install the required libraries use the requirements.txt file.

1. **Pandas**: Library to easily manipulate data in the form of long arrays or matrices.
2. **Numpy**: Makes calculations across sets of data easier
3. **Matplotlib**: Data visualization tool
4. String: Used in this instance for the useful list of special characters in the library.
5. Pickle: allows dumping of file that contains the trained machine learning model.
6. **Scikit-learn**: Library containing many useful tools for machine learning, including splitting into test data, applying different machine learning algorithms easily and analysing effectiveness of trained models.
7. **Joblib**: pipelining for Python

**Data Analysis/Preparation**

After loading the csv file the first order of business was to have a look at the data itself and how the data was spread among the different classes (languages):

|  |  |  |
| --- | --- | --- |
|  | text | language |
| 0 | Ship shape and Bristol fashion | English |
| 1 | Know the ropes | English |
| 2 | Graveyard shift | English |
| 3 | Milk of human kindness | English |
| 4 | Touch with a barge-pole - Wouldn't | English |

From the initial look at the data it was apparent that there were capital letters and special characters present that were consistent among all the languages, thus would have no effect on the effectiveness of the training and could in fact negatively influence the results. The “text” column was then cleaned and any capital letters were made lower case and special characters were removed.

It was also noticed after scanning through the data that there were some duplicate and empty cells. These were removed from the dataset using pandas:

df = df.drop\_duplicates(subset=[**'text'**], keep=**False**)

df = df[~df[**'text'**].isnull()#select only rows that are not null

After this was done the amount of data per language was investigated. The results can be seen in the below table:

Instances per class:

|  |  |
| --- | --- |
| English | 2039 |
| Afrikaans | 637 |
| Nederlands | 67 |

“Nederlands” only had 67 entries and was severely under-represented. This “class imbalance” issue might cause the model to ineffectively predict data for this language. A technique was used to rectify this by randomly copying the “Nederlands” data until it had the same number of data points as “Afrikaans”. The new database had the following number of samples per class:

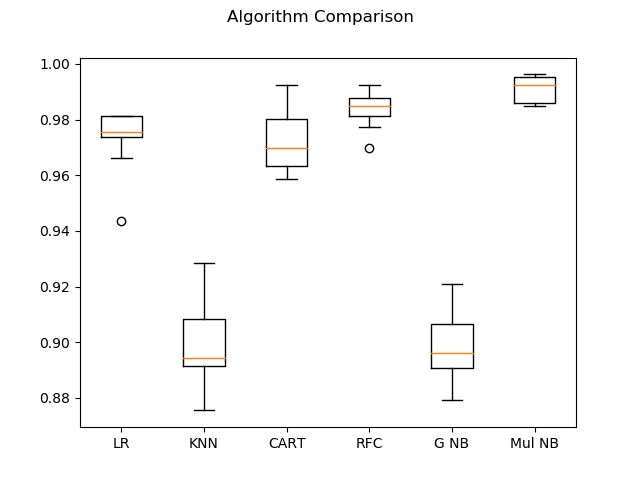
|  |  |
| --- | --- |
| English | 2039 |
| Afrikaans | 637 |
| Nederlands | 637 |

The nature of the problem involves classifying phrases to belong to a certain language. Naturally a classification technique was needed.

**Model Architecture**

The Multinomial NB (MNB) algorithm has been shown to work very well for text classification and thus was chosen as the preferred model. Text data can’t be assumed to follow a gaussian (normal) distribution therefore the MNB was chosen instead of a Gaussian Naïve Bayes approach.

To confirm that the MNB is in fact the most suitable model for this problem a cross validation analysis (bins = 10) on the data was done using a few of the well-known machine learning algorithms and the results can be seen in the plot below (80% of the data was used for training, y-axis represents accuracy):



LR = Logistic Regression

KNN = K Neighbors Classifier  
CART = Decision Tree Classifier  
RFC – Random Forrest Classifier  
G NB = Gaussian Naïve Bayes

Mul NB = Multinomial Naïve Bayes

From the chart it was seen that MNB was indeed the most successful algorithm (without optimizing any model parameters) in classifying the data. Showing a narrow distribution and accuracy of around 99%.

The Scikit Learn function MultinomialNB was used to apply the algorithm to the data after the text was prepared as discussed in the previous section.

*MultinomialNB(alpha.0, class\_prior=None, fit\_prior=True)*

The laplace smoothing parameter (alpha) was set to 0, as the data do not need to be in a normal distribution with MNB. The fit prior probability settings were kept as is with the prior probabilities being calculated.

**Training**

The algorithm was applied to the data also with 80% used for training. This amount was chosen to give the model enough training data, but also left some data for validation.

To feed String data into the algorithm it must first be changed to some form of numerical input. The strings were converted to count vectors. This process takes all the words analysed and uses this as the vocabulary. Each individual string is then analysed and the output is given as a matrix of occurrences of tokens (specifically words in this case). Term frequency (TF-ID) was not used due to the data not being large amounts of text with too many ubiquitous words (“The”, “a”, “he”, etc.).

The test data was loaded into the algorithm using the scikit-learn function MultinomialNB(), whilst simultaneously converting the strings using scikit-learn CountVectorizer() using the pipe command also from the scikit-learn library:

pipe.fit(X\_train, y\_train)

**Testing**

After successful training of the model, testing was done on the remaining 20% of the data. A list was made from the “text” inputs the language for each was predicted using the trained model. These results were then compared to the known languages for each of those strings. A confusion matrix was drawn up to compare the results (The rows show how many elements of each language was classified as what language):

|  |  |  |  |
| --- | --- | --- | --- |
| language | Afrikaans | English | Nederlands |
| Afrikaans | 136 | 1 | 1 |
| English | 0 | 384 | 1 |
| Nederlands | 0 | 0 | 140 |

The matrix showed that the model did extremely well in classifying the data with only a few samples being classified incorrectly for “Afrikaans” and “English” and all the “Nederlands” samples being predicted to the right class. These results were confirmed with a classification report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| Afrikaans | 1.00 | 0.99 | 0.99 | 138 |
| English | 1.00 | 1.00 | 1.00 | 385 |
| Nederlands | 0.99 | 1.00 | 0.99 | 140 |
| avg / total | 1.00 | 1.00 | 1.00 | 663 |

The Precision indicates how many false negative results there were, while the Recall gives an indication as to the number of false positives, lastly the f1-score is a weighted average of the precision and recall. The support is the number of samples for each class.

From the classification report and confusion matrix it was obvious that the model performed extremely well in all departments, classifying the data into the correct language with almost 100% accuracy.

Finally the accuracy, defined as the ratio of correctly predicted samples to total samples was:

0.9954

The accuracy also reflected what the classification report showed that the model performed very well in classifying the 20% test data.

**Saving the model**

The model was deemed good in predicting the data and saved using the library joblib. This stored the model in a file that can be loaded and used in another program. The model was then tested on the entire .cvs file (for lack of having new data) and the accuracy was calculated to be 0.998906306963179. The confusion matrix was also drawn:

|  |  |  |  |
| --- | --- | --- | --- |
| language | Afrikaans | English | Nederlands |
| Afrikaans | 634 | 1 | 2 |
| English | 3 | 2032 | 4 |
| Nederlands | 0 | 0 | 67 |

The model can now be loaded using joblib and applied by storing it in a folder along with the csv file containing the data. A program was written called “Trained\_Classifier.py” which should also be stored and run from the same folder as the other two files. The only change that needs to be made is changing the “filename” parameter to the name of the csv file that needs to be tested.

If the model needs to be used for any other purposes just use the following line of code to call it:

pipe = joblib.load(**'model.pkl'**)

Then use:

pipe.predict(X), where X is the vectorized string you want to predict.

**Limitations**

Due to how the fact that the NB used the vocabulary from the input data to calculate the probabilities, it might be possible that on new data with words not contained in the training document could cause some inaccuracies. To counteract this one could either find more training data or use another way of converting the strings to numerical values that look at patterns between the letters to try and identify the language.