Report on Classification of Authors by Texts

Wenyi Liang, Ernesto Javier Aguilar Madrid, Eliane Maria Zanlorense
M20190551, M20190559, M20190802

1 Introduction

This report illustrates the approaches used on a project and the results. The aim of the project is to develop a model for classification of authors by text excerpts with 500 or 1,000 words. Several books from six Portuguese authors and text excerpts that contain no information about the authors were given for us to conduct the project.

2 Approaches

The project was done in Jupyter Notebook with Python 3. Libraries such as Pandas, Numpy, NLTK and Scikit Learn were imported.

We conducted the project by following these stepwise approaches:

2.1 Access the corpora

The corpora were loaded respectively and assigned to six dataframes by author. Each dataframe contains two columns, 'Text' and 'Author_Label'. The 'Text' column holds the book content, while the 'Author_Label' column stores the names of the authors.

2.2 Data cleaning and linguistic preprocessing

The original texts comprises misalignments or otherwise noises that would confuse the model training process.

This project mainly uses NLTK library to do the data cleaning and linguistic preprocessing, which include:

- ➢ Lowercasing
- ➢ Non-alphanumerics removal
- ➢ HTML syntax and URL removal
- ➢ Excessive white space compiling
- ➢ Lemmatization

It is worth mentioning that different from English, Portuguese language uses diacritics such as ç and ã. We did not remove any letters in 'ÁÉÍÓÚÀÂÊÔÃÕÜÇáéíóúàâêôãõüç'.

The data cleaning and linguistic preprocessing was done separately for each author's dataframe. Dataframes were not concatenated at this point because we would need to proceed the chunking for each author in the next step.

2.3 Break books into chunks

Since the the text excerpts which we need to predict an author have either 500 or 1,000 words, we decided to break the books into chunks of texts randomly between 450 and 1,050 words and train several models to receive text inputs with varying sizes. Each author's books were broken into different number of chunks without repetition through iterations. After the chunking, the order of the chunks in each dataframe by author were not shuffled, maintaining the maximum sequence between rows.

In the end, Jose Rodrigues Santos has the most chunks (1,563) while Luisa Marques Silva has the least (55).

2.4 Text re-cleaning

The data cleaning and linguistic preprocessing were repeated for each dataframe by author in this step because quotation marks, commas and brackets were generated during the chunking in the previous step.

To clarify, we did not process chunking first and then data cleaning and linguistic preprocessing for only once because the loaded corpora have a significant number of HTML tags and URLs. By doing so, the final number of words in some chunks could have been much fewer than 450 words after the data cleaning and preprocessing.

2.5 Undersampling for a balanced learning

The dataset is imbalanced as some authors have more text chunks than others. To better evaluate the model performance, oversampling was considered. However, we found it difficult to get extra books for the authors with fewer texts.

As a consequence, undersampling was eventually adopted and done by randomly dropping rows in each dataframe by author and keeping each dataframe to have the same number of rows. Six dataframes by author were concatenated into a single train dataframe for further model trainings at this point. The train dataframe contains 330 rows, with 55 for each author label.

2.6 Model development

This step aims to select a final model for prediction on the text excerpts in the test set by training different models. There are four main parts in this step:

2.6.1 Model evaluation metrics

- Accuracy
- Precision
- Recall
- F1 score
- Confusion matrix

For a balanced dataset, accuracy is the main metric, while other metrics are considered when two models have similar accuracy scores.

2.6.2 Feature engineering

BoW (Bag of Words) and TF-IDF are used to extract features from the text chunks.

For BoW, we used the CountVectorizer in Scikit Learn library to fit and transform all the texts in the train dataframe to be the training input X. In addition, we removed stop words, set the maximum document frequency to be 0.9, and applied unigrams and bigrams.

For TF-IDF, we used the TfidfVectorizer in Scikit Learn library to fit and transform all the texts in the train dataframe to be the training input X. Also, we did not remove any stop words but applied unigrams and bigrams.

Stop words in this project are the default Portuguese stop words in NLTK library.

2.6.3 Stratified Kfold cross validation

Stratified 5-fold cross validation without shuffle was assumed for train/test split to get X_train, y_train, X_val and y_val for four reasons:

- For small corpus with only hundreds of samples, cross validation is more appropriate than the train/development/test split.
- Stratified cross validation ensures that each fold contains the same proportions of all the class labels.
- With 5-fold, each fold holds 80% of the samples as training set and 20% as validation set.
- As the text order relates to different books, not shuffling the data mitigates overfitting and inflated accuracy scores which are caused by similarity regarding book content shared in both training set and validation set.

2.6.4 Training different models

- Kneighbors Classifier
- Multinomial Naive Bayes Classifier
- Logistic Regression Classifier

The three models above were trained for a total of six times. The idea was to try different models and/or feacture extraction methods to improve the baseline. The training process is as follows:

1) Extract features using BoW
2) Apply stratified Kfold cross validation split
3) Train with Kneighbors Classifier as Baseline (see Appendix 1.1 for parameter details)
4) Train with Multinomial Naive Bayes Classifier with default parameters
5) Train with Logistic Regression Classifier with default parameters
6) Extract features using TF-IDF
7) Apply stratified Kfold cross validation split
8) Train with Kneighbors Classifier (see Appendix 1.2 for parameter details)
9) Train with Multinomial Naive Bayes Classifier with default parameters
10) Train with Logistic Regression Classifier with default parameters

2.7 Model selection for prediction on text excerpts

The model for prediction on the text excerpts was selected according to the metrics in 2.6.1. The prediction process is as follows:

1) Access the text excerpts with 500 and 1,000 words in the test set and concatenate them into a single test dataframe
2) Data cleaning and linguistic preprocessing
3) Extract features with the selected feature extraction method by fitting and transforming the texts in the train dataframe (training input and output)
4) Extract features with the selected feature extraction method by transforming the text excepts in the test dataframe (prediction input)
5) Fit the training input and output to the selected model
6) Predict the author labels on the prediction input using the selected model

3 Results

This section summarises the key results during the model development process.

By applying BoW to extract word features as training inputs, the accuracy score of each model is as follows:

- ➢ Kneighbors Classifier (Baseline): 0.89
- ➢ Multinomial Naive Bayes Classifier: 0.98
- ➢ Logistic Regression Classifier: 0.97

By applying TF-IDF to extract word features as training inputs, the accuracy score of each model is as follows:

- ➢ Kneighbors Classifier: 0.76
- ➢ Multinomial Naive Bayes Classifier: 0.89
- ➢ Logistic Regression Classifier: 0.85

We find that the accuracy scores of Multinomial Naive Bayes Classifier and Logistic Regression Classifier are very close with the BoW model, and thus other metrics are analyzed.

The model evaluation report of the Multinomial Naive Bayes Classifier using Bow:

```
                      precision    recall  f1-score

    Almada Negreiros       1.00      1.00      1.00
Camilo Castelo Branco       1.00      1.00      1.00
      Eca De Queiros       1.00      1.00      1.00
Jose Rodrigues Santos       1.00      1.00      1.00
       Jose Saramago       0.92      1.00      0.96
  Luisa Marques Silva       1.00      0.91      0.95

            accuracy                           0.98
```

The model evaluation report of the Logistic Regression Classifier using BoW:

```
                      precision    recall  f1-score

    Almada Negreiros       1.00      1.00      1.00
Camilo Castelo Branco       1.00      1.00      1.00
      Eca De Queiros       1.00      1.00      1.00
Jose Rodrigues Santos       0.91      0.91      0.91
       Jose Saramago       1.00      1.00      1.00
  Luisa Marques Silva       0.91      0.91      0.91

            accuracy                           0.97
```

Examining the metrics, we can see that the Multinomial Naive Bayes Classifier slightly outperforms the Logistic Regression Classifier in this project. As a result, the BoW model and Multinomial Naive Bayes Classifier were selected to predict the authors of the text excerpts.

The predicted results:

Text name	Nm. of Words	Predicted_Author
text1.txt	500	Jose Saramago
text1.txt	1000	Jose Saramago
text2.txt	500	Almada Negreiros
text2.txt	1000	Almada Negreiros
text3.txt	500	Luisa Marques Silva
text3.txt	1000	Luisa Marques Silva
text4.txt	500	Eca De Queiros
text4.txt	1000	Eca De Queiros
text5.txt	500	Camilo Castelo Branco
text5.txt	1000	Camilo Castelo Branco
text6.txt	500	Jose Rodrigues Santos
text6.txt	1000	Jose Rodrigues Santos

4 Discussion / Conclusion

In conclusion, the Multinomial Naive Bayes Classifier performs better than the other two models for this project, probably because it requires less training data. This result also reinforces the common view that the Naive Bayes algorithm is best suited for text classification problems.

In addition, the fact that models using TF-IDF features have lower accuracy scores is surprising as TF-IDF vectorizer is often used to overcome the drawbacks of the BoW model. This extra finding could be a future research topic.

References

Coheur, L., Rei, R., & Guerreiro, N.M.(Eds.). 2020. *NLP in a Nut(s)shell: Readings & materials for learning NLP from the ground up*. Github. <https://ricardorei.github.io>

Li, Susan. 2018. *Multi-Class Text Classification Model Comparison and Selection*. Towards Data Science. <https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568>

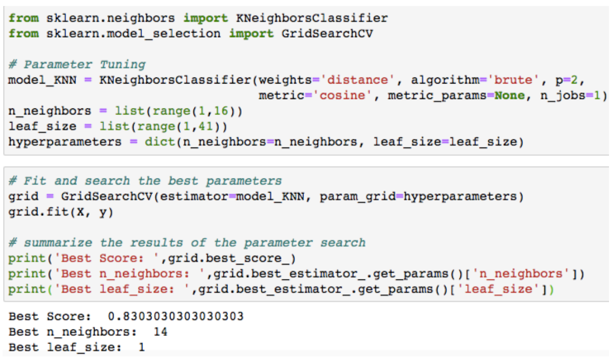
Brownlee, J. 2019. *How to Tune Algorithm Parameters with Scikit-Learn*. Machine Learning Mastery. <https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/>

Scikit-learn developers (BSD License). 2019. *3.1. Cross-validation: evaluating estimator performance Scikit-Learn.*<https://scikit-learn.org/stable/modules/cross_validation.html>

Appendices

Appendix 1: GridSearchCV of Scikit Learn library was used to tune two parameters in the Kneighbors Classifier: the number of neighbors and leaf size. This is because these parameters are usually difficult to define, especially the number of neighbors.

Appendix 1.1: Attached below is the parameters in the Kneighbors Classifier and the parameter tuning results for the two parameters in the Kneighbors Classifier using BoW as the Baseline.

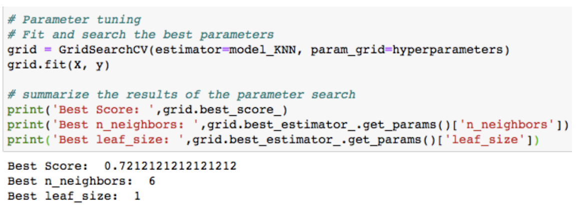


```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Parameter Tuning
model_KNN = KNeighborsClassifier(weights='distance', algorithm='brute', p=2,
                                 metric='cosine', metric_params=None, n_jobs=1)
n_neighbors = list(range(1,16))
leaf_size = list(range(1,41))
hyperparameters = dict(n_neighbors=n_neighbors, leaf_size=leaf_size)
```

```
# Fit and search the best parameters
grid = GridSearchCV(estimator=model_KNN, param_grid=hyperparameters)
grid.fit(X, y)

# summarize the results of the parameter search
print('Best Score: ',grid.best_score_)
print('Best n_neighbors: ',grid.best_estimator_.get_params()['n_neighbors'])
print('Best leaf_size: ',grid.best_estimator_.get_params()['leaf_size'])
```

```
Best Score:  0.8303030303030303
Best n_neighbors:  14
Best leaf_size:  1
```

Appendix 1.2: Attached below is the parameter tuning results for the two parameters in the Kneighbors Classifier using TF-IDF. Other parameters in this Kneighbors Classifier are the same as shown in Appendix 1.1.



```
# Parameter tuning
# Fit and search the best parameters
grid = GridSearchCV(estimator=model_KNN, param_grid=hyperparameters)
grid.fit(X, y)

# summarize the results of the parameter search
print('Best Score: ',grid.best_score_)
print('Best n_neighbors: ',grid.best_estimator_.get_params()['n_neighbors'])
print('Best leaf_size: ',grid.best_estimator_.get_params()['leaf_size'])
```

```
Best Score:  0.7212121212121212
Best n_neighbors:  6
Best leaf_size:  1
```