NLP - Author Attribution

Identifying authors through excerpts

Python3, BASH, RegEx, NLTK, and a lot of sklearn

Course: Text Mining
Nova IMS 2019/2020

Davide Montali M20190201
Francisco Cruz M20190637
Umberto Tammaro M20190806

1. Overview

We present an approach to identifying authors in an imbalanced setting using Python3, some BASH, and the NLTK and sklearn Python libraries. Alongside this report we submit a compressed folder containing the code and data used for this project. The data was provided by the instructors of the Text Mining course at Nova IMS. Our goal was to provide an accurate and lightweight model to predict the authors, instead of attempting to work with complex models and high computational resources, we attempted to solve the task as efficiently as possible. We placed particular emphasis on code reusability and execution time, constantly profiling our code to be as efficient as possible.

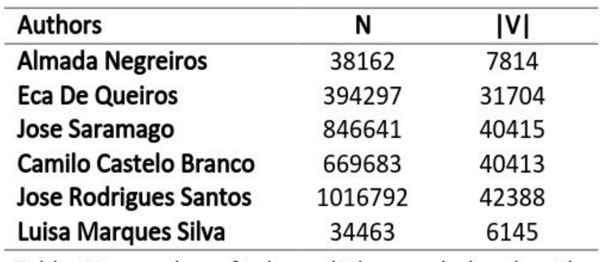
2. From Books to Corpora

Before starting to code, we explored the data using Bash, to get both an idea of the amount of data we were dealing with as well as some summary statistics. While we mostly used Bash as an exploratory tool, we also ended up including some Bash scripts in our final project (vastly quicker than python) . Namely, we transformed all the text into lowercase, and stripped away multiple spaces and rare characters – we then merged all documents of each author together to form a single corpora per author.

3. Preprocessing

Once we had our corpora in place, we made use of the NLTK stemmer and stopwords to normalise our data as well as reduce its dimensionality. We intentionally retain the punctuation given the varying use of punctuation by the authors. A further issue we noticed was the large imbalance between the data we had, for some authors we had over a million tokens, while others were below 40’000 tokens.

To balance our data we decided to split each author’s corpora into fragments of text, each 700 words long – we ran our models on a number of parameters to determine at what point we start converging (see Appendix). We also looked at undersampling and oversampling to balance out total words per author. Given that in an oversampling scenario we would have to generate data from the already scarce data we had, we decided to undersample data. Balancing the input data is particularly important for models which will calculate a prior, such as Naïve Bayes. We settled on randomly sampling 300 documents (of 700 words each) from each author, again we determined this by benchmarking our models and determining convergence (see Appendix).



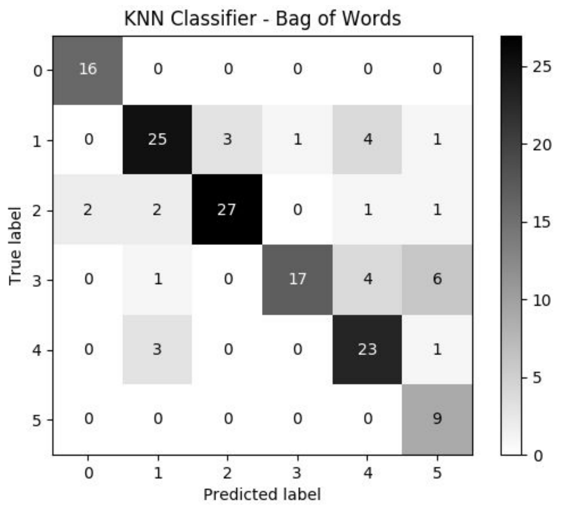
Authors	N	|V|
Almada Negreiros	38162	7814
Eca De Queiros	394297	31704
Jose Saramago	846641	40415
Camilo Castelo Branco	669683	40413
Jose Rodrigues Santos	1016792	42388
Luisa Marques Silva	34463	6145

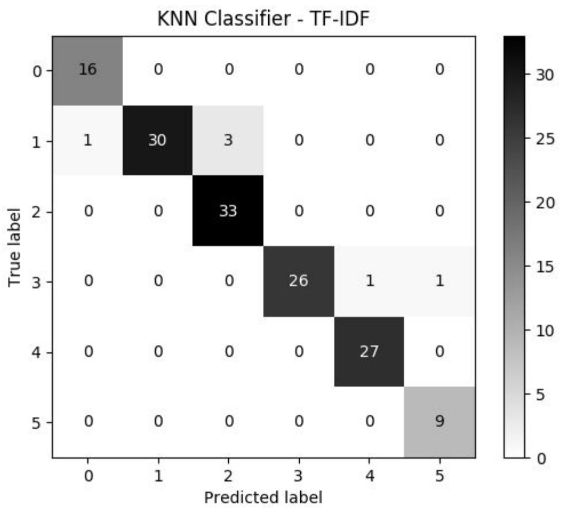
Table: N = number of tokens, |V| = vocabulary length.

We divide our data into a training set and a dev set - and then run 10 fold cross validation on our training set to evaluate performance, and ultimately run our model on the dev set.

4. Baseline Classifier

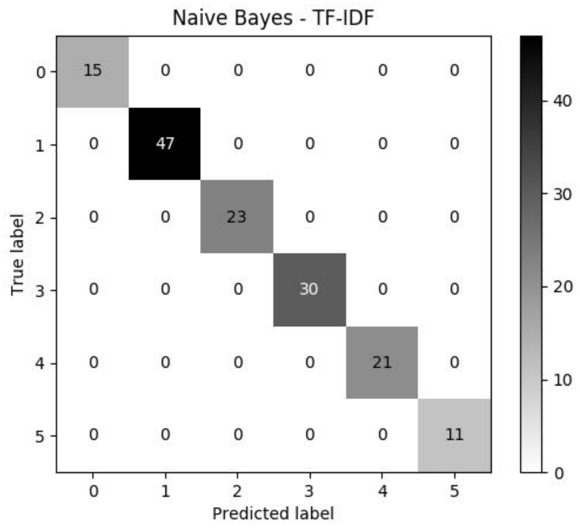
As a baseline model, we took the clean data and vectorised it with both into a bag of words, with CountVectorizer as well as using TF-IDF with the TfidfVectorizer – both provided by sklearn. We soley used unigrams, and already achieved a relatively high accuracy. With a 0.76 weighted average F1 score on the Bag of Words classifier, and 0.93 on the TF-IDF classifier.





5. Naive Bayes Classifier

Our second approach was to use a generative classifier, namely Naive Bayes. Still running on unigrams, and testing it on multiple randomly sampled parts of the data, we almost always achieved a weighted average F1 score of 1. The mean cross validation score with 10 folds, is 0.98. We believe this is also due to the fact that we benchmarked our models (see Appendix) to find good parameters.



We also set a smoothing parameter to the Naive Bayes classifier, to avoid causing zero probability for unessn events.

6. Linear SVM Classifier

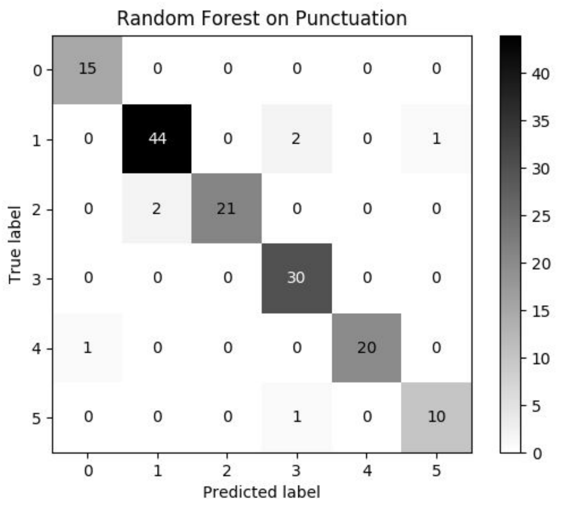
With a good performance on our generative classifier, we decided to also run a discriminative classifier, namely a Linear SVM. We retained the same parameters as for our Naive Bayes Classifier, but also added bigrams to the model. We achieved almost identical performance to the Naive Bayes Classifier, with all dev set data being predicted correctly.

7. An Elementary Model

Given the high accuracy our two models already have, following Occam's razor philosophy, we attempted to create a simpler model rather than a complex one. We attempted to predict texts by solely accounting for punctuation use of the authors.

To turn just punctuation into tokens we passed a custom RegEx string tokenizer to our tokenizer:
token_pattern=r"(?u)!+|\?+|\.+|,+|-+|:|;"

We included the “one or more” RegEx flag, to capture punctuation as it is in the text, for instance three dots are considered a single token rather than counting them as three dots. We then used a Random Forest Classifier to model the data based on entropy splits.



Our model dropped in accuracy into the 0.75 to 0.85 F1 score range. Yet, we believe this to be a decent prediction ability based on just punctuation use.

8. Text Predictions

For the unknown texts our model predicts the following (we submit the Naive Bayes predictions as our predictions):

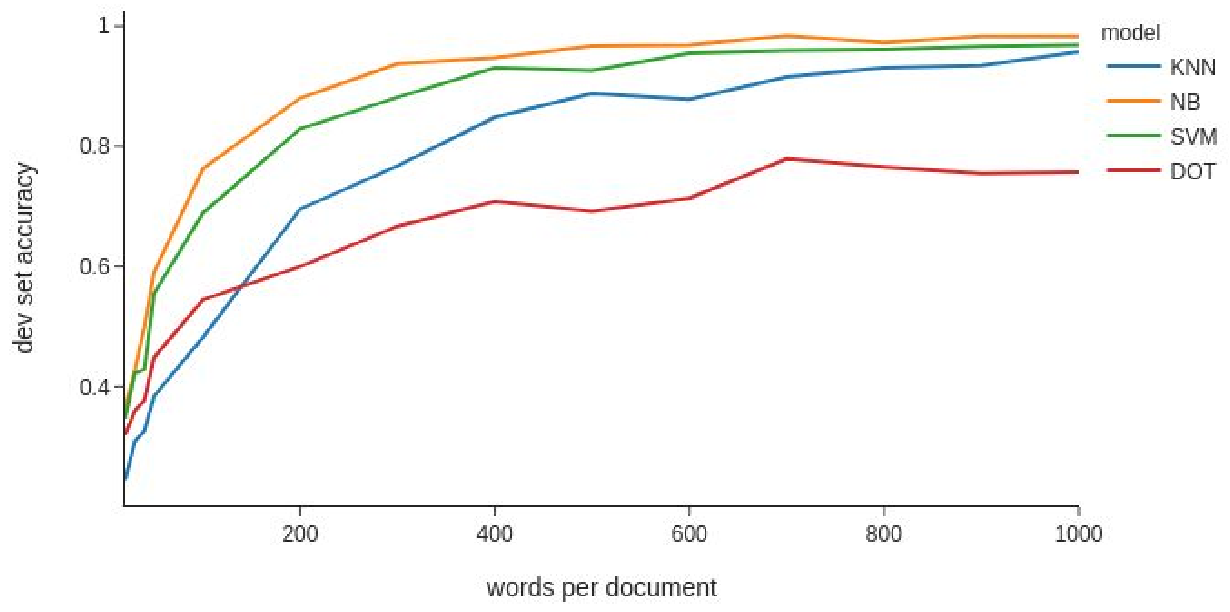
Documents	KNN	Naïve Bayes	SVM	Punct Forest
500Palavras/text1	JS	JS	JS	JS
500Palavras/text2	AN	AN	AN	JRS
500Palavras/text3	CCB	LMS	LMS	LMS
500Palavras/text4	EDQ	EDQ	EDQ	EDQ
500Palavras/text5	CCB	CCB	CCB	CCB
500Palavras/text6	JRS	JRS	JRS	JRS
1000Palavras/text1	JS	JS	JS	JS
1000Palavras/text2	AN	AN	AN	AN
1000Palavras/text3	JS	LMS	LMS	LMS
1000Palavras/text4	EDQ	EDQ	EDQ	EDQ
1000Palavras/text5	CCB	CCB	CCB	JS
1000Palavras/text6	JRS	JRS	JRS	JRS

9. Conclusion

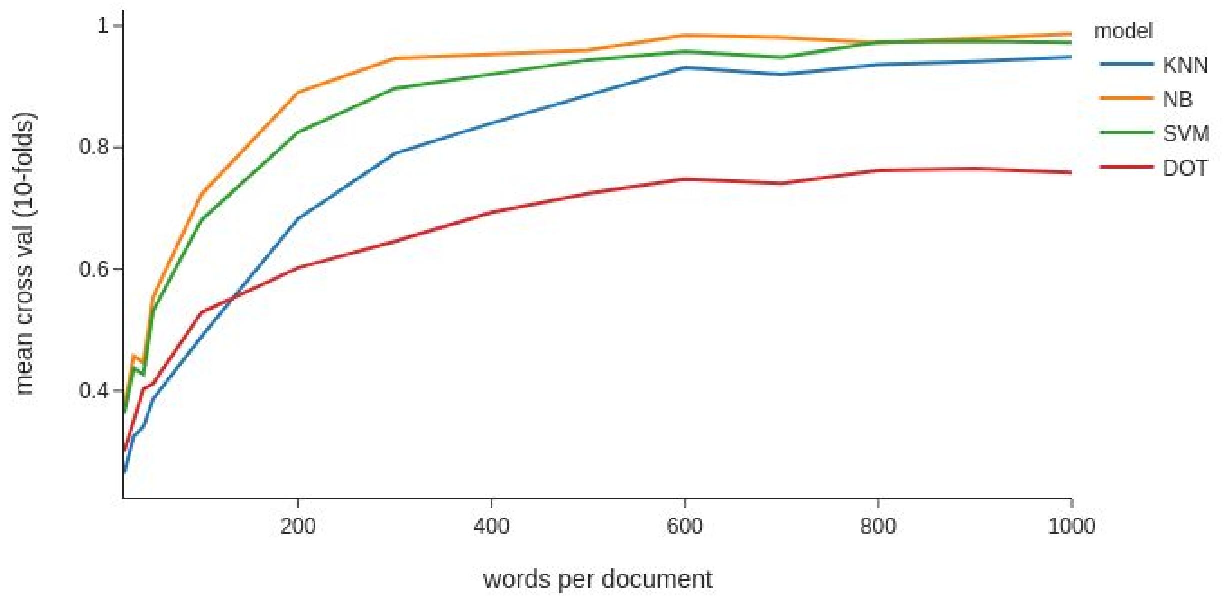
Given the task was to identify authors of exterpts, we made assumptions with that specific task in mind. With infinite time at our disposal, we would have explored the option of oversampling from the minority authors - building a model which could be extended to other authors and potentially be able to generalize better. Yet, for the task at hand we’re confident our models are good - and are surprised by the accuracy which can be achieved by only accounting for punctuation.

Appendix

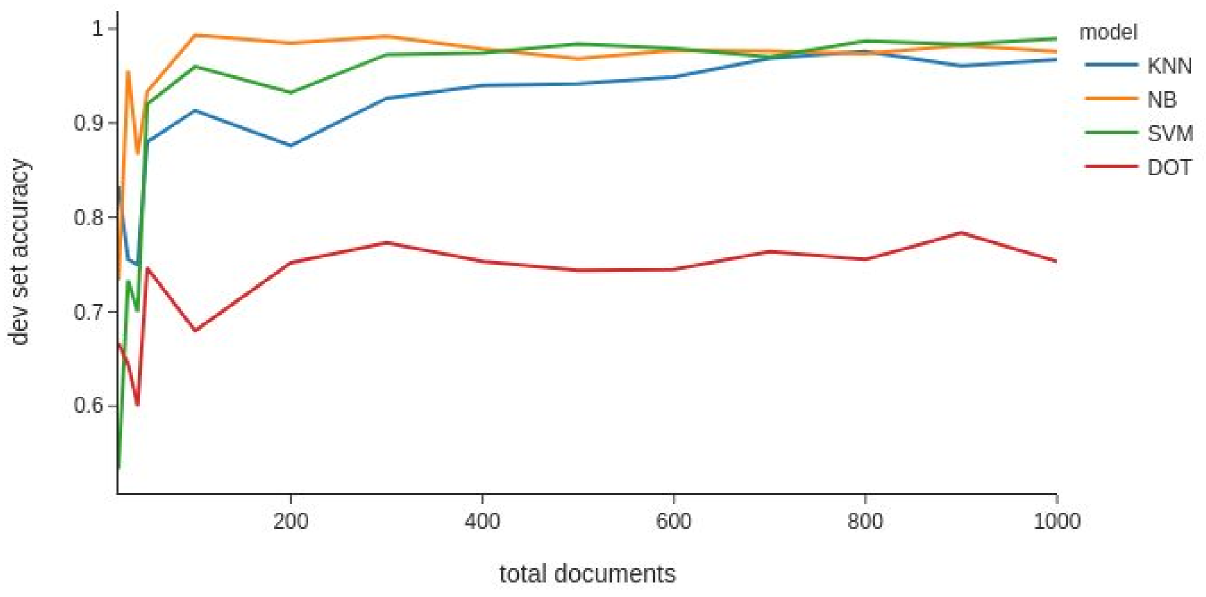
Dev Set Accuracy by Words per Document



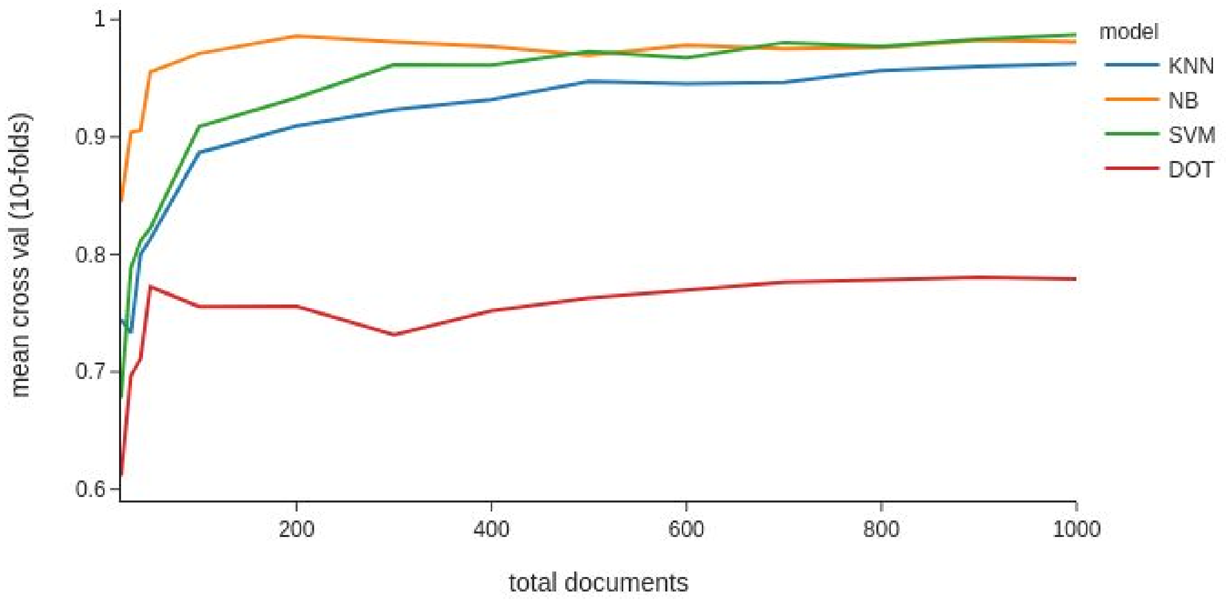
Mean Crossvalidation Score (10 fold) by Words per Document



Dev Set Accuracy by Number of Documents (700 words per document)



Mean Crossvalidation Score (10 fold) by Number of Documents (700 words per documer


mean cross val (10-folds)
total documents
model
KNN
NB
SVM
DOT