

# Machine Learning Engineer Nanodegree

## Capstone Project

Thiago Vieira June 7th, 2019

### I. Definition

#### Project Overview

The goal of this project it's to build a model for authorship attribution classification using as a background the work developed in [VJO2011](#) e [OOJ2013](#) that is available in [The Laboratory of Vision, Robotics and Imaging of Federal University of Parana](#).

The use of electronic documents like e-mails continue to grow exponentially, and even though reliable technology is available to trace a particular computer/or IP address where the document has been produced, the fundamental problem is to identify who was behind the keyboard when the document was created (OOJ2013). Practical applications for author identification have grown in several different areas such as criminal law (identifying writers of ransom notes and harassment letters), civil law (copyright and estate disputes), and computer security (mining e-mail content).

This problem is very relevant for my work since I'm developing several NLP classification models to label court decisions and the importance of a case to the federal attorney. I intend to apply the knowledge obtain from this project to my work and share it, as well.

#### Problem Statement

Authorship attribution can be defined as the task of inferring characteristics of a document's author from the textual attributes of the document itself. The challenge here is to estimate how similar two documents are from each other, based on patterns of linguistic behavior in documents of known and unknown authorship. This is known in the literature as authorship attribution or authorship analysis.

#### Metrics

Based on the context of NLP and the multiclass problem property, I'll use the following metrics to compare models:

- Accuracy [link](#);
- F1 score (trade-off between TP and FP) [link](#);
- Recall [link](#);
- Precision [link](#);
- Confusion Matrix [link](#).

Given the context of the problem and to replicate the same metrics used in the base references, These set of metrics is suitable to the problem because it's the most used in the NLP field.

### II. Analysis

#### Data Exploration

The dataset used and proposed by [VJO2011](#) contains 100 different authors whose texts were uniformly distributed over 10 different subjects: Miscellaneous, Law, Economics, Sports, Gastronomy, Literature, Politics, Health, Technology, and Tourism. In this database, all the subjects have ten different authors.

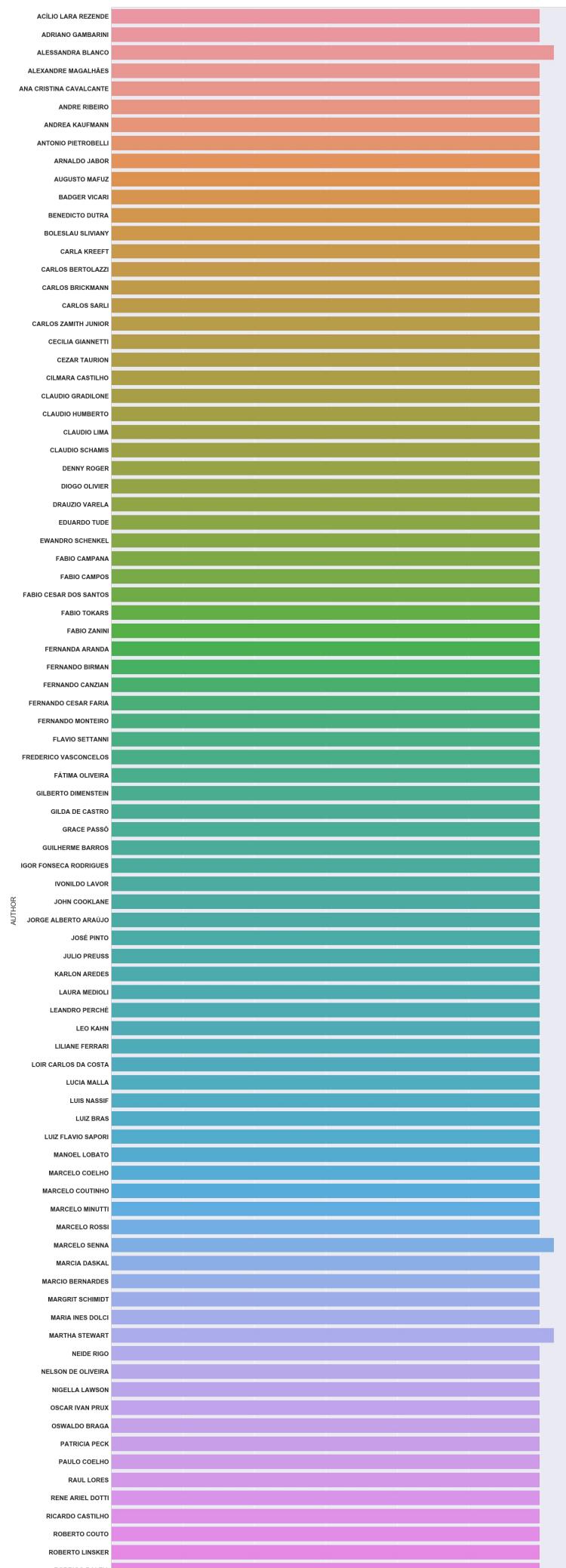
For each author it was chosen 30 short articles, thus summing up 3000 pieces of documents. The articles usually deal with polemic subjects and express the author's personal opinion. In average, the articles have 600 tokens (words) and 350 Hapax (words occurring once). One aspect worth of remark is that this kind of articles can go through some revision process, which can remove some personal characteristics of the texts. Besides, authorship attribution using short articles poses an extra challenge since the number of features that can be extracted is directly related to the size of the text.

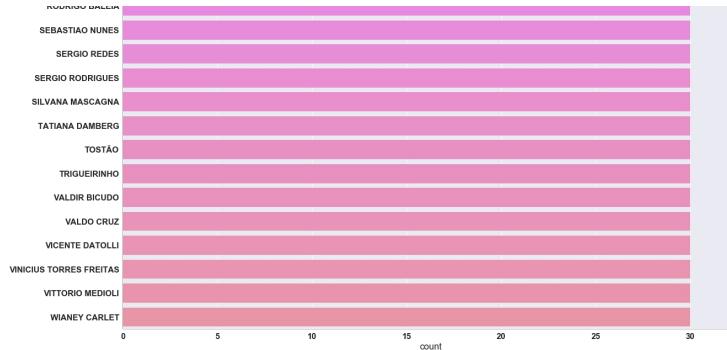
At first, it created a CSV file to store all the text from the initial dataset provided by the original work. The dataset was distributed in folders organized by subjects and authors in folder data/raw/BASE DE DADOS - PAULO JR VARELA. In notebook notebooks/00-Make-Dataset.ipynb this CSV file is built and stored at folder data as data\_raw.csv.

#### Exploratory Visualization

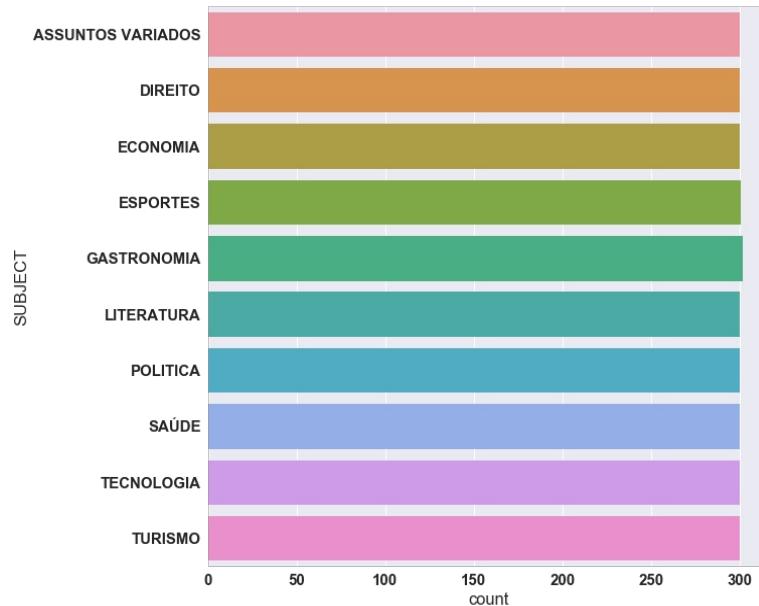
It was verified that the authors and subject distributions were by the reported by the original work as it's shown in the figures below:

- Authors





- Subjects

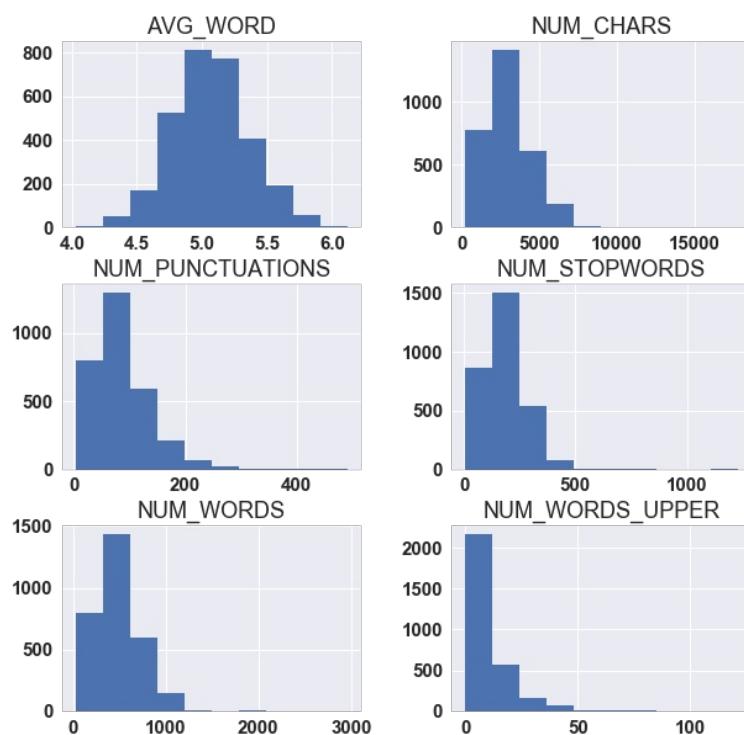


## New Features

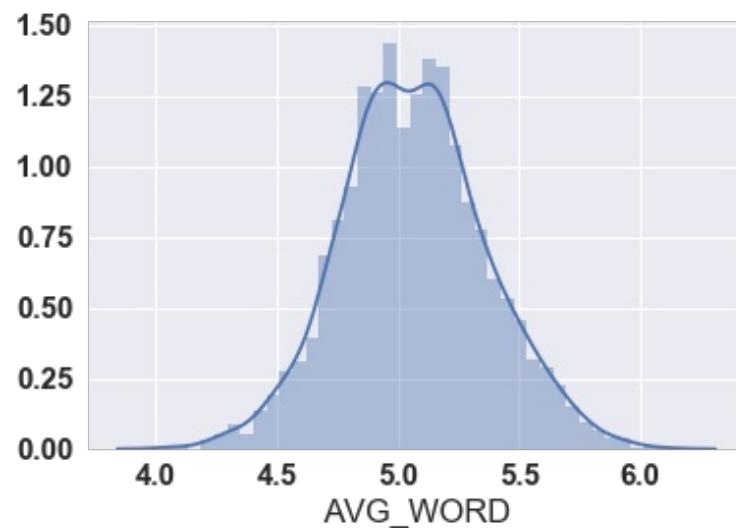
Then in notebook notebooks/01-Make-Features.ipynb I created new features based on the text characteristics as Number of stopwords, Number of punctuations, Number of title case words, Number of chars, Number of words and Average word length. In this notebook, the text is also cleaned, and a new CSV file containing all these transformations is created and stored in /data/data\_feat.csv.

Some Exploratory Data Analysis is made in notebooks/02-EDA.ipynb over the new features and text data.

By the new created features, their distributions are as follow:



It's important to notice the normal shape distribution of average word length:



## WordCloud

Another visual analysis applied was a technique to better visualize the most appeared words in the text. First, I generate a wordcloud using all cleaned text and I obtained the following figure:



We can use this analysis to find and remove potential no relevant words to add to a stopword list.

Then, I generate the word cloud for each of the subjects since there are many authors. In this approach, the word cloud was generated based on the most relevant words in each subject by using TF-IDF feature extractions technique:

- Assuntos Variados



- Esporte



- Saúde



- Gastronomia



- Economia



- Direito





## Algorithms and Techniques

Several algorithms and techniques were used and applied based on the literature on NLP classification problems. Most of them were used without any hyperparameter tuning.

For the classical machine learning techniques tested like Logistic Regression, Stochastic Gradient Descent and Multi-Layer Perceptron the input text was preprocessed using TF-IDF and for more advanced techniques like Deep Neural Networks, Convolutional Neural Networks and Recurrent Neural Networks it was also used word embeddings ([link](https://en.wikipedia.org/wiki/Word_embedding))[[https://en.wikipedia.org/wiki/Word\\_embedding](https://en.wikipedia.org/wiki/Word_embedding)].

Basically, I tested the following classification algorithms and feature representation:

- Logistic Regression;
- Stochastic Gradient Descent;
- Multi-Layer Perceptron;
- Shallow Neural Network;
- Deep Neural Network;
- Convolutional Neural Network;
- Recurrent Neural Network;
- FastText.

## Benchmark

In the image below, it's shown some works on authorship attribution published in the literature. Comparing different works is not a straightforward task since most of the works use different databases and classifiers.

Published works on authorship attribution.

Ref.	Classifier	Database	Rec. rate (%)
[23]	SVM	Web pages	66–80
[7]	SVM	German newspaper	80
[10]	SVM	3 sister's letters	75
[24]	kNN	Novels	66–76
[5]	Distance	Brazilian novels	78
[19]	SVM	Brazilian newspaper	72
[6]	Bayes	Mexican poems	60–80
[21]	Bayes	Turkish newspaper	80
[25]	SVM	Brazilian newspaper	74

For this project, I'll use as a benchmark model the accuracy of the work in [VJO2011](#) e [OOJ2013](#), which were a 74% and 77% using with an SVM classifier + a feature selection using a multi-objective algorithm and compression models approach, respectively.

Since there are 100 authors in the database, analyzing the confusion matrix would be complicated, these works provided the analysis of the confusion matrix grouped by subject. Such a matrix can show that the recognition rate in terms of subjects is about 86% in [VJO2011](#) and 80% in [OOJ2013](#).

## III. Methodology

### Data Preprocessing

The text data was cleaning in notebooks/01-Make-Features.ipynb where I removed bad characteres, stopwords, and create new features.

Other two data preprocessing technique were used to prepare the text for the models. They are TF-IDF and [Glove](#) word embeddings.

### Implementation

In VJO2011 experiments they used 7 documents for training and the remaining 23 for testing. In order to be able to compare the results, we adopted the same protocol. The documents were randomly divided into training and testing.

Let's summarize all the implementation done in this project in parts.

## Feature Extractions

- It was implemented some function to new features in notebooks/01-Make-Features.ipynb;

## Visualization

- It was implemented some function to visualize data in notebooks/02-EDA.ipynb, like generate\_wordcloud, top\_tfidf\_feats, top\_feats\_in\_doc, top\_mean\_feats and top\_feats\_by\_class used to show relevant words in wordcloud figures;

## Classic Machine Learning

- It was implemented some function to train the model and show results in notebooks/03-ML.ipynb, like conf\_matrix, train, evaluate\_, train\_evaluate, show\_roc, show\_report used to better organized the experimentation process;

## Neural Networks

Here, the models were implemented using the framework Keras.

- It was implemented some function to train the model and show results in notebooks/04-ML.ipynb - notebooks/05-DNN.ipynb - notebooks/06-CNN.ipynb - notebooks/07-RNN.ipynb - notebooks/04-ML.ipynb, like plot\_history, Metrics, evaluate\_, train\_evaluate, show\_roc, show\_report used to better organized the experimentation process and add metrics to traning process;

## Word Embeddings

Here, some implementations used the Glove Word Embeddings downloaded from NILC. More precisely, the GLOVE 100 that can be downloaded in this link.

## Refinement

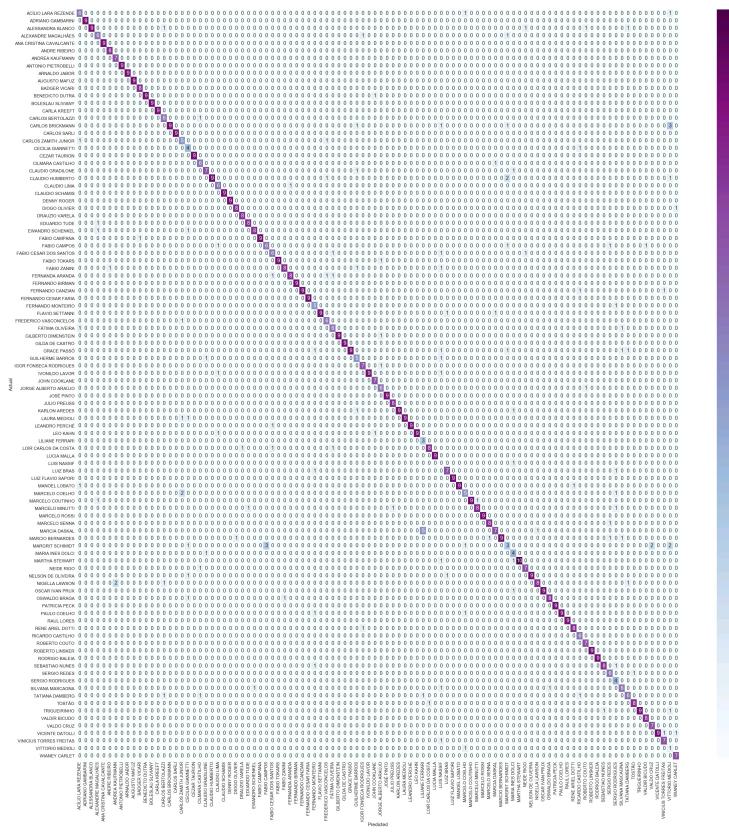
The best classical algorithms was applied to a GridSearch with cross validation to find a better hyperparameter setting as shown in notebooks/03-ML.ipynb.

This process resulted in the best model in this project as shown below:

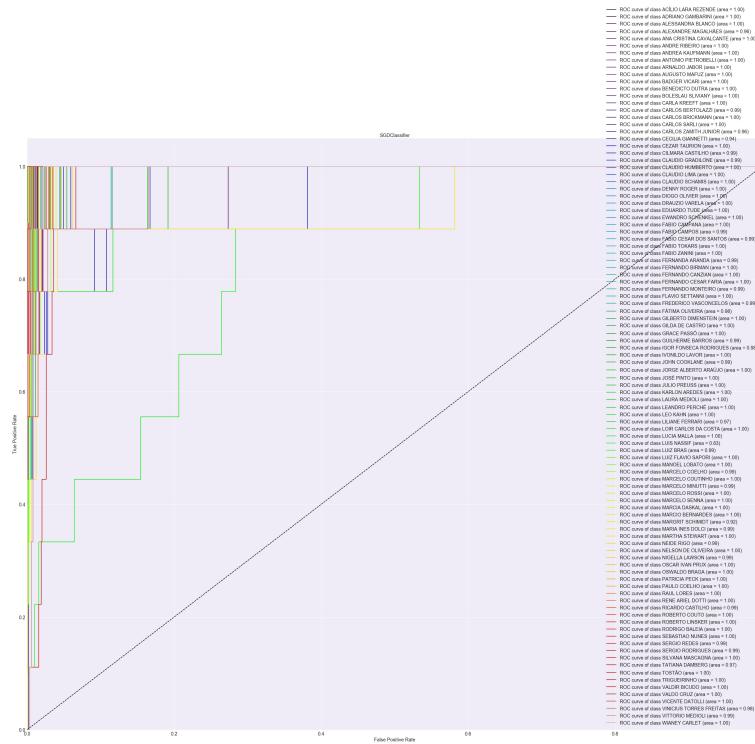
```
Train acc: 1.0
Test acc: 0.8423973362930077
Train f1-score: 1.0
Text f1-score: 0.8423973362930077
```

The classification report over all author gave an average of precision, recall and f1-score of 0.85, 0.84 and 0.83, respectively over the 901 test samples.

- Confusion Matrix



## • ROC CURVE



Then I added the average word length feature to the model, as it's follow a normal distribution. But, it made the accuracy decrease:

```
Train acc: 0.9895337773549001
Test acc: 0.7280799112097669
Train f1-score: 0.9895337773549001
Text f1-score: 0.7280799112097669
```

## IV. Results

### Model Evaluation and Validation

Let's, again, summarize all the models evaluations done in this project in parts. I'm not going to show all tests on all models since many of them resulted in poor results, but I'll show the best of each approach.

As mentioned before, the best result was:

```
Train acc: 1.0
Test acc: 0.8423973362930077
Train f1-score: 1.0
Text f1-score: 0.8423973362930077
```

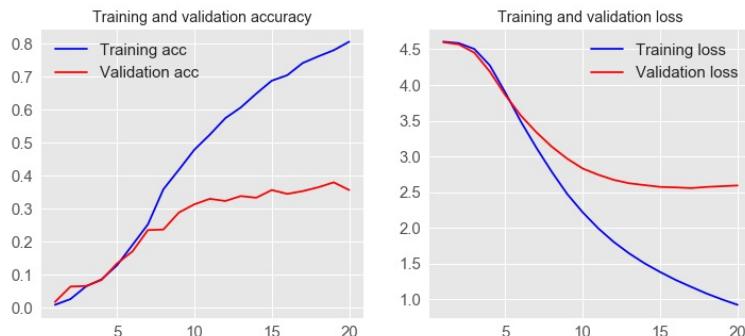
## Shallow Neural Network Evaluations - notebooks/04-NN.ipynb

Better result was with word embedding and maxpooling:

```
embedding_dim = 50

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(100, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['sparse_categorical_accuracy','accuracy'])
```

```
Training Accuracy: 0.8518
Testing Accuracy: 0.3561
```



## Deep Neural Network Evaluations - notebooks/05-DNN.ipynb

Better result was with TF-IDF and Dropout:

```
input_dim = 600
model = Sequential()
model.add(layers.Dense(10, input_dim=input_dim, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(100, activation='softmax'))
```

```
Training Accuracy: 0.9713
Testing Accuracy: 0.5441
```



## Convolutional Neural Network Evaluations - notebooks/06-CNN.ipynb

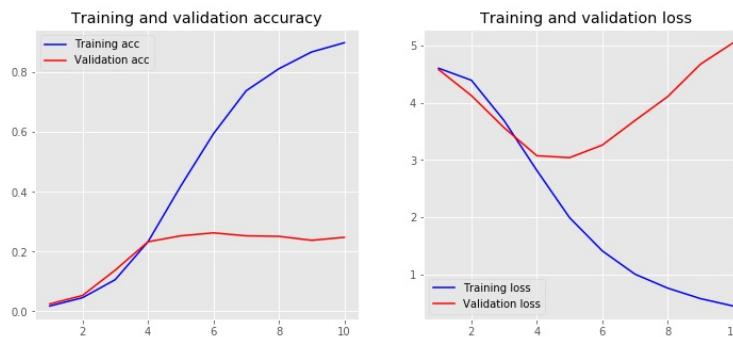
Better result was with word embedding:

```
embedding_dim = 100

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(100, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Training Accuracy: 0.9009

Testing Accuracy: 0.2479



## Recurrent Neural Network Evaluations - notebooks/07-RNN.ipynb

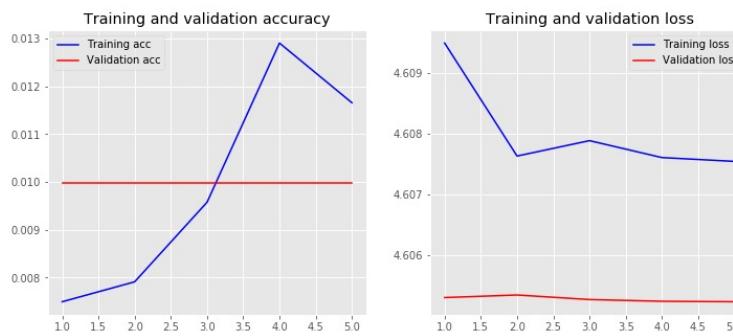
Better result was with word embedding and LSTM, but probably it has some problem because the classification was very bad:

```
embedding_dim = 100

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.SpatialDropout1D(0.3))
model.add(layers.LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(layers.Dense(100, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Training Accuracy: 0.0121

Testing Accuracy: 0.0100



## FastText Evaluations - notebooks/08-FastText.ipynb

Using supervised learning it obtained a accuracy of 0.324. Using n-grams did not improve the score.

As we can see, the best model was the one with a Stochastic Gradient Descent classifier with log loss. But, we can observe that it is overfitted. Another result, also with Stochastic Gradient Descent but using hinge loss, that can not provide probabilities, did a good job but with the cost of worst accuracy and better recall.

## Justification

Looking only for model accuracy, my model did a better job of predicting the text author using the same sample split. As the original work did not provide f1-score or the seed used for the train/test split, I can't be sure that my result is really better.

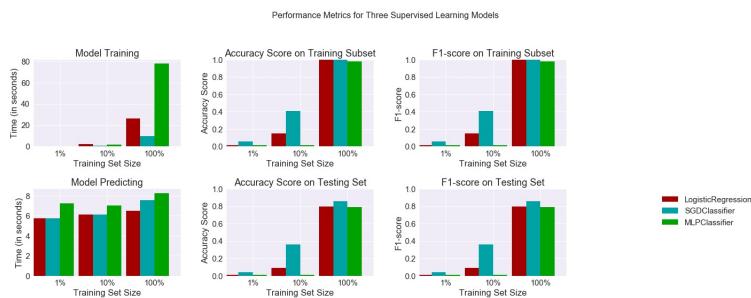
My best accuracy was 0.84 on the overfitted model and 0.72 in a less overfitted model. The original result was 0.74.

The solution presented is more like a baseline for future work than a proper solution. As the original work was more intended to show a new approach than to provide a solution to the problem since the dataset is tiny.

## V. Conclusion

### Free-Form Visualization

Here I'd like to plot the graph of the model selection that I borrowed from the course, and it showed very usefully. I'd like to make this graph with all models built, but it was more difficult than I imagine. This plot helped me to find a better classical machine learning algorithm for the problem and to analyze it in a much faster way.



### Reflection

The goal of this project it's to build a model for authorship attribution classification using as a background the work developed in [VJO2011 e OJ2013](#) that is available in [The Laboratory of Vision, Robotics and Imaging of Federal University of Parana](#).

During the project, I have done the following:

- Collect the data;
- Create the dataset;
- Cleaned the data;
- Create new features;
- Did some EDA and visualizations analysis;
- Processed the text data with TF-IDF and Word Embeddings;
- Built classical machine learning models;
- Built Deep Neural Networks models;
- Preprococed the text to follow FastText schema;
- Tested a SOTA text classification framework, [FastText](#).

The most challenge thing about this project was to learn the basics [Keras](#) and [FastText](#), so I could use them. The other tricky thing was the lack of data, this dataset is tiny.

As my final thoughts, I think the more straightforward approach resulted in a satisfactory solution to the problem, and it was better than the benchmark result.

### Improvement

I think that the best model could be improved by building an ensemble model to balanced the miss-classified authors. It could be made a more in-depth analysis of the most miss-classified text to seek insights, like words that could be generating these errors.

It's necessary to double check some deep neural networks embeddings preprocessing because some model failed terribly and maybe this could be the reason.

Also, I could build a topic model using LDA - Latent Dirichlet allocation and use it as a feature to improve classification performance.

Another final improvement is to gather more data by these same authors.