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.

The problem consist of given a text from some news article, classify it between 100 authors. As input, our classifier receives the text (in a certain format depending on the classification algorithm) and as output it gives the probability of been from certain author for all 100 candidates.

It's important to mentions that it's crucial to use algorithms that output probabilities for a better understanding and interpretability of the model. Especially, because of the metrics that's going to be analyzed.

Metrics

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

- Accuracy link;

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

- F1 score (trade-off between TP and FP) link;

$$f1score = 2 * \frac{(precision * recall)}{(precision + recall)}$$

- Recall link;

$$recall = \frac{TP}{(TP + FN)}$$

Given:

- TP - True positives
- FP - False positives
- FN - False negatives

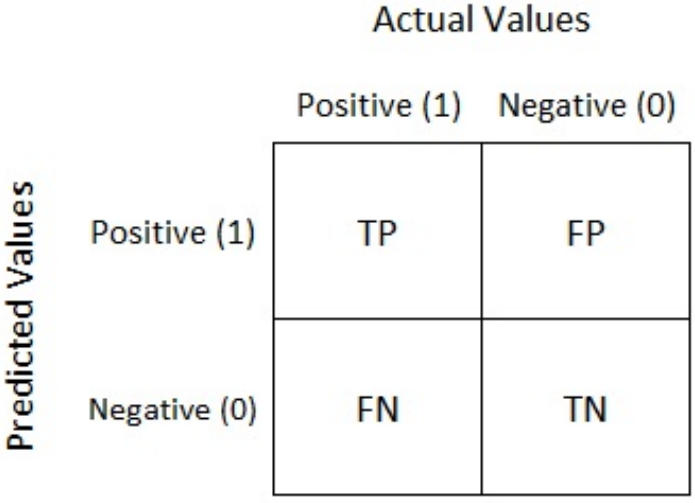
- Precision link;

$$precision = \frac{TP}{(TP + FP)}$$

Given:

- TP - True positives
- FP - False positives
- FN - False negatives

- 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.

For some models I could get all these metrics, but not all of them. Neverless, for comparison to the original word I only used the accuracy

metrics because it's the only one provided by them.

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.

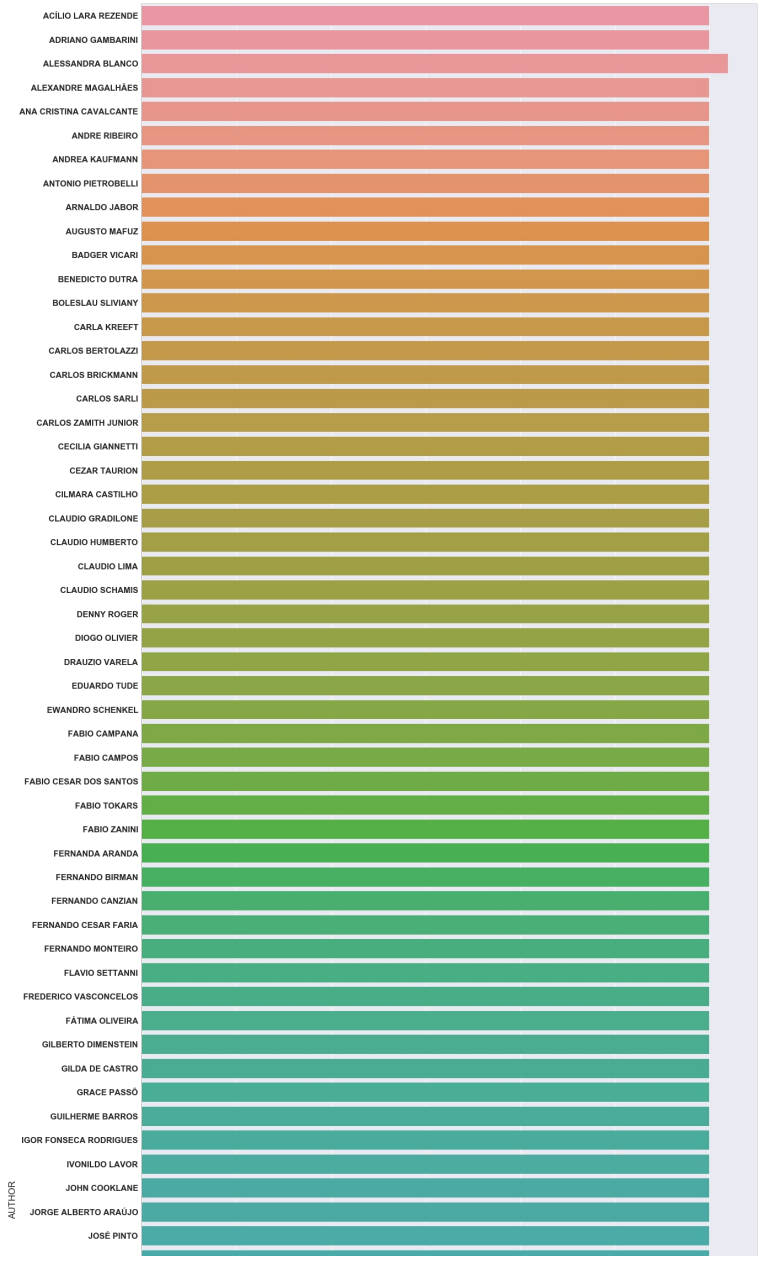
At first, it created a CSV file to store all the text from the initial dataset provided by the original work. The dataset was distributed in folders organized by subjects and authors in folder `data/raw/BASE DE DADOS - PAULO JR VARELA`. In notebook `notebooks/00-Make-Dataset.ipynb` this CSV file is built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

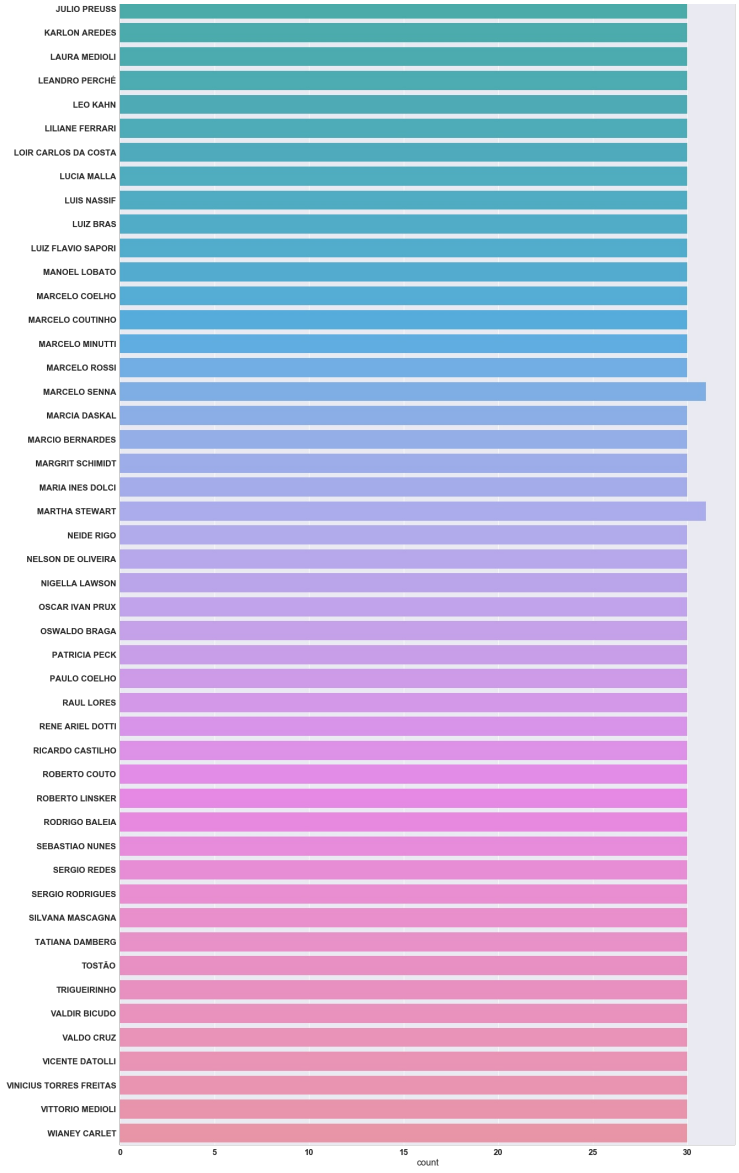
This dataset is very

Exploratory Visualization

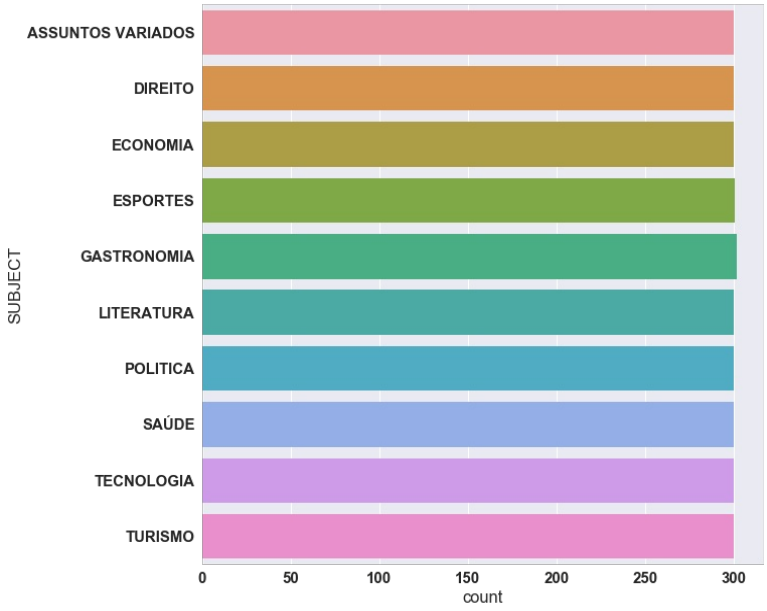
It was verified that the authors and subject distributions were by the reported by the original work as it's shown in the figures below:

- Authors





- Subjects

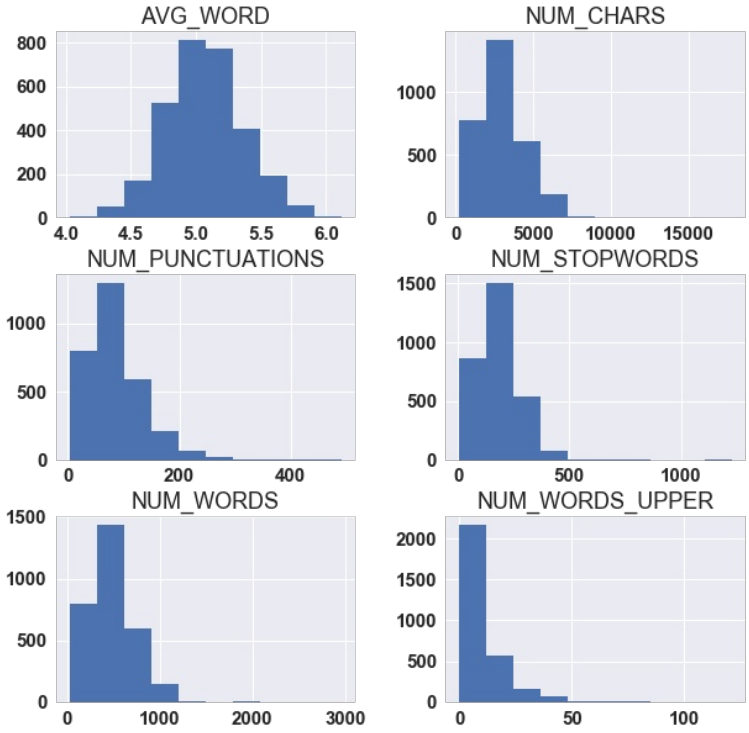


New Features

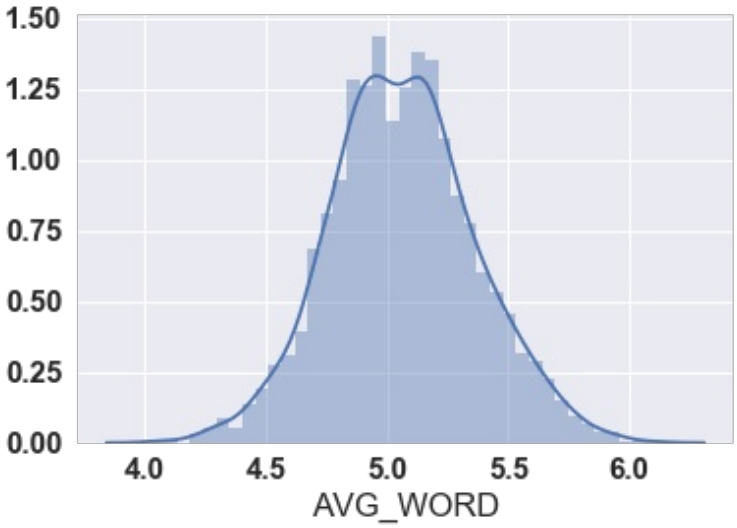
Then in notebook notebooks/01-Make-Features.ipynb I created new features based on the text characteristics as Number of stopwords, Number of punctuations, Number of title case words, Number of chars, Number of words and Average word length. In this notebook, the text is also cleaned, and a new CSV file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

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

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



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



WordCloud

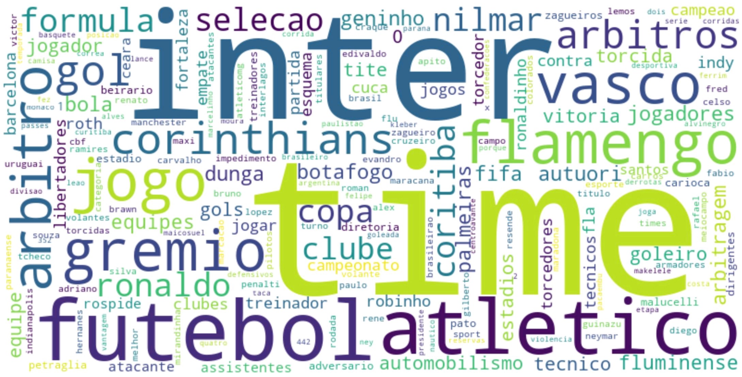
Another visual analysis applied was a technique to better visualize the most appeared words in the text. First, I generate a wordcloud using all cleaned text and I obtained the following figure:

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

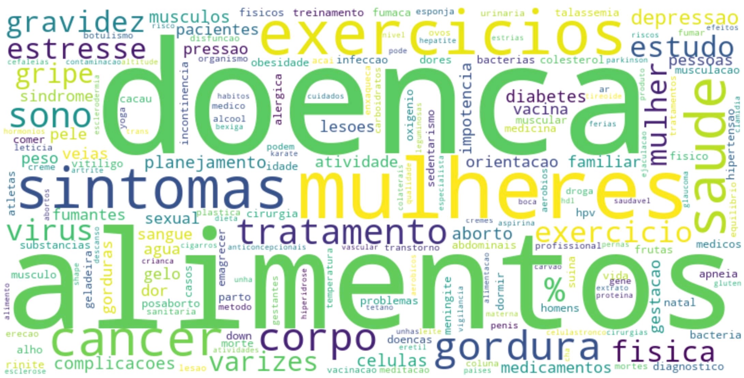
Then, I generate the word cloud for each of the subjects since there are many authors. In this approach, the word cloud was generated based on the most relevant words in each subject by using TF-IDF feature extractions technique:

- Assuntos Variados

- Esporte

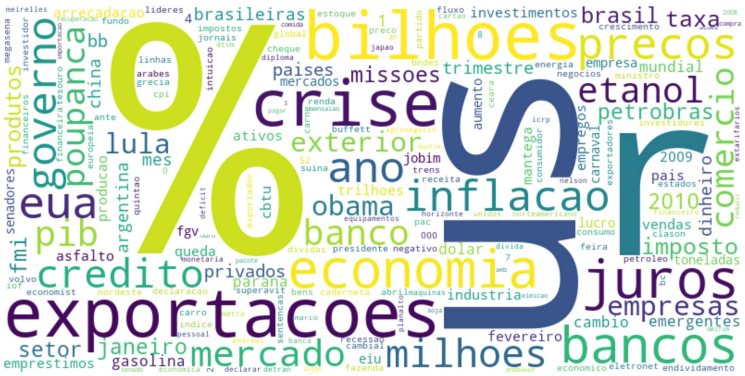


- Saúde



- Gastronomia

- Economia



- Direito

- Literatura

- Política

- Tecnologia

- Turismo

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 tunning.

For the classical machine learning techniques tested like Logistic Regression, Stochastic Gradient Descendent 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].

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

- Logistic Regression:

- Logistic Regression is very widely used in the case of binary classification problems in many fields like NLP and simple linear problems. LR in NLP

Strengths of the model: Logistic Regression

- It is very efficient;
- It's highly interpretable;
- Can be implemented relatively easy and quick.

Weaknesses of the model: Logistic Regression

- Can't solve non-linear problems;
- Will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other.

Logistic Regression is a type of classification algorithm. The output of a Logistic Regression model is a probability that the given input point belongs to a certain class. The central premise of Logistic Regression is the assumption that your input space can be separated into two nice 'regions', one for each class, by a linear(read: straight) boundary.

- Stochastic Gradient Descendent:

- Used for NLP tasks, because works well with sparse data as a Logistic Regression; Sentiment Analysis

Strengths of the model: Sklearn SGD

- It is efficient and can get results as good as a Logistic Regression;
- It is easy to implement and provides a lot of opportunities for code tuning.

Weaknesses of the model: Sklearn SGD

- Can be difficult to tune hyperparameters;
- It's sensitive to feature scaling, so data should be normalized.
- Multi-Layer Perceptron,link
- Shallow Neural Network:

These type of neural network are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer. They are suitable for classification prediction problems where inputs are assigned a class or label.

- Deep Neural Network: link

This type of neural network are comprised of multiple and big layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.

- Convolutional Neural Network: link

Convolutional Neural Networks, or CNNs, were designed to map image data to an output variable. The CNN input is traditionally two-dimensional, a field or matrix, but can also be changed to be one-dimensional, allowing it to develop an internal representation of a one-dimensional sequence.

Although not specifically developed for non-image data, CNNs achieve state-of-the-art results on problems such as document classification used in sentiment analysis and related problems.

- Recurrent Neural Network: link

Recurrent Neural Networks, or RNNs, were designed to work with sequence prediction problems. Sequence prediction problems come in many forms and are best described by the types of inputs and outputs supported.

The Long Short-Term Memory, or LSTM, network is perhaps the most successful RNN because it overcomes the problems of training a recurrent network and in turn has been used on a wide range of applications.

RNNs and LSTMs have been tested on time series forecasting problems, but the results have been poor, to say the least. Autoregression methods, even linear methods often perform much better. LSTMs are often outperformed by simple MLPs applied on the same data.

- FastText:

FastText is a library for efficient learning of word representations and sentence classification. It enable efficient learning of word representations and sentence classification. It is written in C++ and supports multiprocessing during training. FastText allows you to train supervised and unsupervised representations of words and sentences. These representations (embeddings) can be used for numerous applications from data compression, as features into additional models, for candidate selection, or as initializers for transfer learning.

- Word Embeddings

Word representations and sentence classification are fundamental to the field of Natural Language Processing (NLP). Word representation treats a corpus of text as an indiscriminate bag of words, aka, BoW. BoW cares about only 2 things: a) the list of known words; b) tally of word frequency. It does not care about the syntactic (structure) and semantic (meaning) relationships among the words in the sentence, sentences in the text. For some purposes, this treatment is good enough, for example, spam detection. For others, BoW falls short. Instead, word vectors becomes the state-of-the-art form of word representation.

Word vectors represent words as multidimensional continuous floating point numbers where semantically similar words are mapped to proximate points in geometric space. In simpler terms, a word vector is a row of real valued numbers (as opposed to dummy numbers) where each point captures a dimension of the word’s meaning and where semantically similar words have similar vectors.Word Embeddings.

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.

Othertwo 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. It was a challenge to create the class `Metrics` to try to track f1-score, but didn't work for every neural network.

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. This approach was difficult because of the size of the file and the preprocessing phase that was needed. By the end, didn't seems to be a good attempt.

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.

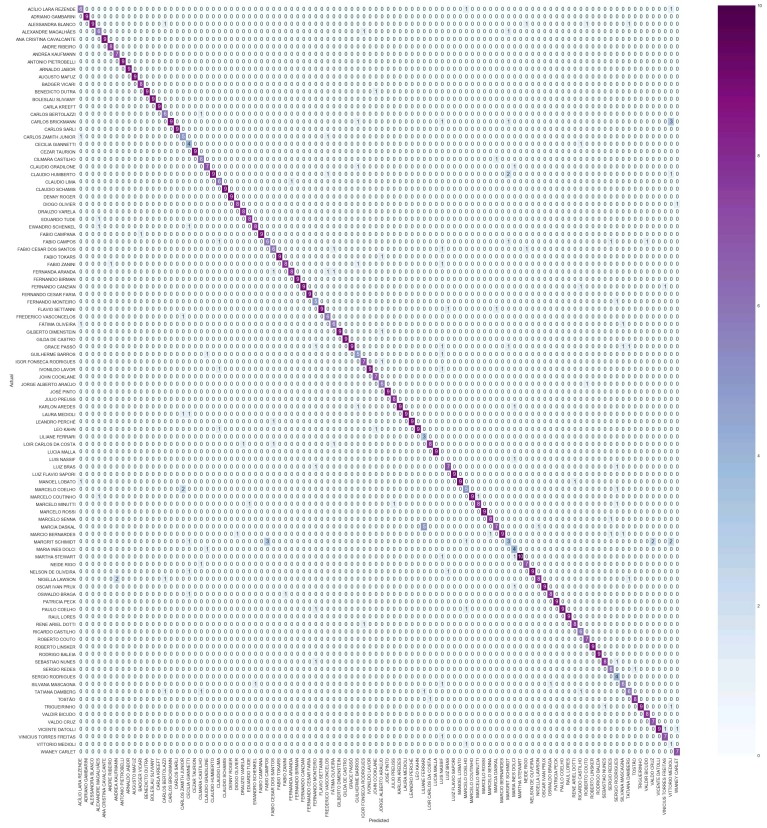
```
# the parameters list.
parameters = {
    'clf__loss': ['log', 'hinge', 'modified_huber'],
    'clf__random_state': [42],
    'clf__n_jobs': [-1]
}
```

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

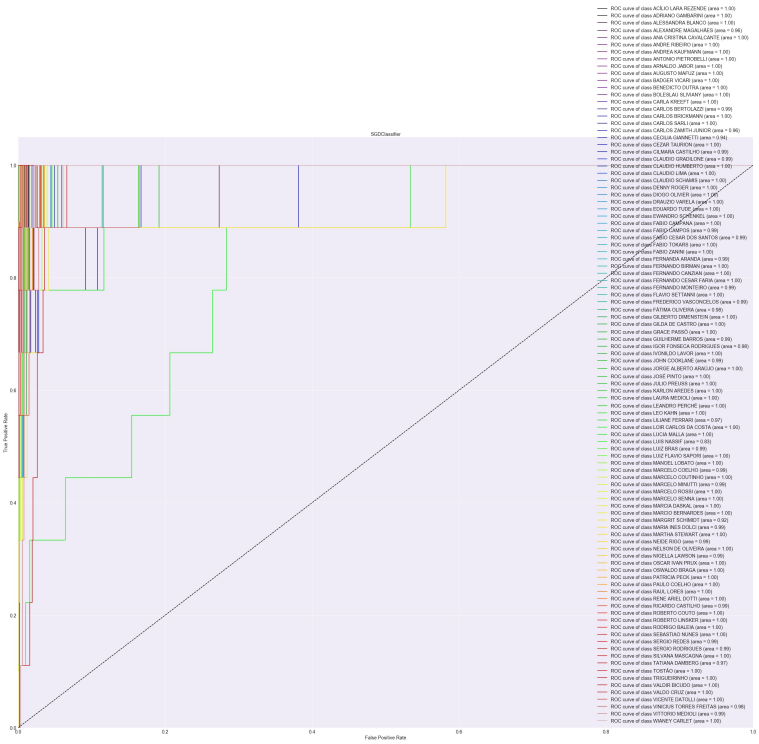
```
Train acc: 1.0
Test acc: 0.8423973362930077
Train f1-score: 1.0
Text f1-score: 0.8423973362930077
```

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 distribuition. But, it made the accuracy decrease:

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

In the more complex models it was tested, in most cases, three types of features extractions: TF-IDF, Word Embeddings and Glove Word Embeddings. At the model level, I tested some layes with Dropout, Max-Pooling, Global Max-Pooling and keep the main layers (Convolutional, LSTM, and Dense) as simple as possible.

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.

Machine Learning Evaluations - **notebooks/03-ML.ipynb**

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
```

With the following hyperparameters with TF-IDF and Stochastic Gradient Descendent Classifier:

```
[('vect', TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
          dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
          lowercase=True, max_df=1.0, max_features=None, min_df=3,
          ngram_range=(1, 3), norm='l2', preprocessor=None, smooth_idf=1,
          stop_words=None, strip_accents=None, sublinear_tf=1,
          token_pattern='(?u)\\b\\w\\w+\\b',
          tokenizer=<function word_tokenize at 0x1a17e4e9d8>, use_idf=1,
          vocabulary=None)),
 ('clf',
  SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
         eta0=0.0, fit_intercept=True, l1_ratio=0.15,
         learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
         n_jobs=-1, penalty='l2', power_t=0.5, random_state=42, shuffle=True,
         tol=None, verbose=0, warm_start=False))]
Optimized Model
------
Final accuracy score on the testing data: 0.8568
```

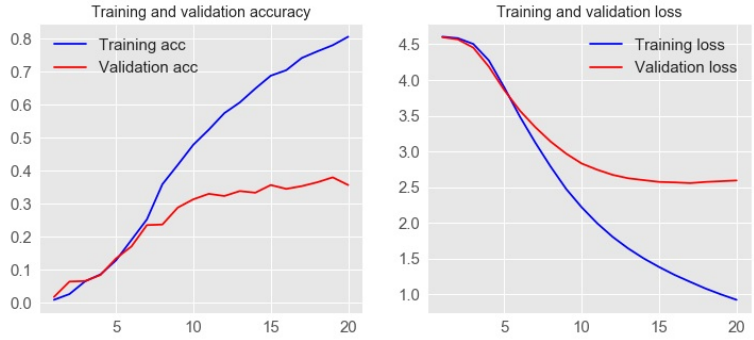
Shallow Neural Network Evaluations - **notebooks/04-NN.ipynb**

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
```

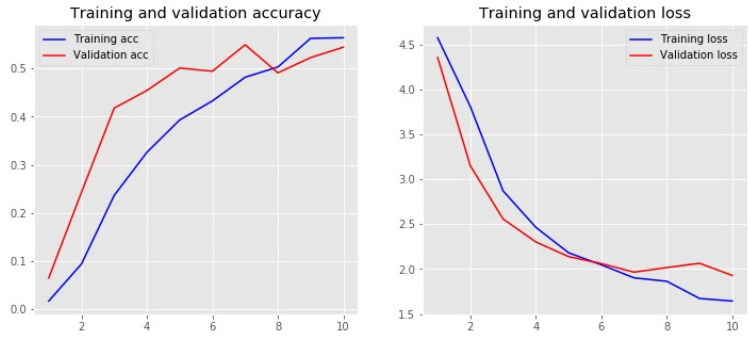


Deep Neural Network Evaluations - **notebooks/05-DNN.ipynb**

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
```

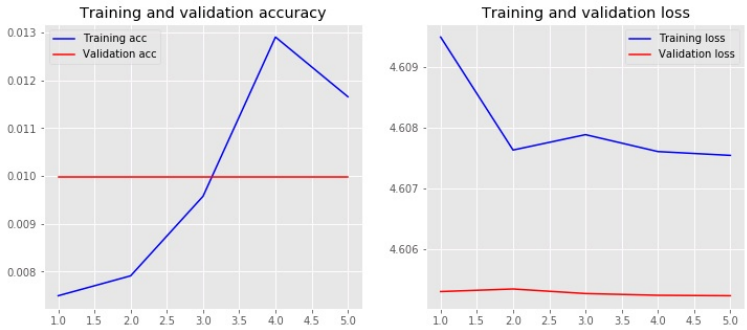
Recurrent Neural Network Evaluations - **notebooks/07-RNN.ipynb**

Better result was with word embedding and LSTM, but probably it has some problem beacause 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.

```
N   601
P@1 0.324
R@1 0.324
```

As we can see, the best model was the one with a Stochastic Gradient Descendent classifier with log loss. But, we can observe that it is overfitted Another result, also with Stochastic Gradient Descendent 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.

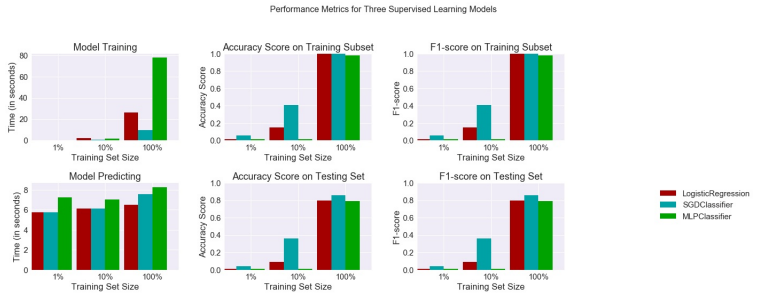
My best accuracy was 0.84 on the overfitted model and 0.72 in a less overfitted model. The original result was 0.74.

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 then 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 OOJ2013 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;
- Preprocced 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.