Application of Machine Learning Techniques for Book Genre Classification

Final project for the course Mining of Large Datasets 2022 at Télécom Paris

Rodrigo Calzada Haro Alex Elenter Thibaud Labat Guillermo Toyos Marfurt

***Abstract*—This work studies the application of different machine learning techniques, specifically in the domain of natural language processing (NLP), for classifying the subject of a text. For this project, the content of different books will be processed using NLP tools to feed different algorithms, such us decision trees, binary bayes classifiers, linear and logistic regressions, and neural networks. Our main conclusion is that NLP requires more complex techniques, however decent results have been achieved using SVC and neural networks trained with tf-idf features. Using neural networks, we obtained an average 0.71 F1-score for classifying 21 different subjects.**

I. Introduction

Nowadays, the analysis of big data has become of paramount importance for taking decisions, discovering relationships and creating solutions.

In this work, we tackle the problem of book classification. It might be tedious to manually read an entire book to classify it based on its subject. And sometimes it's hard, even for the author, to assign a precise subject to their book, as this decision might affect its reach and target audience. Additionally, there are books which are hard to classify, and fit into multiple subjects.

Because of this, we found this problem challenging and compelling enough to aboard it on the final project of this course.

Our objective is, given the content of a book, perform the classification task of determine the subjects of the book. For achieving this, we use multiple machine leaning algorithms - such as decision tree, naive Bayes classifier, neural networks, support vector classifier, linear regression, logistic regression and random forest - and we compare their results.

The rest of this work is organised as follows: Section II-A explains the dataset used, how the data was fetched, pre-processed and the techniques used for extracting features. In section III we show the different machine learning techniques we applied and study their results in section IV. Finally, in section V we give some conclusions and show future lines of work and points to improve.

II. Data fetching and processing

A. About the dataset

For the experimental evaluation we used the a dataset of books provided by the Gutenberg Project [3]. This dataset is composed of a catalogue of 60'000 free e-books in UTF-8 format including different metadata elements such as: title, author, date of release, language, and subjects.

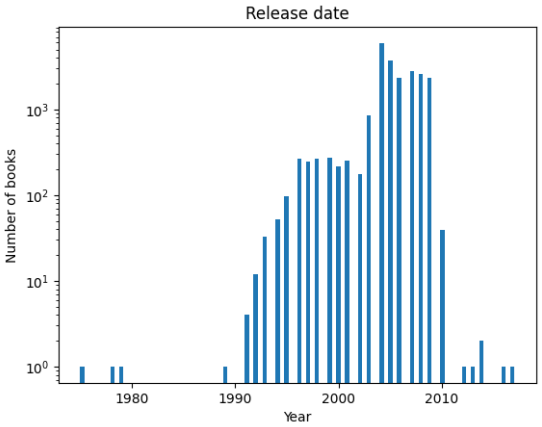

Figure 1: Distribution of books according to publication date.

B. Data exploration

In this section we study some of the key characteristics of the dataset, its weak points, and the value distribution of the metadata.

Regarding the publication date, in figure 1 we observe that the publication date of most books go are around 1990 and 2010. This is an important characteristic of our dataset: We are working with books from 10 to 30 years ago. As language, evolves over time, we expect that our models will specialise in classifying books of this period and are not going to be as effective for classifying books from the mid XX century or older.

One of the main concerns of this dataset, which could impact the evaluation results of our model, is if they are repeated books. This could happen as there might be multiple editions of a book, which would then have essentially the same content. Repeated titles could produce a bias when measuring the accuracy of our model, as we may get the same book in the evaluation and training samples, Figure 2 shows the number of books that share the same title. We can see here that most of the titles only appear once (99%). So we decided to remove these duplicates.

Lastly but most importantly, we study the distribution of the book subjects. Figure 3 shows the distribution of the different

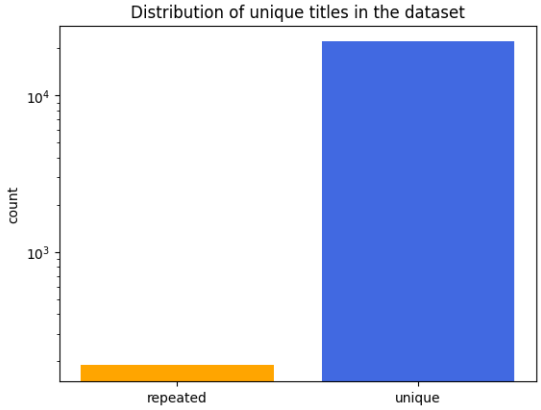

Figure 2: Number of unique and duplicate books. Although cleaning is necessary, the loss of information is not noticeable.

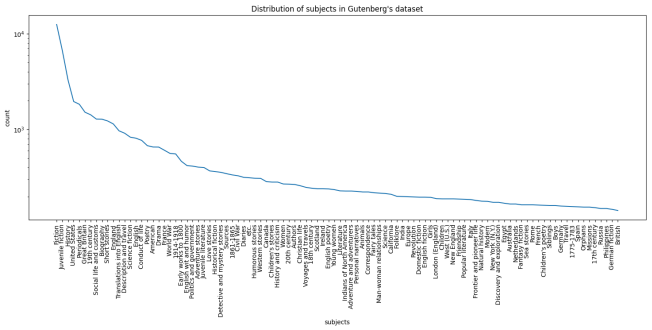

Figure 3: Number of times each topic appears in a book. Limited to the 100 most common.

subjects. We observe that they are not evenly distributed as some subjects have a bigger count by two orders of magnitude. Additionally, we consider that some subjects are semantically less relevant. For instance, we are not really interested in classifying books by the label "California" or "1775-1783" but rather classifying by "Drama", "Science fiction" or "Mystery". This is of course a subjective assessment, other scientists may differ in this point and obtain different results.

C. Data cleaning

We filtered this dataset, taking only books in English, and which their metadata is complete (i.e we removed null values). After filtering, we ended up with a collection of 20'000 English books with appropriate metadata values.

Additionally, the books needed to be pre-processed: for each book, we removed their first and last pages that included informative and legal content of the Gutenberg Project and are not part of the book itself. After that, we removed stop-words (such as punctuation signs, connectors, articles, etc.) using NLTK [1].

D. Feature extraction

1) Subjects: Taking into account what we discussed in section II-B, we decided to make a selection of the subjects that we thought that are most significant to classify while

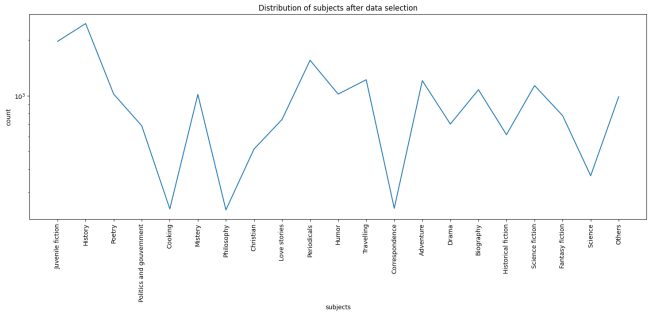

Figure 4: Number of times each topic appears in a book. Limited to the 100 most common.

taking in consideration the amount of books that are available for these subjects. That is, we handled the data imbalance by selecting the most significant subjects and down-sampling the ones that had the biggest book count. Figure 4 shows the distribution of this new sample. Despite data imbalance was reduced considerably, it is not completely balanced. We consider this level of imbalance shouldn't affect the performance of our algorithms due to the metrics and methodologies applied in the following sections.

After this, we have for each book, a list of subjects. We built a column for each subject and applied a one-hot encoding. As a result, we have for each book a column which points out if the book is of a certain subject or not. This let us focus on analysing the subjects that we consider the most relevant. For instance, as we've observed in figure 4, classifying by the "fiction" label might lead to data imbalances. By using this encoding we handle this imbalance by not considering certain subjects as features for the different algorithms.

2) Book content: For extracting features from the content of each books. We tried two different approaches tf-idf, which we have seen during the course, and Word2Vector [2].

For applying tf-idf, we used the TfidfVectorizer class of Scikit-learn [5], which analyses a set of documents, here each document is a book. After analysing the corpus, a matrix is created where each columns is mapped to a feature and each row is a book. The nature of each column depends on the parameters used, but each one will be an expression found in the corpus. Finally, the cell (i, j) will be set to $tf(i, j) \prod idf(i, j)$, this represents the product of how frequent expression i appears in book j *times* a number that grows inversely to the number of times the expression appears in the corpus. We will now describe the main parameters used for instantiating the TfidfVectorizer. Here is how a TfidfVectorizer object is created:

Listing 1: creating a TfidfVectorizer instance

```
tfidf_vectorizer = TfidfVectorizer(
    ...
    stop_words='english',
    ngram_range = ngram_range,
    max_features = max_features,
    max_df = max_df)
```

Stop words are presumed to be uninformative in representing the content of a text, and they may be removed to avoid them being construed as signal for prediction, setting this parameter to English means we will use the built-in dictionary of stop words. The *n-gram_range* parameter is a tuple (i, j), this means tf-idf will search for expressions containing from i to j words. The *max_features* parameters is self explanatory. Finally, the *max_df* is set to a number between 0 and 1, let's suppose it is set to 0.8, then the model will not take into account expressions that appear in more than 80% of the texts, we can think this expressions as stop words. For defining the hyper-parameters of the tf-idf vectorizer we empirically tried with different combinations and tested how their performance impacted on the different models while also taking into consideration the execution time that was needed to generate the tf-idf matrix. By doing this, we set the parameters to: $max\_df = 0.8$, $max\_features = 1000$ and $n - gram\_range = (1, 1)$. The last one means that features are composed of single words.

On the other hand, Word2Vector tries to accomplish the following: Given an arbitrary value d, create a vector space $\mathbf{R}^d$ such that words that are semantically "similar" have a small distance in this space. By semantically similar, Word2Vec does a contextual similarity strategy: Training a neural network which maximises the likelihood of a word appearing in a certain context. That is, the relationships of words are learnt by the company they keep.

An important observation is that once that this word vector map is generated. It can be used in any context. If it has been trained in a diverse enough dataset, the word embedding should be usable in multiple different context. For our project, we decided to use the public word vector embedding provided by the Standford's GloVe project [6]. Where we used the version where $d = 25$

However, there is a problem when we apply this embedding to books: the number of features we will get will have the size $d*n$ where n is the number of words. So it becomes infeasible to use many words as features. Because of this, a reduction strategy must be applied.

In this project we tried for each paragraph, calculate an average vector according to the words present in it. This lets us take as features a vector of dimension d for each paragraph. We acknowledge that this is a rather basic strategy and more complex algorithms exist such as Doc2Vec [4].

III. Data Mining

A. Baseline (Binary Naive Bayes)

For comparing the performance of each algorithm, we establish a minimum expected performance by using the Naïve Bayes classification algorithm, as we consider that it is the simplest algorithm that could output acceptable results and is computationally light to train. We used the SciKit-Learn implementation of the Naive Bayes [5].

B. Linear methods

With the hope of getting a better performance than the binary naive Bayes classifier, we decided to start with very known classifiers.

- Linear regression classifier. The obtained values after applying the tf-idf transformation has been used as parameters to train a model with a discrete output that can take two values: 0,1. To use the model as a classifier, the tf-idf values of a new book are used as input. If the output of the model is lower than 0.5, the output is considered a 0. While, if the result is higher than that threshold, we consider it a 1.
- Logistic regression classifier. This function is a classifier. Therefore, it is able to directly predict whether the output is 1 or 0.
- Support-vector classifier. The same situation as the former one, it produces a discrete output between 0 and 1.

The process to obtain the classifier of the three of them is similar. We have used the implementation of Scikit-learn of each of them.

Before creating the models, the dataset has been divided between train and test. As X_train, the tf-idf values are used. The Y_train allows us to establish whether a book can be classified into a category or not.

We want to create binary classifiers. Therefore, it is not feasible to predict all the categories of a book with a single model (the set of possible combinations of categories would be huge). So, instead of using a single model to predict all the categories, we created a model for each of the categories. This way, our models can tell us whether a book can be classified into a category or not.

The results are going to be presented at the end in a comparison between the different techniques of prediction. However, it can be said that these first approaches performs already much better than our baseline model in the majority of the categories.

C. Decision Trees

We decided to use decision trees for predicting the genre of the books, which has the main advantage that we can easily visualise which are the key words used to classify. Moreover, it also allows us to see which path a specific vector followed to be classified.

For example, figure 5 and figure 6 show the decision trees for the cooking and mystery classifiers respectively. It's interesting to observe that the words "murder", "police", "secret" and "watching" are used to classify mystery books, meanwhile the words "add", "butter", "served", "sugar" and "food" are used by the cooking classifier. In addition, we can observe that the classifier for mystery has a lot more depth and branches. This highlights that there are subjects that are much more complex to classify than others.

D. Neural Network

We implemented two different Neural Network architectures to solve our classification problem. The first one is a multi-

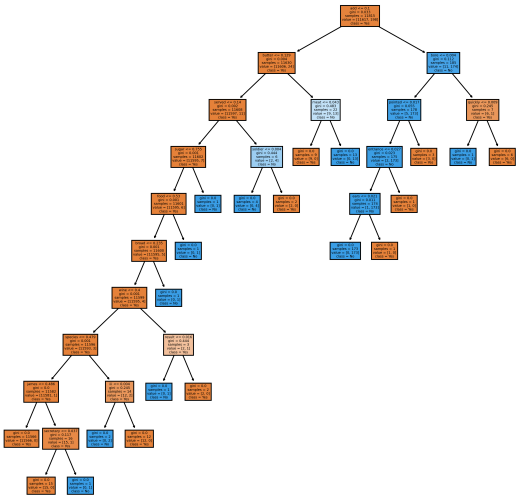

Figure 5: Decision tree to classify books by cooking or non-cooking.

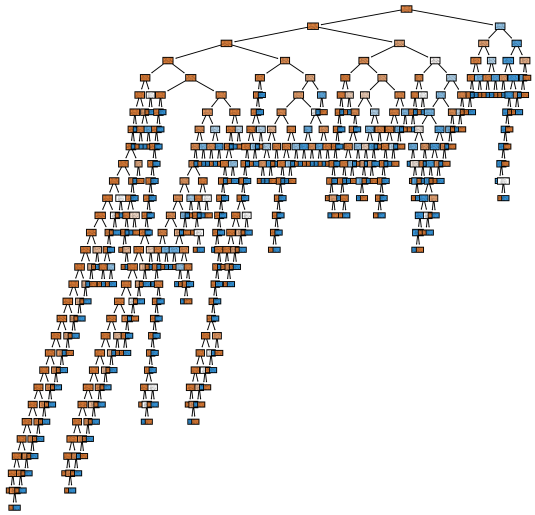

Figure 6: Decision tree to classify books by mystery or non-mystery.

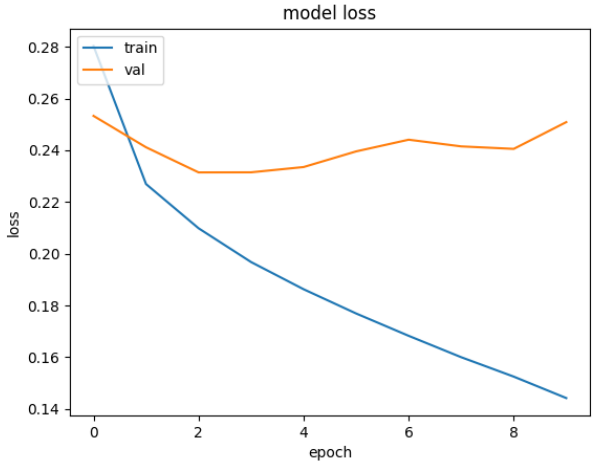

Figure 7: We can observe over-fitting around epoch 3 and onward, as the loss function evaluated on training data diverges from the loss of validation data.

class classifier, which tries to predict if a book belongs to each category in one pass. The second architecture we considered is a binary classifier, duplicated for each category (so we have as many instances of this model as the number of categories we have).

The multi-class classifier is a classic Feed-Forward Neural Network model, taking as input a tf-idf feature vector of size 1000 (see above to know how it is generated), with two hidden layers of size 50 with ReLU activation, with finally an output layer of size 21 (as we only consider here the top 21 categories for our books) with Softmax activation. The shape of the network is classic enough not to explain it, whereas the size of each hidden layers has been chosen so that over-fitting does not occur. To know whether we have over-fit our model, one can plot the figure of the Loss function both for the training data set and the validation data set, against the number of elapsed epochs of model training (see figure 7). If the loss function evaluated on training data is well under the loss of validation data, there is over-fitting. We gradually decreased the number of coefficients in our model, by reducing the number of hidden units, so that it does not happen.

The binary classifier model we used, trained and used fully independently for each category we wanted to predict, has the same structure than the multi-class one. We reduced the number of hidden units : The first hidden layer has 20 units, when the second has 10. Those numbers were chosen with a trial-and-error approach with the goal to reduce over-fitting, as stated above.

When the models are trained, either once for the multi-class classifier or multiple times for the binary classifier (one per category), we can predict the category of a book by feeding the models with the tf-idf vector computed from a given text, and using a threshold defined beforehand (once and for all ; one threshold for each category) to determine if the output of the network, which is in the range (0,1), is negative or positive for

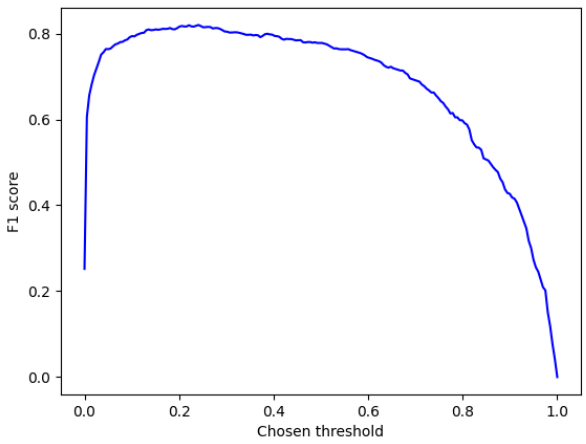

Figure 8: For the "Juvenile fiction" category, this is the F1-score for a given activation threshold.

Category	multiclass F1	binary F1
Juvenile fiction Others	0.820	0.852
History Others	0.529	0.546
Poetry Others	0.814	0.796
Politics and gouvernment Others	0.742	0.793
Cooking Others	0.909	0.911
Mistery Others	0.725	0.725
Philosophy Others	0.597	0.696
Christian Others	0.608	0.671
Love stories Others	0.521	0.526
Periodicals Others	0.819	0.872
Humor Others	0.800	0.839
Travelling Others	0.695	0.681
Correspondence Others	0.444	0.649
Adventure Others	0.485	0.565
Drama Others	0.864	0.848
Biography Others	0.665	0.683
Historical fiction Others	0.615	0.649
Science fiction Others	0.905	0.903
Fantasy fiction Others	0.696	0.687
Science Others	0.658	0.762
Others Others	0.313	0.270
AVERAGE	0.677	0.711
STANDARD DEVIATION	0.155	0.149

Figure 9: F1 scores obtained in each category for both models, with best chosen activation thresholds.

our classification problem. The performance of the network, and the number of true/false-positives/negatives, depend on the choice of the threshold. Such a plot of F1-score depending on the threshold is plotted on figure 8.

For each category (whether we use the binary or multi-class model), we computed the F1-score of each model for many thresholds in the range (0,1), and keep the thresholds giving the best scores. We keep these threshold values and see them as coefficients of our model for solving the given classification problem. The computed scores are shown in figure 9. The average F1 score (for each category) is 0.677 for the multi-class model and 0.711 for the binary models.

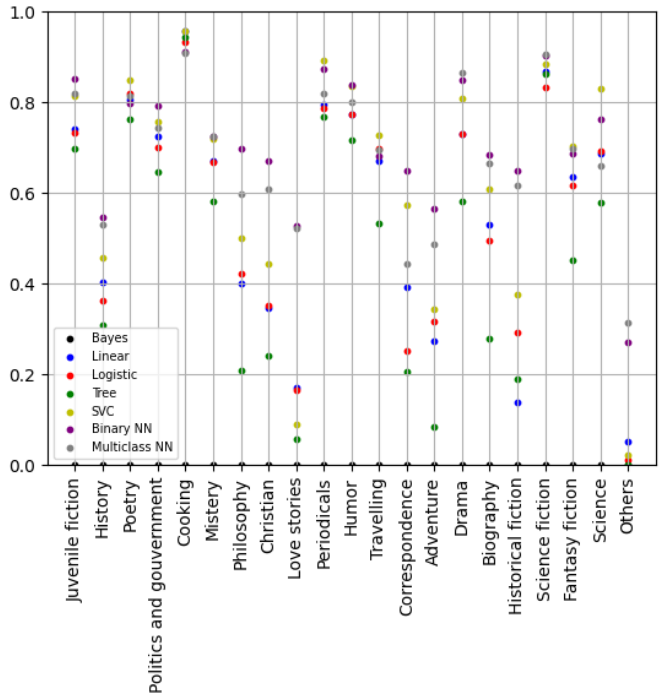

Figure 10: F1-score for the classifiers of each subject using different algorithms and features generated by tfidf

IV. Results

A. *Tf-idf*

The results are presented in figure 10. The F1-score of the different models has been used for the evaluation. We have chosen this metric because it correctly handles the dataset imbalance and is appropriate for a binary classification task. The scikit-learn implementation has been used [5].

The naive-bayes classifier obtains the worst result. The positive point is that we can conclude that choosing this model as the baseline can be considered a good strategy.

Regarding to the gradient-based methods, we can see an improvement on the performance. The expected results are good in general, with several categories outstanding over the others. We can see that the category *other* is difficult to classify. This can be easily explained. In this category, the books that are not included in other categories are selected. Taking into account that it can exist no relationship between the content of these books, it is intuitive to conclude that the amount of words obtained from the tf-idf for this category can be very broad, so we cannot use them to predict books classified as *other*. Regarding to the algorithms, the one that stands out is SVC. This was the expected result. The linear and logistic classifiers are similar. To sum up, the difference is reduced to the use of a *sigmoid function* at the end or not. However, SVC not only separates the results, but also is able to establish the threshold that better allows to classify between different values. In our domain, SVC defines the average limit to decide whether a book belongs to a category or not. This

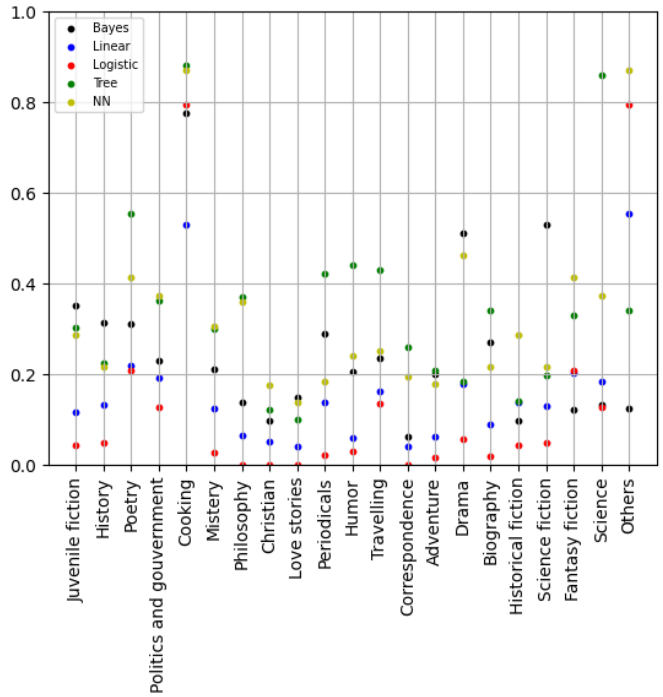

Figure 11: F1-score for the binary classifiers of each subject using different algorithms and features generated by word2vec

limit is more accurate, so, the results for the test set, which are the ones presented in the scatter plot, are better for this algorithm.

B. Word2Vector

Figure 11 shows the f1-scores of each classifier coloured by the type of model used. We couldn't apply SVC and random forest due to the long training time of these algorithms under a high number of features.

We observe that for almost all subjects (except cooking, science and others) all models perform relatively poorly. We argue that the technique used for extracting the word vector features, average vector by paragraph, might not be effective. We conjecture that the great results obtained for cooking and science subjects are due to the fact that these type of books tend to use a more specific range of vocabulary (for example, cooking books tend to include mainly kitchen/food related terms in each paragraph. On the other hand, an adventure book could virtually talk about anything on each paragraph.) thus the average vectors could be more characteristic of the book's subject.

V. Conclusions

The experimental results have proven that the task of book classification is very challenging. Considering the F1-Score metric, and starting from a baseline (Bayes classifier) that gives very poor results, we were able to obtain better results using other well-studied algorithms such as SVC, Decision Trees and Neural Networks. However, these algorithms did not yield good results in every scenario studied.

On the other hand, at a higher computational training cost, the neural networks performed fairly better than the baseline and the decision tree algorithm. This shows that it is required to use more powerful tools to successfully tackle this classification task.

In conclusion, the task of NLP remains a challenging one and is still being studied to this day. Even if we were able to obtain decent results using basic techniques, more complex algorithms are to yield better results.

Finally, we consider that this project has been a great opportunity for the authors of this work to learn the subjects seen in class and broaden their knowledge on the NLP area. It also provided the opportunity to discover the challenges of solving a data science problem in a practical way. From the obtaining of the dataset to the analysis of the results given by the machine learning algorithms. We consider this has been a very enriching experience.

A. Future lines of work

We have identified multiple lines of work that could improve and enrich our approach to solving this classification problem:

1) Applying other text document encoding techniques such as Doc2Vec [4]. With this, we might extract more effective features
2) Use transformer neural networks. Which has been proven that they are a powerful tool for natural language processing which throw great results in text classification tasks.
3) Consider the use of clustering techniques to see if the features extracted can be grouped into relevant groups that take on account the subject of the book.

B. Acknowledgements

We thank Project Gutenberg for providing the free books we used for this experiment and Télécom Paris for letting us to use their servers for running the different experiments.

C. Contributions

Regarding the contributions of each member, Thibaud was responsible for creating the work environment (setting up a real-time collaborative Jupyter Lab notebook on a server at Télécom), and doing the Neural Network part with tf-idf features. Rodrigo worked on the text feature extraction using tf-idf, hyper-parameter tuning and computation of linear, logistic and SVC models. Alex did the decision tree experiments and hyper-parameter adjustment of tf-idf. Guillermo was responsible for creating and cleaning the dataset from the Gutenberg project, defining the baseline, exploring the dataset, plotting the decision trees, and designing the algorithms and feature extraction techniques with word2vec.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[2] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method, 2014. cite arxiv:1402.3722.

[3] Project Gutenberg. Project Gutenberg. https://www.gutenberg.org.
[4] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.
[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.