DTI5125[EG]: Data Science Applications

Group Assignment 1

Classification

Group: 6

Contents

1. Introduction

Problem Formulation

In this assignment, we are trying to solve a text multi-classification problem for some of the Gutenberg books. In abstract level, an input of text partitions/paragraphs of a book goes to a pipeline that ends up with an output of the predicted book name.
We will go further into more details to check each step of this process that led us to a specific prediction and how we can evaluate whether this prediction is good enough for us or not along with what caused this prediction to be in this way not in any other way.
We shall aid our explanation with some visual illustration to ease the perception of the process.

2. Data

2.1. Chosen data

We are working on the Gutenberg digital books. We chose specifically 5 books that we believe might be close in their semantic/genre, so that it will make our problem more challenging. Each book is provided with the URL that you can reach it from.

2.2. Data Preparation

As we mentioned, we focus on five books from the Gutenberg digital books, from each book we take **n** *(e.g. 200)* **random samples**, where each sample consists of **m** *(e.g. 100)* **words** from the book. Hence, for 5 books this means we have an overall of 200*5 = 1000 samples/records and 200*100*5 words/tokens.
Each sample is accompanied by its label *(e.g. the book's name it was drawn from)*
In abstract level, a sample is just a part of a paragraph.

The figure below shows an example for this:

198	list	1	[('milton-paradise.txt', 'fast With horrid strides Hell trembled as he ...
199	list	1	[('milton-paradise.txt', 'which we would know whence learned who saw W ...
200	list	1	[('chesterton-thursday.txt', 'be the shortest possible Syme also strol ...
201	list	1	[('chesterton-thursday.txt', 'which is as old as the world Is he reall ...
202	list	1	[('chesterton-thursday.txt', 'saw said the other looking at the leaden ...

Figure 1: Part of the 1000 samples as an example

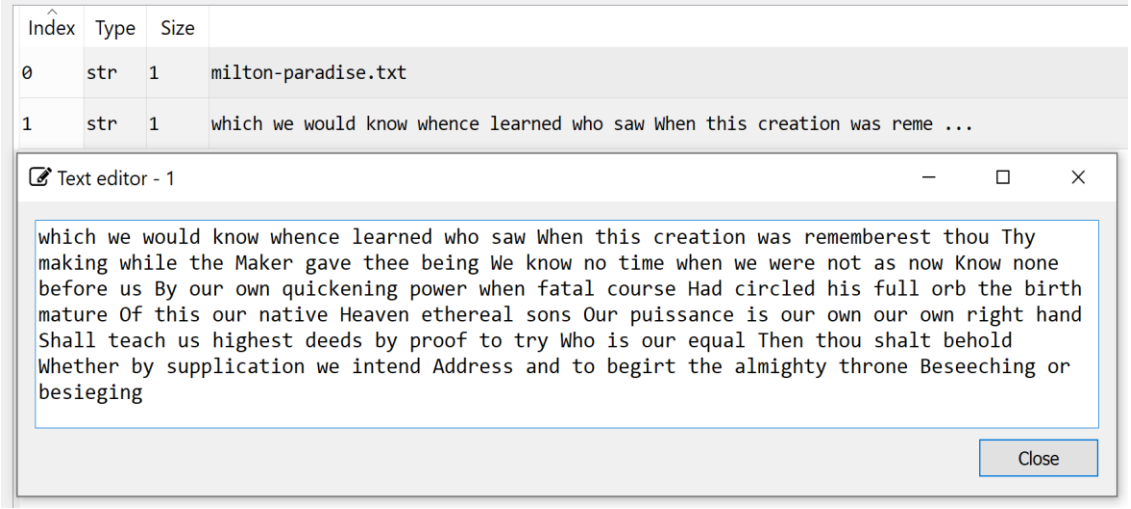


Figure 2: comprehensive view for a sample that is a list of the book's name and 100 words of the book

2.3. Data Preprocessing

This is no easy task, even though text is easy -somehow- to be interpreted by human beings, it's a no-easy-task for the machines. Text might have punctuations that mostly are irrelevant to a problem, also text is full of junk words that are low-to-zero contextual and highly repeated as "the, a, have,… etc". Furthermore, the paramount issue is that machines don't understand "words" in the first place. Machines are only cognizant of numbers so we have to find some methodologies to resolve these issues and this is the preprocessing and feature engineering phases.

Let's start with basic preprocessing:

2.3.1. Basic Preprocessing

2.3.1.1. Tokenization

We have been saying “words” too much up until now, but do we input our text files and documents as separated words? Or -normally as expected- we input them as they’re; documents comprising of several paragraphs.
Thus, this implies that we have to break down these long, connected paragraphs into single words to facilitate our work. This is what’s known by: Tokenization.
Tokenization: Breaking down sentences into words (and maybe characters but here we’re using word tokenization)
“NLTK” provides us with various tokenization techniques that can be checked. [1]

2.3.1.2. Punctuation

Punctuations as “? , . !.....etc” don’t encompass much information, so it’s sometimes better to remove them to halt learning from noise and so on.
Again, “NLTK” helps us in this removal with simple commands. [2]

2.3.1.3. Stopwords

Stopwords is a notion that refers to words that are very common to be used in natural speaking and writing without adding much value, words like: “the”, “is”, “in”, “for”, “to”, “at”. These words pose high redundancy without value so we’d just get rid of them. [3]

2.3.1.4. Lowercasing

As a rule of thumb, most of times you’d just lower case all the letters to unify your work, as upper-case letters differ from the lower ones for the machines, but they don’t mean much in this context.

2.3.1.5. Stemming

Words can come in different forms even if they contextually mean the same, things like different tenses, single and plural forms.
For instance: cat & cats – play & plays – kill & killed
For each of the 3 pairs, they're the same, right? except for the suffix, thus, can we have something that removes this suffix? to unify the perception and deepen the perception of the machine for this word and put different forms of the same word in the same basket.
Here where "stemming" comes into action to -naively- remove the suffix. [4]
Therefore, after stemming, cats cat – killed kill – wolves wolv; and that's exactly why we said naively.

Nevertheless, can we move a step further to work with more intelligence? This is where "lemmatization" comes to action.

2.3.1.6. Lemmatization

Lemmatization is about returning to the basic/root form of the word, instead of just eliminating the suffix, here we are returning to the lemma *"root in Latin"* of the word. [4]
So, cats cat, killed kill & wolves wolf

We just accentuated on the main difference, however, there're essential uses for both and neither is the best all the time. [4]
Both techniques can be maintained by the "NLTK" libraries and others.

Document	tokinized_doc	label
thee time hath arriv â twixt rosi dawn rosi da...	[thee, time, hath, arriv, â, twixt, rosi, dawn...	Thus Spake Zarathustra
thi love thi modesti make revel unto rage soul...	[thi, love, thi, modesti, make, revel, unto, r...	Thus Spake Zarathustra
known hitherto fervour tone also pure enough m...	[known, hitherto, fervour, tone, also, pure, e...	Thus Spake Zarathustra
case differ inasmuch commun oppress suffer une...	[case, differ, inasmuch, commun, oppress, suff...	Thus Spake Zarathustra

Figure 3: Sample for filtered records after we performed all the afore-mentioned preprocessing steps

2.3.2. Feature Engineering

Now that we finished cleaning the text itself, we need to make it interpretable by the machine. In other words, we need to convert the text into numbers; a process known as: vectorization, where these numbers (mapping from words to numbers) are saved in matrix *(i.e. vectors)*
Vectorization has different methodologies that we shall discuss some of.

2.3.2.1. Count vectorizer (Term frequency; TF)

This means that each single word in all the documents will have a cell in the vector denoting how many times it occurred. Apparently, this infers a dramatically sparse matrix [5].

2.3.2.2. N-grams

N-grams is a modification for the “count vectorizer” where instead of just taking each single word (unigram), we can take a combination of words.
Thereby, for a sentence like “I hate tight deadlines” the TF will be [I, hate, tight, deadline], but for n-grams (bigram) it will be [I hate, hate tight, tight deadlines]
This can be extended to any n-grams (3-grams/ 4, 5,….) [5]

2.3.2.3. Term Frequency-Inverse Document Frequency (TF-IDF)

An extension for TF technique where another factor is incorporated which is the IDF. To begin with, DF is the number of documents a word appeared at, IDF is the inverse of it. Why does this help? This helps significantly in pulling out rare words that comprise a predominant value and meaning for the problem in hand [5].

Easy-to-use built-in functions using “python” and “NLTK” can be found in [6] for all the three vectorization methods.

Now after we had clean and neat vectorized samples for each book, namely features to enter the model. We need to move forward with the model that will take this input and learn from it. Nevertheless, there is a tiny step that is left over before digging into the model, which is splitting this clean-and-neat data.

2.3.3. Splitting

As usual, before digging into a machine learning model, it’s prevalent that we split this data into three partitions: “Train” for training the model, “Validation” for evaluating the model’s performance before productionizing it, “Test” which acts as the production/test data mimicking the real-world scenario.
Here, We split the data by the ratio of 80:20 for Train and Test, respectively; *(i.e. 800 train samples and 200 test samples)*.
Moreover, the train set is furtherly split using the k-fold cross validation techniques that will be discussed in the evaluation section.

Now, no left overs anymore and we can safely move on to the models.

3. Models and Algorithms

For the models section, there are tons of models' combinations that can be used, we took a snippet of them as SVM, Decision trees and KNN.

In this assignment, we used several models that we will give a short note of, aided with each ones' metrics as the "classification report" to inform us of the viability of this model where we use that fitted model to predict the test cases we kept aside beforehand.

In all the coming models we perform kfold cross validation to split the model into train and validation sets

3.1. Evaluation (KFold Cross validation)

To evaluate the model performance, we are going to split the train data using kfold cross validation with k=10 folds for each model and then assess the average accuracy score for the 10 folds.

3.2. Different Models, Performance

3.2.1. Decision Tree

Firstly, we tried working with decision tree, we applied the three different vectorization techniques with the decision tree, however the one with the best accuracy, which is TF-IDF didn't yield sufficient accuracy, as shown here.

```
Accuracy:  0.815
cross-val scores:  [0.7625 0.7625 0.7875 0.7625 0.8625 0.825  0.7875 0.825  0.7375 0.75  ]
cross-val scores' mean:  0.78625
```

But at least this directed us towards focusing more on using TF-IDF with different models.

Decision trees are known of their tendency to be of a high variability and overfitting due to the depth of the trees. The bias and variance tradeoff is manifested in the following figure.

```
Average expected loss: 0.248
Average bias: 0.220
Average variance: 0.138
```

That's why we also tried to depict this by visualizing the splits of the decision tree as shown below.

Out[69]:

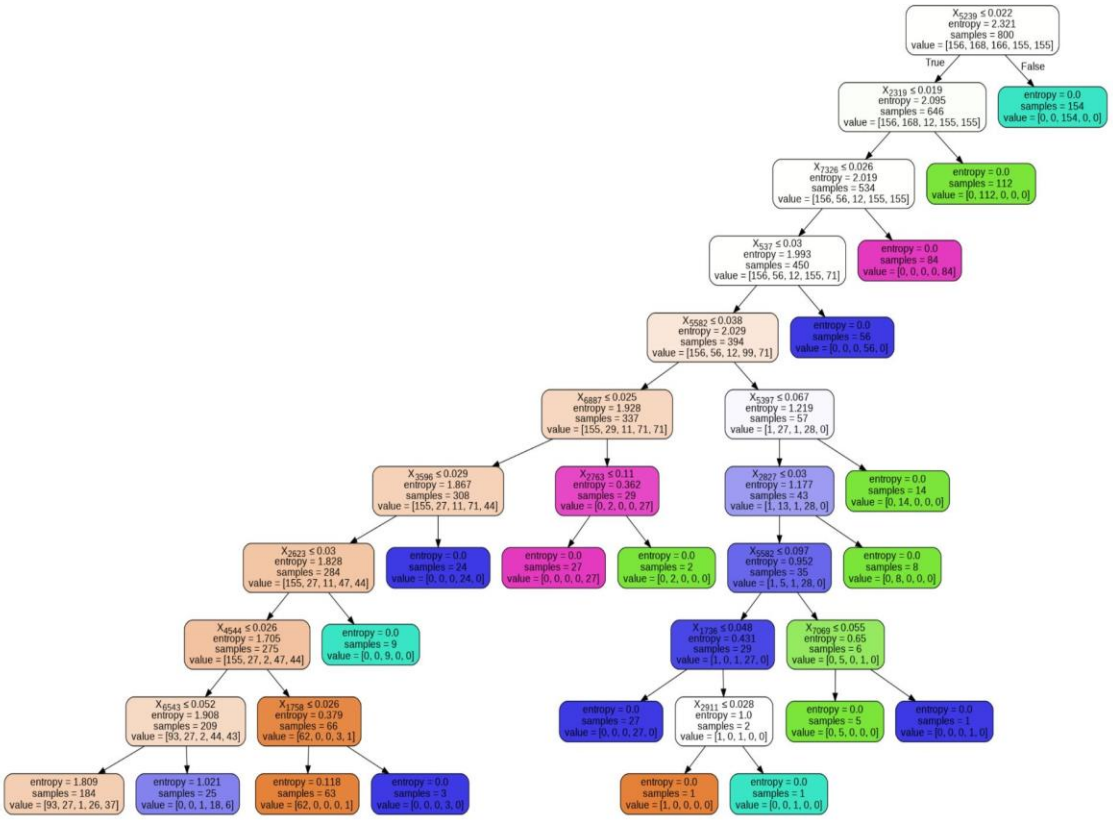


Figure: The branches/splits of the decision tree manifested in this graph

Thereby, we started to get back to a simpler yet well-performing model, one of the most intuitive machine learning algorithms which is KNN.

3.2.2. KNN

After we shifted to KNN in order to improve the accuracy, the performance really enhanced in a good way, showing that simpler algorithms are not always useless but can yield high results.

```
Accuracy:  0.94
cross-val scores:  [0.9375 0.95   0.9625 1.     0.9625 0.95   0.9125 0.95   0.9125 0.9625]
cross-val scores' mean:  0.95
```

For the bias and variance tradeoff:

```
Average expected loss: 0.080
Average bias: 0.050
Average variance: 0.058
```

Obviously this is a tremendous improvement. Also, the bias and variance decreased dramatically. However, is there any room for more advancement? We were still yearning for so.

3.2.3. SVM

Support vector machines which have been known for their high performance in both linear and nonlinear classifications were our next step. And -as we expected- the results were extravagantly higher than what we started with in Decision trees.

```
Accuracy:  0.995
cross-val scores:  [0.9625 0.9625 1.     1.     1.     1.     0.975  0.975  1.     1.    ]
cross-val scores' mean:  0.9875
```

And same goes for the bias and variance tradeoff:

```
Average expected loss: 0.008
Average bias: 0.005
Average variance: 0.004
```

Last but not least, we shifted to more of a linear algorithm; logistic regression to check if non-linearity of SVM is adding any more information or it can be even better if we suffice with linear solutions. Also, we wanted to work with logistic regression to analyze the misclassification using our error analysis techniques.

```
Accuracy:  0.995
cross-val scores:  [0.975  0.9625 1.     1.     1.     0.9875 0.9625 0.9875 1.     1.    ]
cross-val scores' mean:  0.9875
```

Apparently, the accuracy didn't change from what we got at SVM, but it wasn't degraded too, meaning that linear separation is yielding good results in this multiclass problem.

3.3. Bias-Variance Tradeoff

In the table below, the error, bias, and variance values are listed for the investigated models.

	Decision Tree	**KNN**	**SVM**	**LR**
Error	0.248	0.080	0.008	0.011
Bias	0.220	0.050	0.005	0.01
Variance	0.138	0.058	0.004	0.006

As we can see, the model with the highest variance is the **decision trees**. This is due to its inflexibility which makes it prone to overfitting the data. Moreover, we have the **KNN**, which has a much reduced variance and bias compared to the former. However, one can notice that, unlike the **Decision Trees,** the value of the variance is higher than the bias, which emphasizes on the notion of the trade-off between the bias and the variance.

In the case of **SVM**, which is a relatively less complex algorithm than the **KNN,** the value of the variance decreased more than the bias. The more a complex a model is, the more prone it is to overfitting. Last but not least, the **logistic regression** algorithm continued on the same trend as its precedent, favouring having a less variance compared to the value of the bias.

In our opinion, the **SVM** algorithm did a great job in finding a good balance between bias and variance, however, we chose to proceed our research with **LR** for simplicity and visualization purposes.

4. Error Analysis and Visualization

As we have clearly seen, there are some drifts in the models and misclassifications. But just saying it like this has never been enough, Why did the model predict this class not that? What made it misclassify? What are the main features that caused the model to drift or even to classify correctly? Answers to such questions help us interpret the results more, superseding the notion of using machine learning as mere "black box" techniques.
For this aim, we use two main libraries that help us reach more interpretation and explanation for what features are those that induced the model to classify this way.

4.1. ELI5

The first library is "eli5", after installing and importing it, we use it by two ways: general and local.

General is where we use the “TextExplainer” for the library to draw conclusions about the most informative features for each class in general [6].

y=Criminal_Psychology top features		y=Deep_Waters top features		y=Murder_in_the_Gunroom top features		y=The_Brothers_Karamazov top features		y=Thus_Spake_Zarathustra top features	
Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature
+5.078	may	+14.011	ernest	+14.936	rand	+8.166	alyosha	+9.684	zarathustra
+4.889	wit	+7.609	mildr	+7.092	river	+6.653	mitya	+6.787	nietzsch
+4.857	observ	+5.969	er	+7.036	fleme	+6.527	itâ	+6.603	thou
+4.666	crimin	+5.912	clara	+5.109	pistol	+6.518	though	+5.574	ye
+4.603	case	+5.400	comston	+4.181	dunmor	+5.403	didst	+5.567	unto
+4.545	etc	+5.348	said	+3.861	collect	+5.391	iâ	+5.484	thu
+4.117	crime	+5.315	god	+3.791	mr	+5.087	ivan	+5.450	howev
+3.649	der	+4.548	predestin	+3.730	gresham	+4.873	father	+5.290	nietzscheâ
... 2809 more positive ...		+4.385	chapter	+3.706	ritter	+4.634	come	+5.225	hath
... 6279 more negative ...		+4.332	presbyterian	+3.322	revolv	*... 2305 more positive ...*		+4.793	zarathustraâ
-3.847	rand	*... 2113 more positive ...*		*... 2468 more positive ...*		*... 6783 more negative ...*		*... 2613 more positive ...*	
-4.619	would	*... 6975 more negative ...*		*... 6620 more negative ...*		-5.722	ernest	*... 6475 more negative ...*	

Figure 4: Global application of eli5 on the dataset

Local is where we specify a certain data point (text sample) and examine the probability of each label for it along with the features that increased-decreased the probability of this label.

In this context, we pass specifically the misclassified samples (index: 125) to check for them, and we can get them simply by comparing the prediction of the classifier with the true label of the sample.

```
error_df=pd.DataFrame({'X':X_test,'y':y_test,'y_pred':y_pred_LR})
error_df=error_df.query('y != y_pred')
error_df
```

	X	y	y_pred
125	fail make mark outset career never made later ...	The_Brothers_Karamazov	Criminal_Psychology

Figure 5: Returning the index of the misclassified samples to be investigated further using eli5

In the following figure, you can see different colors. The colors comprise a prime meaning, where green color implies positive collaboration to that specific class whereas red color implies a negative one. Moreover, the intensity of the color relates to the degree of the collaboration. Also, in an interactive notebook you can see that when you hover over a word; its collaboration number appears (like 0.29, -0.13 and so on)

Figure 6: Checking over the wrongly-predicted class and the true one, check the words in the tested -misclassified- sample to check which words had an impact on the decision; here we can see that "signific" and "case" words have quite high impact "0.239" to lead the model to 0.686 probability of CRIMINAL_PSYCHOLOGY book.

The above figure depicts that the model interprets words like "case, signific, phenomenon" as representations (feature space) of the CRIMINAL_PSYCHOLOGY book.

4.2. LIME

Lime works nearly the same as "eli5", with different visual representation, where you have the capability of choosing how many features (words) to display along with their contribution too [7].

Here we only manifest 6 features/words with the top 3 labels, these features/words can be also mapped on the text words. The right hand sided words are the words that caused positive collaboration to that class whereas the left sided ones are the words that decreased the vote for this class.

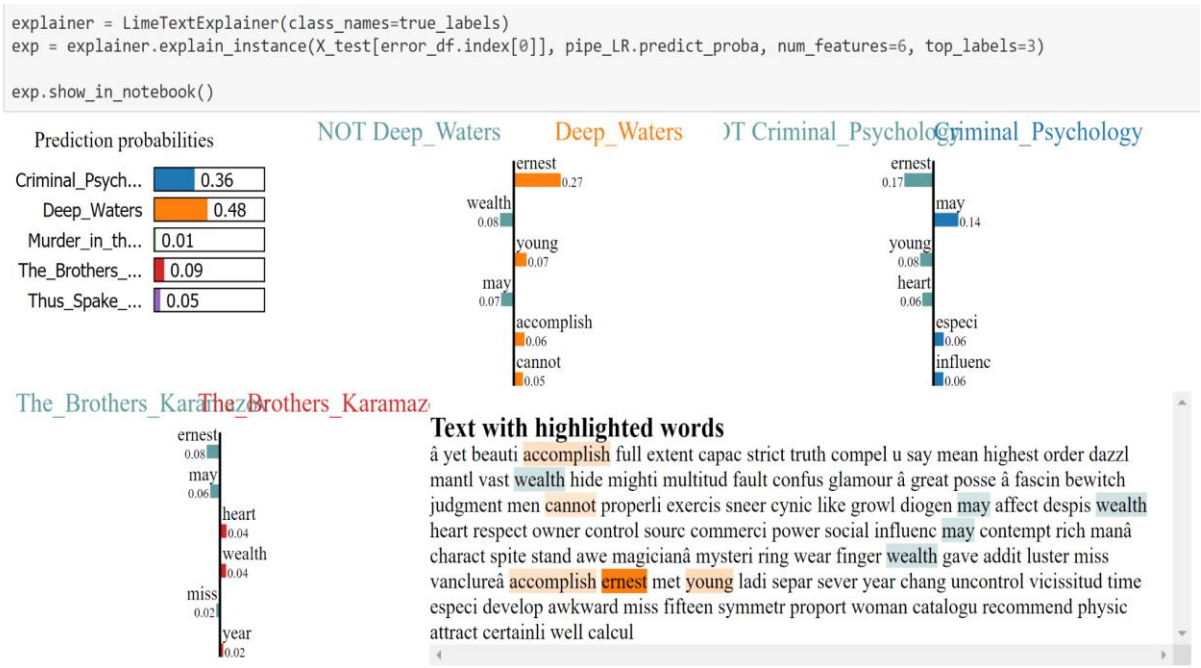


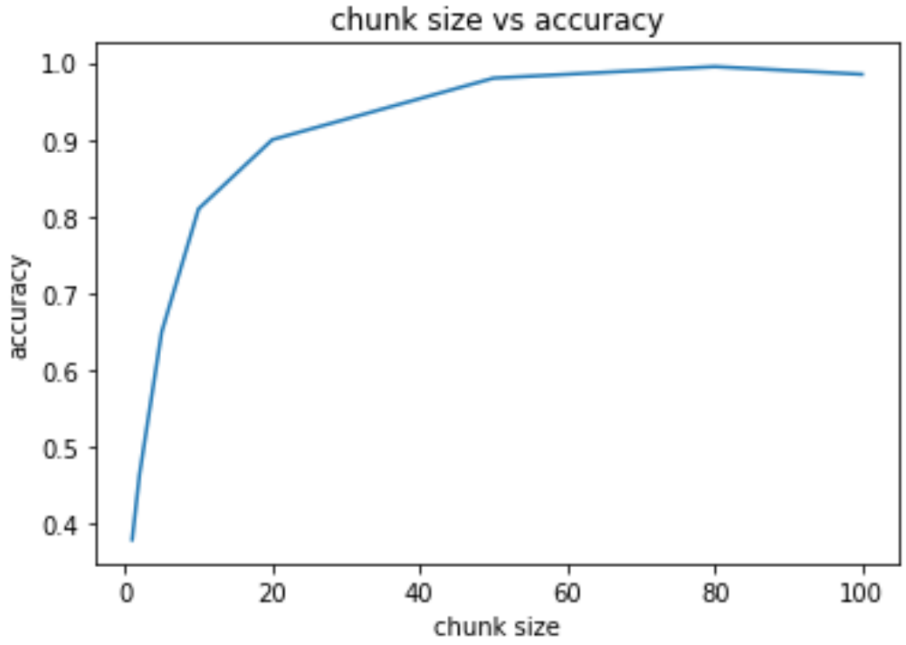
Figure 7: The probability of top classes along with the 6 most effective features to relate or avoid this label, these features mapped on text

In general, whether we use this representation or that, it's quite obvious that certain words are responsible for the bias of the model towards that false class: "Criminal Psychology" over the true one.

5. Factors' Modification (Sample length and size)

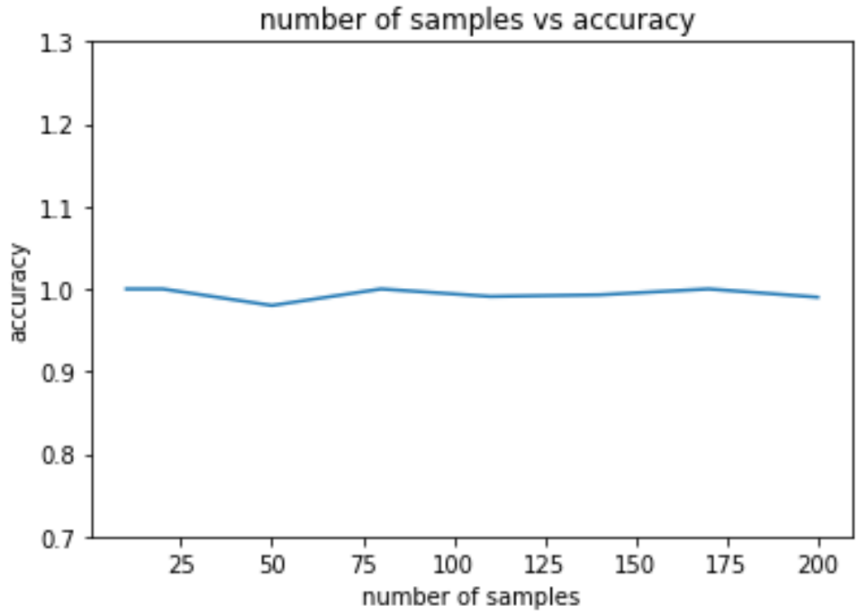
In all the previous experiments and despite the progress that happened in nearly every step, it seems that the accuracy was high enough since the very first model, we say "high" according to the general performance of NLP and text processing tasks where they tend to be lower than that.

This made us curious to play a little bit with the features from the very beginning; changing the length of the sample itself to be less or more than 100 words and observing how the accuracy changes.



The findings were impressive and had a logical interpretation, where as you can see from the graph: The accuracy increases with the increase in the sample length until a certain point where it stops improving as a plateau/saturation point.

In addition to that, we can change the number of samples per document, instead of using 200 samples we can enlarge this number or lessen it while observing the performance.



Namely, changing the number of samples didn't derive any change in the accuracy.

6. Conclusion and Findings

As always told and advised, machine learning is an empirical science, you need to try different combinations of models, transformations and features to check the best performing combination. Visualizations assist extensively in interpreting the predictions of the models which is an essential part to have some knowledge of the next steps and improvements that should be maintained to enhance the solution.

Observing the performance metrics and things like the bias-variance tradeoffs are great indicators of where you're going, accuracy isn't enough anymore especially in this bombardment of algorithms.

6. References

[1] nltk.tokenize package — NLTK 3.6.2 documentation

[2] How to remove all punctuation marks with NLTK in Python (kite.com)

[3] How To Remove Stopwords In Python | Stemming and Lemmatization (analyticsvidhya.com)

[4] Stemming and Lemmatization in Python - DataCamp

[5] NLP in Python- Vectorizing. Common vectorizing techniques employed… | by Divya Raghunathan | Towards Data Science

[6] How to Encode Text Data for Machine Learning with scikit-learn (machinelearningmastery.com)

[7] TextExplainer: debugging black-box text classifiers — ELI5 0.11.0 documentation

[8] Lime - multiclass (marcotcr.github.io)