Sorbonne Université
Sciences et ingénierie
Msc Computer Science: DAC

Natural Language Processing: Automated Vectorization and Classification of Text Datasets

Armand Bonn - 21400077

Course:	TAL
Date version:	31st March 2025

1 Introduction

In recent decades, machine learning models have been increasingly important for various classification and prediction tasks. Even more recently, Artificial Intelligence models and more specifically Language Models (LM) topics have exploded everywhere: research, news media, and business applications. However, what models were used before these famous LMs and before we had the computing power to deploy models with giant architectures? In this report, we will focus on creating and discussing a general pipeline for a Bag of Words (BoW) type of language model, also known as the predecessors of LMs in language processing. The goal is to give a thorough methodology of applying these classical BoW processes to two datasets and discuss the results. First, the methodology of this BoW pipeline will be presented, after which an analysis of the vocabulary of the two datasets will be discussed. Lastly, the results of the classification models in the BoW pipeline will be presented for the two datasets. The aim of this report is to get an in-depth comprehension of how to acquire the best parameters for a BoW vectorizer to obtain the best classification results. Furthermore, the report is accompanied by the classes and code written for the Bag of Words automation pipeline.

The results and model classification are applied on two different datasets: *Movies*, and *Presidents*. The former is a database of movie reviews with the labels positive and negative, hence the classification task is to be able to predict if a movie review is positive or negative (i.e. sentiment analysis). The latter is a dataset made of speeches of two different French presidents, where each observation is a distinct sentence of a speech for a given president. Here, we want to build a classification model to be able to predict which speech is from which president.

2 Methodology Bag of Words Vectorizers

For the entire methodology of the Bag of Words vectorizer pipeline, we built several classes made for Text processing, Vectorizer Tuning, Vocabulary Analysis, and more. These Python classes can be re-used for the vectorization data process of finding the optimal BoW model for class prediction of any biased and unbiased dataset. Only slight adaptions to importing the data have to be made and the remaining of the entire process is afterwards automatic. See also the code corresponding to the report.

2.1 Text preprocessing

Generally, for BoW models, text needs to be processed as we create a matrix representation of the word frequencies in each document. Through processing the documents (i.e. observations) we create a sparser matrix with fewer columns, as we remove irrelevant noise in the classification task such as punctuation, numbers, and special symbols. For the datasets, we used up to 4 different pre-processing methods to better understand the impact of the different methods on the classification tasks.

In Table 1 and 2, the different pre-processing techniques can be found for the two datasets. The first column also named *standard* refers to standard pre-processing which includes removing all considerable noise in the documents including removing: URLs, HTML tags, punctuation, digits, emojis, and special characters. Furthermore, the standard process also expands contractions and puts all words in lowercase. As the documents, in the *Movies* dataset are large texts we also decided to have a process that only includes the first four and last four sentences to understand if it would have any impact on the results of the classification.

As can be seen in Table 1 and 2, stop words are also removed from the documents in the different text processing techniques. However, as the *Presidents* dataset contains text exclusively in French, not English

Process	Name	Standard	Stem	Lemmatize	First n	Last n	Stop words
1	No process						-
2	Stem	X	X				English
3	Lemma	X		X			English
4	Lemma (4,4)	X	X	X	4	4	English

Table 1: Pre-processing methods for *Movies* dataset

stop words are removed but French stop words. The documents in this dataset contain primarily one sentence, hence, it is not possible to select any first or last sentences as in the *Movies* dataset. Hence, we obtain the different pre-processed documents for each dataset and will apply the classification tasks to each different process. In summary, processes can be distinguished by whether the standard techniques were applied and whether stemming or lemmatization was applied to the words in the documents.

Process	Name	Standard	Stem	Lemmatize	First n	Last n	Stop words
1	No process						-
2	Stem	X	X				French
3	Lemma	X		X			French

Table 2: Pre-processing methods for *Presidents* dataset

2.2 Vocabulary analysis

After we apply the processing to the documents of each dataset, it is important to better understand the vocabulary of the respective datasets. Moreover, we also want to be more aware of how the parameters of the BoW vectorizers affect the vocabulary subsequently used for the classification. Hence, we built a *VocabularyAnalysis* module, which takes in the collection of documents for the datasets and visualizes the initial vocabulary and the impact of changing the vectorizer parameters. It is important to analyze this impact as our goal in this report is to give a robust BoW methodology for using vectorizer models in Natural Language Processing.

The most important input parameters to a vectorizer are given in Table 3. These variables have a big impact on the final vocabulary used for training and predicting during the classification task. The *max_df* parameter removes terms that are frequent in all documents, these words do often not add any information to distinguishing the different classes as they are present in the majority of documents.

On the contrary, the *min_df* parameter removes words which present in very few documents. These words are very specific towards for example a single document but do not add any class-specific information as they are not frequent enough. The *max_features* parameters reduce the feature spaces by only selecting the terms that are the most frequent. Reducing the feature space can also contribute to better predictions as the vocabulary size is expected to be very large, and fixing the feature space will prevent over-fitting and a sparse matrix representation. Lastly, the *ngram_range* parameter can help the classifier acquire a deeper understanding of a text by increasing the range of words considered. The only issue with the n-gram parameter is the fact that it increases vocabulary size significantly. In the *VocabularyAnalysis* module, we will visualize how these parameters can affect the word frequency.

Variable	Description
max_df	Ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words).
min_df	Ignore terms that have a document frequency strictly lower than the given threshold.
max_features	Build a vocabulary that only considers the top max_features ordered by term frequency across the corpus.
ngram_range	Defines the lower and upper boundary of the range of n-values for different word n-grams.

Note: For more details, visit Scikit-learn CountVectorizer.

Table 3: Descriptions of Important BoW Vectorizer Input Parameters

2.3 Evaluating models and text processes

After having preprocessed the texts and analyzed the vocabulary, we move on the evaluate different classification models for the documents. In this report, we focus on three classical machine-learning classification models: Naive Bayes (NB), Logistic regression (LR), and Support Vector Machines (SVM). The goal of the evaluation phase is to find the best vectorizer with optimized parameters for the classification task of each dataset. In this section, we will provide the methodology for getting the optimized models and vectorizer parameters. Before evaluating the models, we split the data into a Train/Validation set and a Test set. The Train/Validation set is used to tune the vectorizer parameters to acquire the best model and parameters. The test set will not be used until we acquire the optimal parameters, after which we apply a final test prediction to understand if which optimal model performs the best. The general evaluation pipeline can be found in Figure 1.

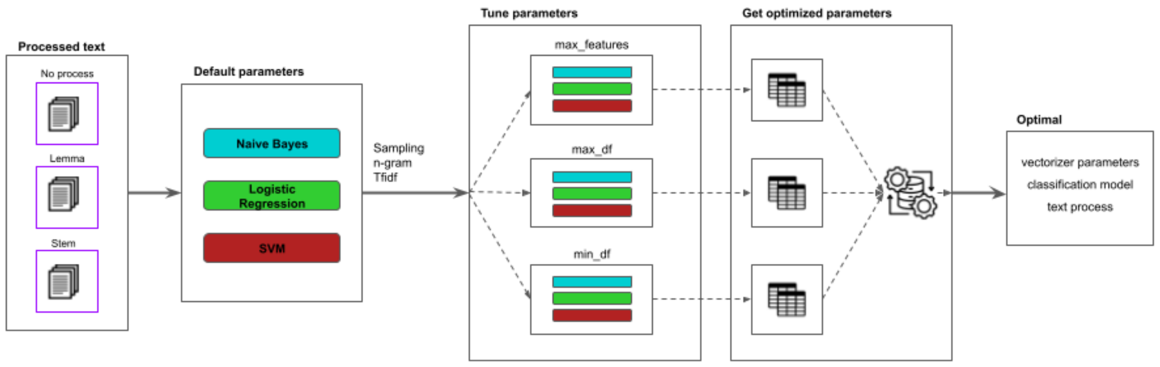


Figure 1: Pipeline for vectorizer and model evaluation to acquire optimal parameters for classification

2.3.1 Initial results

The pipeline in Figure 1 was applied for three different types of vectorizers to acquire the optimal result for each type, namely: (i) CountVectorizer, (ii) TfidfVectorizer, and (iii) TfidfVectorizer with N-gram. For a given vectorizer, we take each respective processed set of documents and fit and evaluate each model with the default parameters of the vectorizer. This classification is applied (for the biased *Presidents* dataset) using oversampling and undersampling. As the dataset is biased, we need to rebalance the observations such that the models will not overfit the majority class. The specific sampling methods used are RandomUndersampling for undersampling and the popular *SMOTE* method for oversampling. For vectorizers (ii) and (iii), we also apply it with different *tf* and *idf* parameters. Lastly, for vectorizer (iii) we also acquire classification performance for different N-grams. Using these initial results, we acquire the optimal sampling method, for (i) & (ii) Tfidf-specific parameters, and for (iii) the optimal N-gram.

2.3.2 Tuning vectorizer parameters

After acquiring these parameters, we move on to tuning the vectorizer parameters which were defined in Section 2.2. Here, to find the optimal parameters we apply a grid-search-like method where we loop over different values of a parameter keeping the other two at their default values. Our assumption here is that the parameters have an independent effect on our model performance, which is unlikely to be true. However, doing an extensive grid search on three dimensions for the three vectorizer parameters is too computationally expensive. Hence, this method is one of the most efficient ways to acquire sub-optimal parameters without having to run models for several days. Throughout the tuning, we saved the mean accuracy or f1-score of each model of the NLP cross-validation method (see Algorithm 1). When evaluating a vectorizer with a corresponding classification model, we need to fit and transform only the training data. Otherwise, the vectorizer will also create columns for words not present in the training data, which will lead to overfitting. This is our justification for building our own cross-validation function. Moreover, it was also necessary to have an extra option for evaluating models with F1 Score as the *Presidents* dataset is biased. The F1-score used during tuning and for final model evaluation is the weighted F1-score. This measure computes the F1-score of both classes (i.e. y=0 and y=1) and weights it by the size of each respective class.

Algorithm 1 Vectorizer Cross-Validation

```
1: Initialize $kf$ as cross-validation with $cfold$
2: Initialize empty list $scores$
3: for each $(train\_idx, test\_idx)$ in $kf$ do
4:     $X\_train\_texts$, $X\_test\_texts$, $y\_train$, $y\_test$
5:     $X\_train \leftarrow vectorizer.fit\_transform(X\_train\_texts)$
6:     $X\_test \leftarrow vectorizer.transform(X\_test\_texts)$
7:     $(X\_train, y\_train) \leftarrow balance\_dataset(X\_train, y\_train, undersampling, oversampling)$
8:     Train $classifier$ on $(X\_train, y\_train)$
9:     Predict $y\_pred$ for $X\_test$
10:    if $undersampling$ or $oversampling$ then
11:        $score \leftarrow F1\text{-}score(y\_test, y\_pred)$
12:    else
13:        $score \leftarrow Accuracy(y\_test, y\_pred)$
14:    end if
15:    Append $score$ to $scores$
16: end for
17: return $scores$
```

2.3.3 Acquiring final optimal model

Lastly, when all model results are saved for each corresponding parameter value of the vectorizer we need to retrieve the optimal vectorizer parameters, model, and process. First, we try and acquire which machine-learning model and text process give maximum prediction results. We take the maximum prediction score for each respective model and process of the tuned parameters. Then these maximum results are averaged over all tuned parameters and text processes to obtain which model gives the best performance on average. To obtain the best text process, we make use of the computed average results and the best model to also retrieve the text process which gives the maximum result for the given model. The algorithm for selecting the best model and text process can be found in Algorithm 2. After the best model and text process for a given vectorizer is computed, we use these results to acquire the three optimal tuned parameters for these inputs.

Algorithm 2 Optimal Model and Text Process

```
Require: Tuning results: $tune\_max\_features$, $tune\_max\_df$, $tune\_min\_df$
                                        ▷ Size best_... matrix: (#text process, #models)
1: $best\_max\_features \leftarrow$ RESULTS_BEST_SCORES($tune\_maxf$)
2: $best\_max\_df \leftarrow$ RESULTS_BEST_SCORES($tune\_max\_df$)
3: $best\_min\_df \leftarrow$ RESULTS_BEST_SCORES($tune\_min\_df$)
                                        ▷ Stack tuning results and compute mean scores
4: Combine $best\_maxf$, $best\_max\_df$, and $best\_min\_df$ into a single stacked array
5: $final\_tune \leftarrow$ element-wise mean across the three tuning results
                                        ▷ Determine Best ML Model
6: Compute the mean score per model with $final\_tune$
7: $best\_model \leftarrow$ model with the highest mean score
                                        ▷ Determine Best Text Process
8: $best\_process \leftarrow$ the best model's highest score
9: return Best model and best process
```

Finally, the optimal model, vectorizer parameters, and text process will be obtained for each vectorizer type (i) CountVectorizer, (ii) TfidfVectorizer, and (iii) TfidfVectorizer with N-gram. Note again that the pipeline given in previous sections and illustrated in Figure 1 needs to be applied on every vectorizer type (i)-(iii). Now that all the tuned parameters are gathered, we use these results to apply a final round of transformation and training. However, now we test the model prediction scores on the still **unseen** test data in order to get the final results of our tuned models. This will provide us with a good indication of how the final tuned vectorizer and models will perform with real-life unseen data. This will be the final part of evaluation for the three vectorizer models and the tuned vectorizer type which gives the best results on the unseen test set will be selected for future predictions on unseen data.

This concludes the entire vectorizer machine-learning classification pipeline in this report. Now the results will be presented on the two datasets *Movies* and *Presidents*.

3 Results

3.1 Visual analysis vocabulary

As discussed previously, first we will visualize some key characteristics of the documents for each dataset by analyzing their vocabulary, and vocabulary size. In Figure 18, the word clouds for different text processes (*Presidents*) are illustrated to give a better understanding of the impact of the text preprocessing. As expected, the word cloud for documents with no process contains stop-words and unfiltered text like HTML tags and letters with an accent. Furthermore, we also see how stemming and lemmatization process words differently (e.g. the word 'plus' in the word cloud).

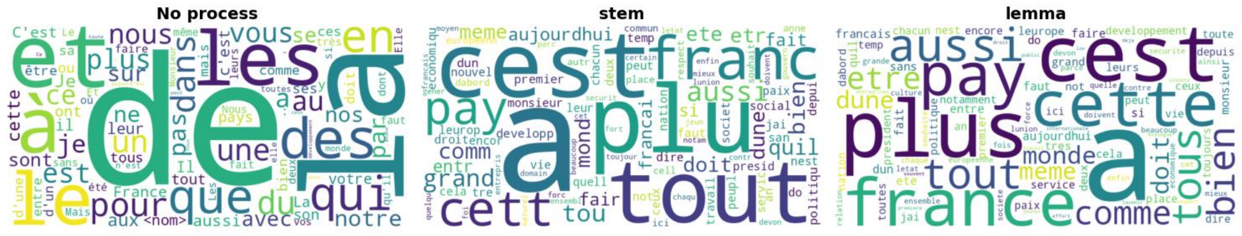
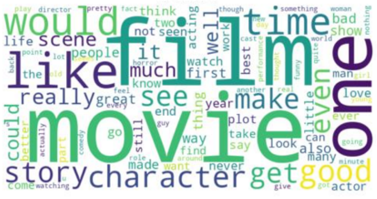
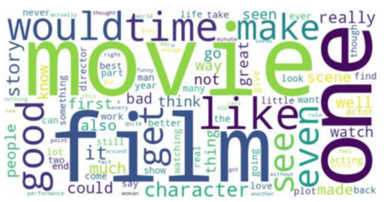


Figure 2: Word frequency word clouds for each text process (*Presidents*)

Furthermore, we also created word clouds based on different frequency criteria, in Figure 3 we presented the three-word clouds for the lemmatization process of the *Movies* dataset. The words appearing in Figure 3a and 3b, show that words with high frequency also appear in most documents. Furthermore, we can also see that the discriminating words for each class represented by the odds-ratio, and also shown in Figure 3c, are not similar to the words in the other two word clouds. This also implies that it could be important to tune the parameter *max_df* of the vectorizer to remove corpus-specific stop words.


(a) Word Fequency


(b) Document Frequency

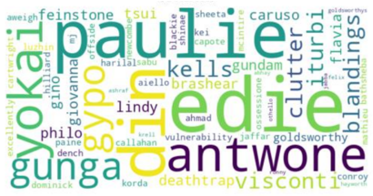
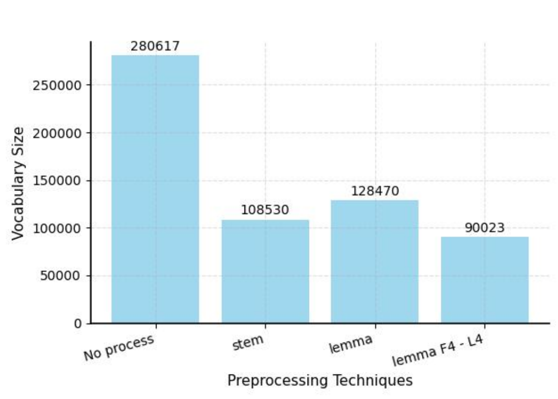

(c) Odds-ratio

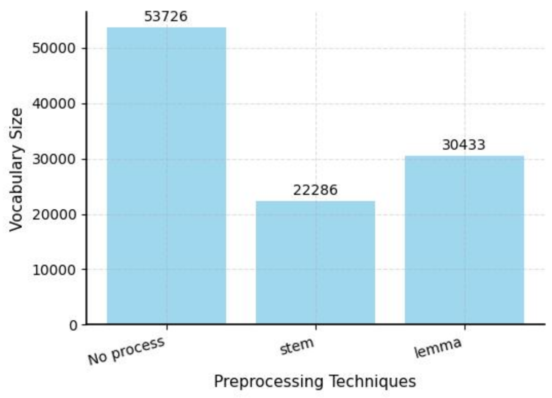
Figure 3: Word clouds for lemmatization text process *Movies* dataset.

In Figure 4, the different vocabulary sizes are visualized for each text pre-processing technique. The figure clearly shows that preprocessing the text has a very large effect on the number of words in the corpus. As punctuation and other noise are not removed when the text is not processed, the vocabulary size is approximately double as large for both datasets. Furthermore, we also see that lemmatizing the text results in a slightly larger vocabulary for both datasets. This could also mean that using lemmatization does not reduce the text vocabulary too much, contrary to the stemming pre-process, and thus could imply better model performance. This is interesting to note and keep in mind when analyzing the model results.

In Figure 5, we also presented how the vocabulary size varies for different n-grams. We varied the (1-i) N-grams and also the (i-i) N-grams. The bar plot shows that incrementing the N-grams results in an almost exponential growth of the vocabulary with the highest size reaching more than $1 * 10^7$. Having such a large vocabulary will in a huge number of dimensions which is unwanted and not manageable. Hence, in the further analysis, we only considered N-gram ranges up to (1,3). Moreover, we also see that the (i,i) N-grams have a smaller vocabulary size relative to the the (1,i) range, which is expected as the former will give a subset of the words constructed by the latter.



(a) *Movies* dataset



(b) *Presidents* dataset

Figure 4: Vocabulary size for different text pre-processes.

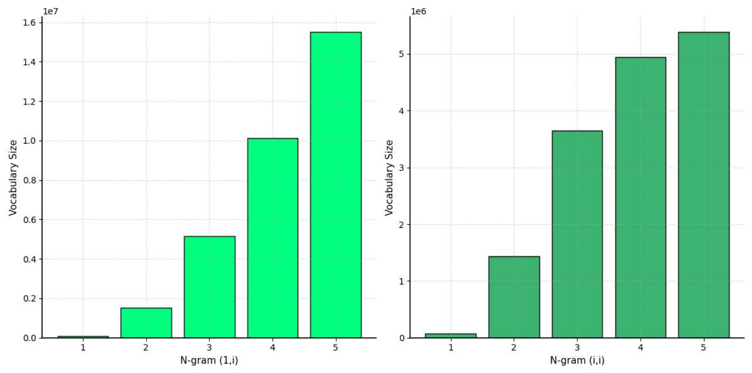


Figure 5: Vocabulary sizes with different N-gram ranges with no pre-process (*Movies*).

In Figure 6, we also illustrated how varying the vectorizer parameter affects the vocabulary and its distribution. The effect and the subsequent effect on the constructed matrix representation give us an indication that varying these vectorizer parameters will have an impact on our model performance. The parameters are also described in Section 2.2. From left to right, increasing the *min_df* results that the low-frequency terms being filtered out, increasing *max_df* results that the frequency of the highest-frequency term decreasing, and decreasing *max_features* results in decreasing terms in the vocabulary.

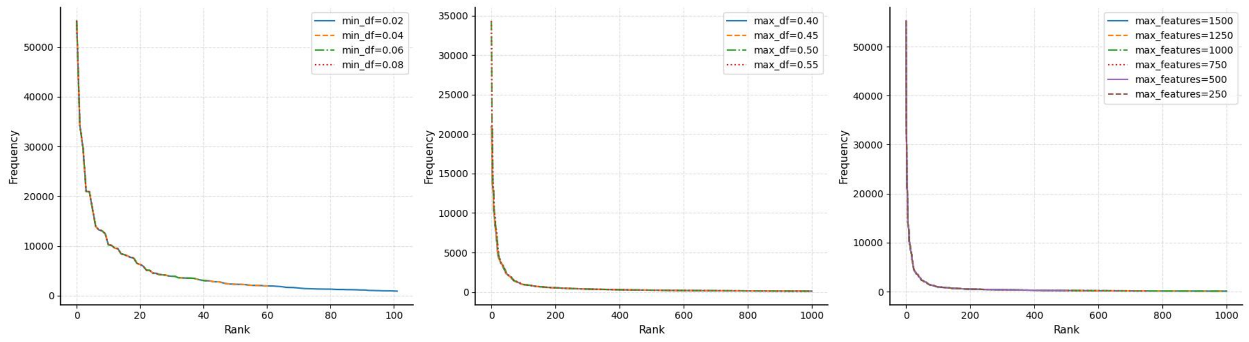
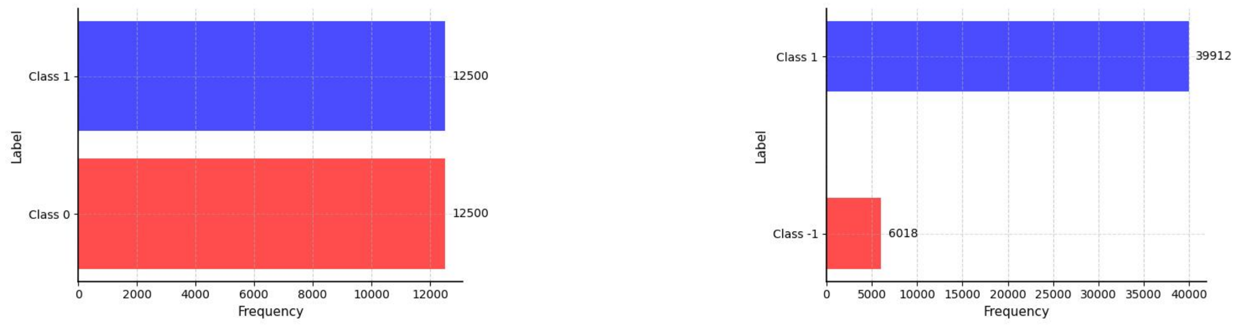


Figure 6: Frequency distribution of 100 most frequent words with varying vectorizer parameters (*President*).

Lastly, we also illustrated the distribution of the prediction class in Figure 7. The *Movies* dataset has a balanced distribution of its label predictions classes, however, it is clearly not the case for the *Presidents* dataset. This is necessary to illustrate the need to apply undersampling and oversampling techniques to balance the *Presidents* dataset. Furthermore, it also supports our method in section 2.3.2 which we computed F1 scores for model performance. Otherwise, the model will be biased towards the majority class which we want to avoid.



(a) *Movies* dataset (b) *Presidents* dataset

Figure 7: Distribution of the two prediction classes for the datasets.

3.2 Model results vectorizers

3.2.1 Initial model performance

As explained in Section 2.3.1, for the three vectorizer types (i) CountVectorizer, (ii) TfidfVectorizer, and (iii) TfidfVectorizer with N-gram, we first apply predictions to understand which parameters to use apart from the vectorizer parameters that will be tuned. One of the important parameters for the biased *Presidents* dataset is the sampling method to balance the dataset during training. As shown in Figure 8, the oversampling method gives on average almost 5% higher F1-score using the default parameters for the vectorizer. This can also be explained because the minority class has very few observations and the undersampling consequently removes too many observations of the majority class for the models to still perform well. Generally, for the defined vectorizers (i) - (iii), the oversampling method always seems to give the best initial results. For the *Movies* dataset, this parameter did not have to be chosen as the data is balanced.

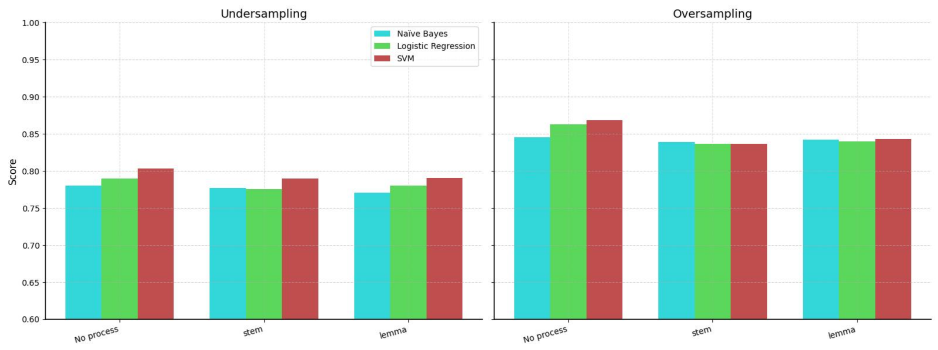


Figure 8: Model F1-scores with undersampling and oversampling using Tfidf-vectorizer (ii) with default parameters (*Presidents*).

Moreover, for the vectorizer types (ii) TfidfVectorizer, and (iii) TfidfVectorizer with N-gram, also the term weighting and transformation parameters *(use_idf, smooth_idf, sublinear_tf)* need to be chosen for optimal model results. To acquire the best parameters for these vectorizers, we applied the Vectorizer Cross-Validation (see Algorithm 1) with the different weighting parameters. The results for differing Tfidf-vectorizer parameters for the SVM classification model are shown in Figure 9. Using these prediction performances, we also select the optimal initial Tfidf parameters for vectorizer types (ii) and (iii) of each dataset. In Figure9, it can be seen that the variations of the Tfidf parameters do not give significantly different results for the SVM model on the *Movies* data. Lastly, we also apply the same method to acquire the best-performing N-gram for vectorizer type (iii) TfidfVectorizer with N-gram (see Figure 20 in the Appendix). Using all these initial results, we have all the optimized default parameters for each vectorizer type to start applying the tuning to the other parameters.

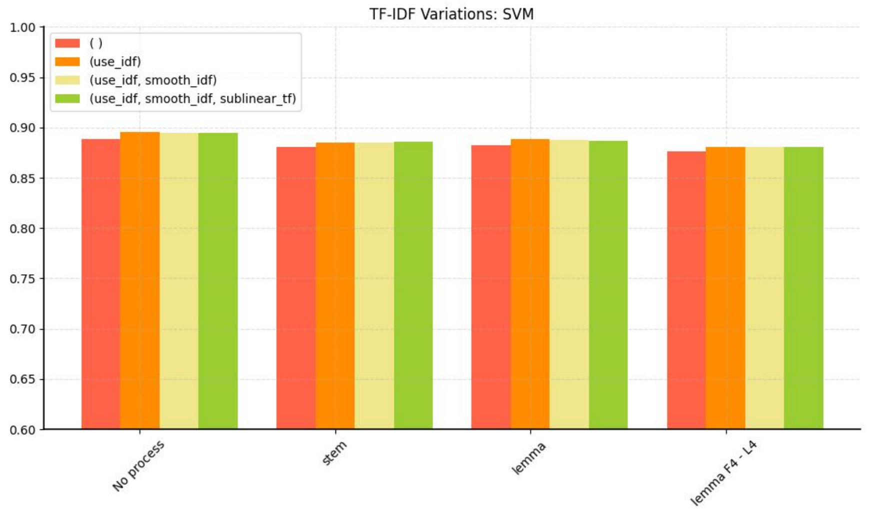


Figure 9: SVM Accuracy with differing Tfidf parameter values using Tfidf-vectorizer (ii) with default parameters (*Movies*).

3.2.2 Parameter tuning

As explained in Section 2.3.2, within the Bag of Words Processing Pipeline we perform a parameter tuning for the general vectorizer parameters. These parameters have an impact on the Bag of Words representation matrix of the texts and consequently on the model results. Hence, in this section, we will provide some explanations behind some results of the vectorizer tuning. More detailed graphs of the tuning results for the different vectorizers and datasets can be found in the Appendix.

In Figure 10, we illustrated the model accuracy results for varying *max_df* values on (i) CountVectorizer with *Movies* dataset. Here, we can first remark that the text pre-processing is only selecting the first 4 and last 4 sentences gives less high model results than other pre-process (this observation is true throughout the whole model performance pipeline). This implies that there is not enough "sentiment" specific information in these sentences for the movie reviews. Furthermore, the Figure and also other model performances using (i) CountVectorizer show that the Logistic Regression (LR) model is the most robust with these Bag of Words vectorizers. The CountVectorizer transforms the dataset of text documents into a matrix with word frequencies, hence not being very adaptable for SVM models, who do not perform as well in classification if the representation data is not normalized.

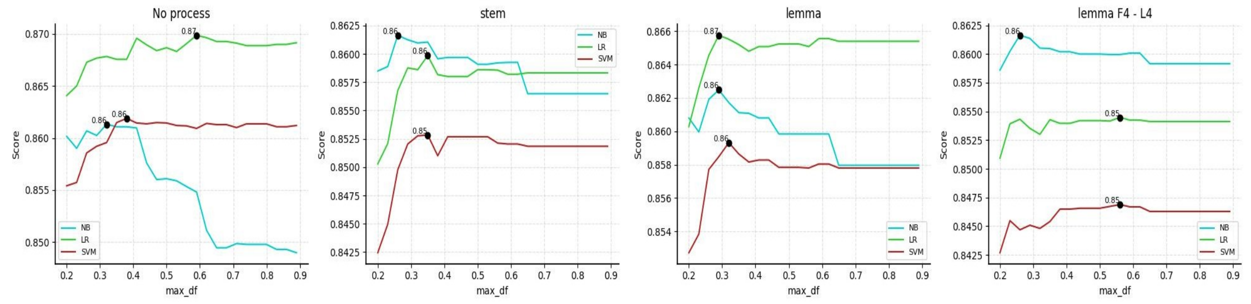
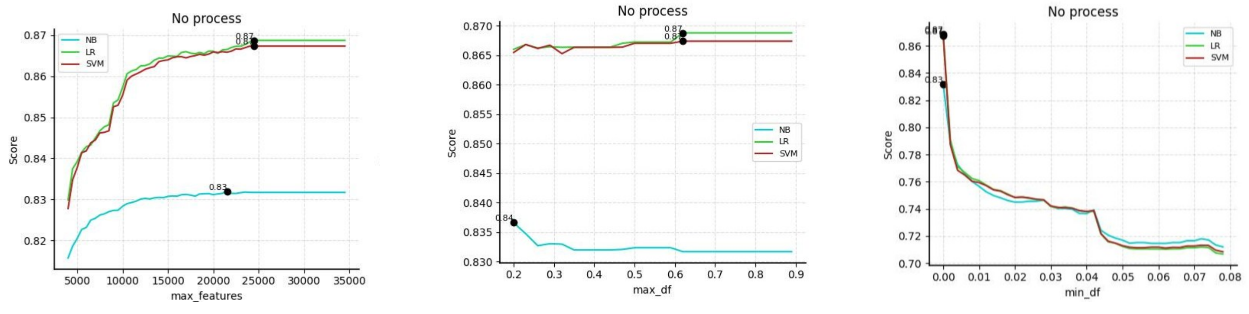


Figure 10: Model accuracies for different text processes with *max_df* parameter tuning on (i) CountVectorizer (*Movies*).

Furthermore, in Figure 11 we illustrated the tuning results of the (ii) TfidfVectorizer for documents without processing. One of the interesting results shown in Figure 11c is that the prediction performance deteriorates very quickly with the president's dataset performance when increasing the *min_df* parameter. This is also due to the fact that the Vocabulary size is relatively less big compared to the *Movies* dataset. Hence, we see that even document-specific words can add to the prediction results for the *Presidents* dataset which has documents of relatively small size and a small vocabulary size. Furthermore, we also can see that the simple Naive Bayes (NB) model performs relatively much worse compared to the LR and SVM. This can be due to the fact that the feature independence assumption is too strong for the vectorization representation of documents. The underperformance of the NB model relative to LR can also be seen again in Figure 10. This trend is consistently evident across all model performance results.

Lastly, Figure 11b also shows in more detail at which point of removing corpus-specific stop words the model performance stagnates. At around a value of 0.70 for *max_df* the model performance in not affected anymore. In other words, removing words that are present in more than 70% of the documents does not affect the performance of the LR and SVM models.



(a) *max_features* (b) *max_df* (c) *min_df*

Figure 11: Vectorizer F1-scores model results of (ii) TfidfVectorizer for documents with no processing (*Presidents*).

In the Appendix, more detailed graphs of the tuning results can be found for the vectorizer types that gave the best results. There the results show again that increasing *min_df* always results in the worst performance. Furthermore, also the results for the *Movies* dataset with N-grams show that increasing the *max_df* parameter does not result in better model performance. Also implies that the prediction decision is not influenced by corpus-specific stop words. Lastly, Figure 11a and 21 show that at a specific threshold, the *max_features* variable does not influence the model performance. This is due to the fact that the vocabulary size of the *Presidents* dataset is relatively small. Hence, after performing the split of training and validation data, Figure 11a clearly shows that the training data on which the vectorizer is fitted and transformed does not exceed a vocabulary size of approximately 25,000. This is not the case for the *Movies* dataset where the vocabulary size is much larger. There the results consistently show that increasing the *max_features* variable leads to better prediction performance.

3.2.3 Results optimal vectorizer and models

After having performed the grid-search-like tuning for the parameters of the vectorizers, we obtain the parameters from the model and text-preprocess which gives us the maximum prediction results (see Algorithm 2). The final optimal vectorizers and classification models are shown in Table 4. One of the more remarkable results that we did not expect is the fact that the text-corpora with no pre-processing (id=0) always gives the most optimal results. This could be due to the fact that our standard preprocessing techniques which remove all considerable noise from the text corpus might reduce too much information. In turn, this loss of information seems to lead to giving the model not enough class-specific "noise". Furthermore, it is also true that tuning the vectorizer parameters is also of type of preprocessing as the vectorizer excludes corpus-specific stop words and only retains a specific number of max_features. Moreover, as also seen in Section 3.2.1, the oversampling method always seems to give better results. This is also due to the large imbalance in the different classes.

Dataset	**Vectorizer**	**Process**[1]	**Model**	**max_features**	**max_df**	**min_df**	**ngram**	**Sampling**
Movies	Count	0	LR	34500	0.59	0.012	(1,1)	-
	TF-IDF	0	SVM	34500	0.62	0.00	(1,1)	-
	N-gram	0	SVM	39000	0.20	0.00	(1,2)	-
Presidents	Count	0	NB	24500	0.62	0.00	(1,1)	Over
	TF-IDF	0	LR	24500	0.62	0.00	(1,1)	Over
	N-gram	0	SVM	44500	0.62	0.00	(1,3)	Over

[1]See table 1 and 2 for text processes.

Table 4: Optimized Vectorizer and Model Configuration Across Datasets

Using the results given in Table 4, we used the optimal vectorizer and model configuration to make a final prediction on a still unseen test set of the data. This test data has not been used in any part of the vectorizer configuration and is now employed to evaluate the optimal models. For the *Movies* dataset we already disposed of a test set of 25,000 observations, the same size as the train/valid set. This was not the case for *Presidents* dataset, here we chose a 80/20 split resulting in a test size of approximately 11,0000 observations. For this test set, we ensured that the labels have the same distribution as in the training set. The prediction results on the test sets are shown in Table 5. As the classes are balanced for the *Movies* dataset, we see that the performance metrics are approximately the same for each independent vectorizer type. The results show that the N-gram optimized vectorizer type gives the best performance on the *Movies* dataset. With an overall accuracy of 89% using a test set of size 25,000, it seems to give quite competitive results after the tuning configuration.

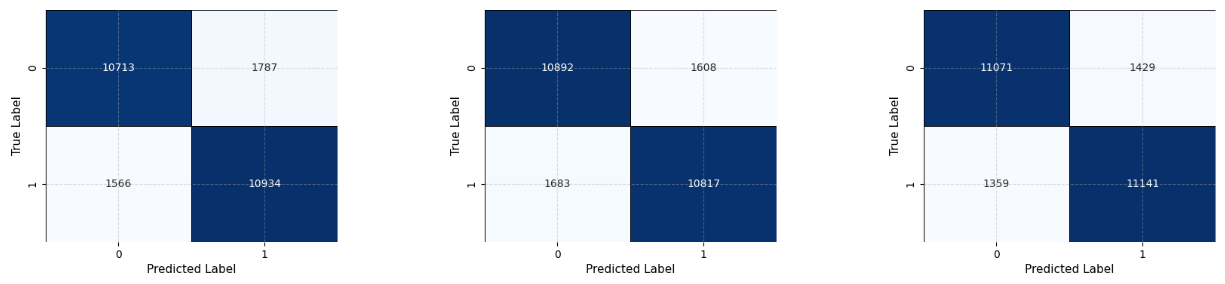
For the *Presidents* dataset, the Count vectorizer gives the worst performance on all metrics. Furthermore, the models seem to give the worst performance in their recall metric. Also means that it predicts False Negatives more often than we would like to, this could also be due to the oversampling. It could be interesting to look more thoroughly into other more advanced sampling and regularization techniques to understand if it leads to better results for these models with the biased dataset.

Dataset	**Vectorizer**	**Accuracy**	**Precision**	**Recall**	**F1**
Movies	Count	0.866	0.866	0.866	0.866
	TF-IDF	0.868	0.868	0.868	0.868
	N-gram*	0.888	0.888	0.888	0.888
Presidents	Count	0.857	0.878	0.857	0.865
	TF-IDF*	0.867	0.886	0.866	0.874
	N-gram	0.863	0.878	0.863	0.869

*Optimal configured vectorizer and model for respective dataset.

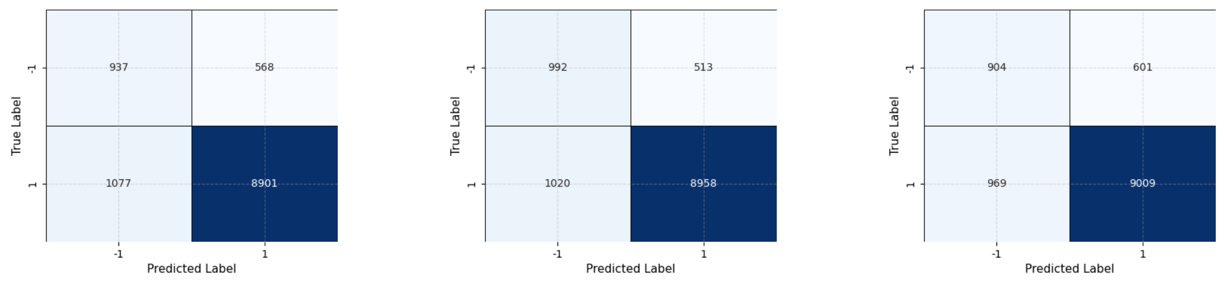
Table 5: Prediction results optimized Vectorizer and Model

Lastly, to get a better understanding of how these models perform we also visualized the different confusion matrices for the optimal vectorizer of each dataset (Figure 12 and 13. As also seen in the performance results in Table 5, the model predicts both classes in *Movies* are equally as good for all vectorizer types. Furthermore, we can also better understand through Figure 13 that the configured models often predict label -1 (minority class), while it should be the positive class +1. Once again, we believe this could be due to the oversampling, and more advanced techniques should be explored.



(a) Count Vectorizer (b) Tfidf Vectorizer (c) Tfidf Vectorizer w/ N-gram

Figure 12: Confusion matrices predictions on test sets for optimal vectorizers and models (*Movies*).



(a) Count Vectorizer (b) Tfidf Vectorizer (c) Tfidf Vectorizer w/ N-gram

Figure 13: Confusion matrices predictions on test sets for optimal vectorizers and models (*Presidents*).

Finally, we used the optimal vectorizers as shown in Table 5 (*) with their tuned vectorizer parameters (Table 4) to perform the predictions on the datasets with no labels. These predictions were then submitted in the NLP model competition for the *Movies* and *Presidents* dataset. The metric results show a 99.2%

accuracy with the unlabeled *Movies* set and a 55.3% F1-score for the minority class. Hence, these results give also a better understanding of strengths and possible improvements in the current Vectorizer pipeline. These will be discussed in the next section.

4 Conclusion and Discussion

To summarize, in this project and report, we built an automated pipeline for evaluating the best BoW vectorizer type for a given text classification task. The report made use of two datasets, *Movies* and *Presidents*, where the task of the former is to classify the sentiment of movie reviews (i.e. positive or negative). The classification task of the latter dataset is to be able to classify the speeches of two possible French presidents. The vectorizer pipeline first starts by processing text using different pre-processing methods. Afterward, we performed an extensive analysis of the vocabulary and the important words in each respective dataset. There, we also included visualizations to understand the impact of varying the vectorizer parameters on the dataset vocabulary. These results also indicated how tuning these parameters could lead to more optimal model performance as the vocabulary and matrix representation change. Subsequently, we tuned the parameters using different classification models to acquire the most optimal vectorizer and model for the classification task. Three different vectorizer types were tuned using the model results of the vectorizer cross-validation method also introduced in the methodology of this report. Using these optimal vectorizer and classification models, the optimal vectorizers were used for prediction on a final unseen test set to understand its final performance.

In the parameter tuning, we remarked that the results concisely showed that the Logistic Regression model performs the best with the Count Vectorizer. This was also explained due to the fact that the matrix representation resulting from the Count vectorizer is not normalized. The Tfidf vectorizer has often been shown to give optimal results in combination with the SVM classifier model. Moreover, the tuning results also consistently showed across vectorizer types and datasets that increasing the *min_df* parameter leads to model performance diminishing rapidly. We can conclude that the words that are not present in most documents are useful for the model to determine discriminating factors between the two classes. Hence, removing these words leads to rapidly decreasing results. In our data analysis section, we also remarked on the fact the *Presidents* dataset had a large class imbalance. Hence, during the processing pipeline, we used sampling techniques to re-balance the class such that the fitted models were not biased towards the majority class. We also used the weighted F1-score to evaluate to model performance. The model results with the different vectorizers showed that oversampling always gave more optimal prediction performance relative to undersampling. As the minority class only represents around 15% of observations, undersampling leads to removing too many observations and consequently leading to poorer performance results. The results also showed that the text documents with no pre-processing gave the most competitive results for the two given datasets. This could be due to the fact that our pre-processing removes all considerable text noise and only keeps the most basic text form. This apparently removes too much "corpus-specific" noise and hence leads to poorer model performance. Furthermore, we also argued that tuning the vectorizer parameter is a type of pre-processing on itself as it removes for example corpus-specific stop words.

Lastly, we used the optimally tuned vectorizers to perform prediction on the unseen test set. The Tfidf vectorizer and N-gram with the SVM model seemed to give the best results on the test set of 25,000 observations reaching almost 90% accuracy on the *Movies* data. Furthermore, the standard Tfidf vectorizer with tuned parameters combined with the Logistic Regression gave more optimal weighted F1-Scores for the

Presidents test set. Our tuned models also seemed to give very high performance on the unlabeled test-set of *Movies*, submitted in the model competition, reaching an accuracy of 99.2%. However, in the *Presidents* submission the model performed an F1-score of 55.2% for the minority class, approximately 10% less than the best model. A current lack in our processing pipeline is the fact we make use of the weighted F1-Score to find optimized vectorizer and model calibrations. Hence, giving not a very competitive result for the *Presidents* data with minority class. For further investigation, it would be interesting to use different F1-scores to see how it affects the performance on very unbalanced datasets such as *Presidents*. The results also indicate that the classical BoW-type vectorizer with standard models is not powerful enough for text datasets with large imbalances. Hence, an interesting further research topic would be to compare our constructed method with more sophisticated transformer models for the *Presidents* data. However, the vectorizer pipeline does seem to give very competitive and high-quality prediction results for balanced datasets such as for the *Movies* dataset. Through this automated vectorization and model tuning pipeline, a user can give as input any type of text dataset with corresponding labels, and the optimal vectorizer, text process, and model classifier will be chosen.

Appendix

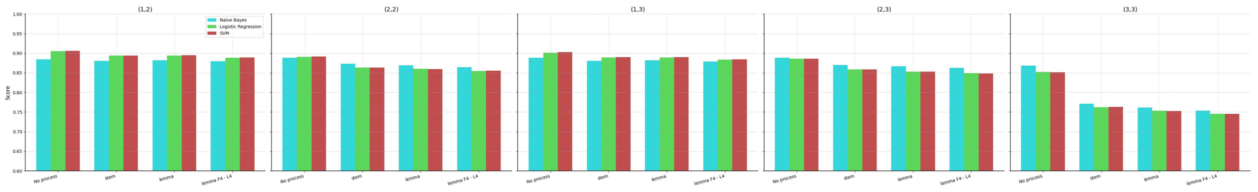


Figure 20: Model Accuracy with different N-grams using Tfidf-vectorizer (iii) with default parameters (*Movies*).

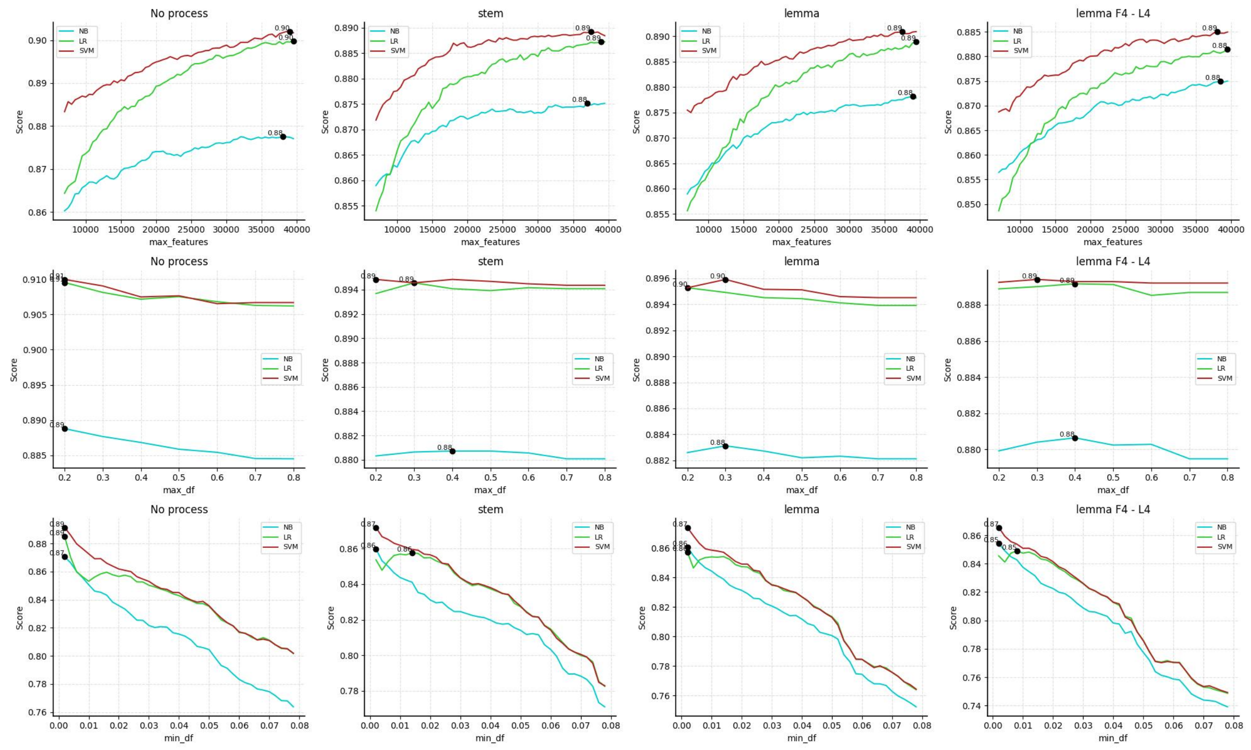


Figure 21: Parameter tuning results for (iii) Tfidf vectorizer with N-gram *Movies* dataset.

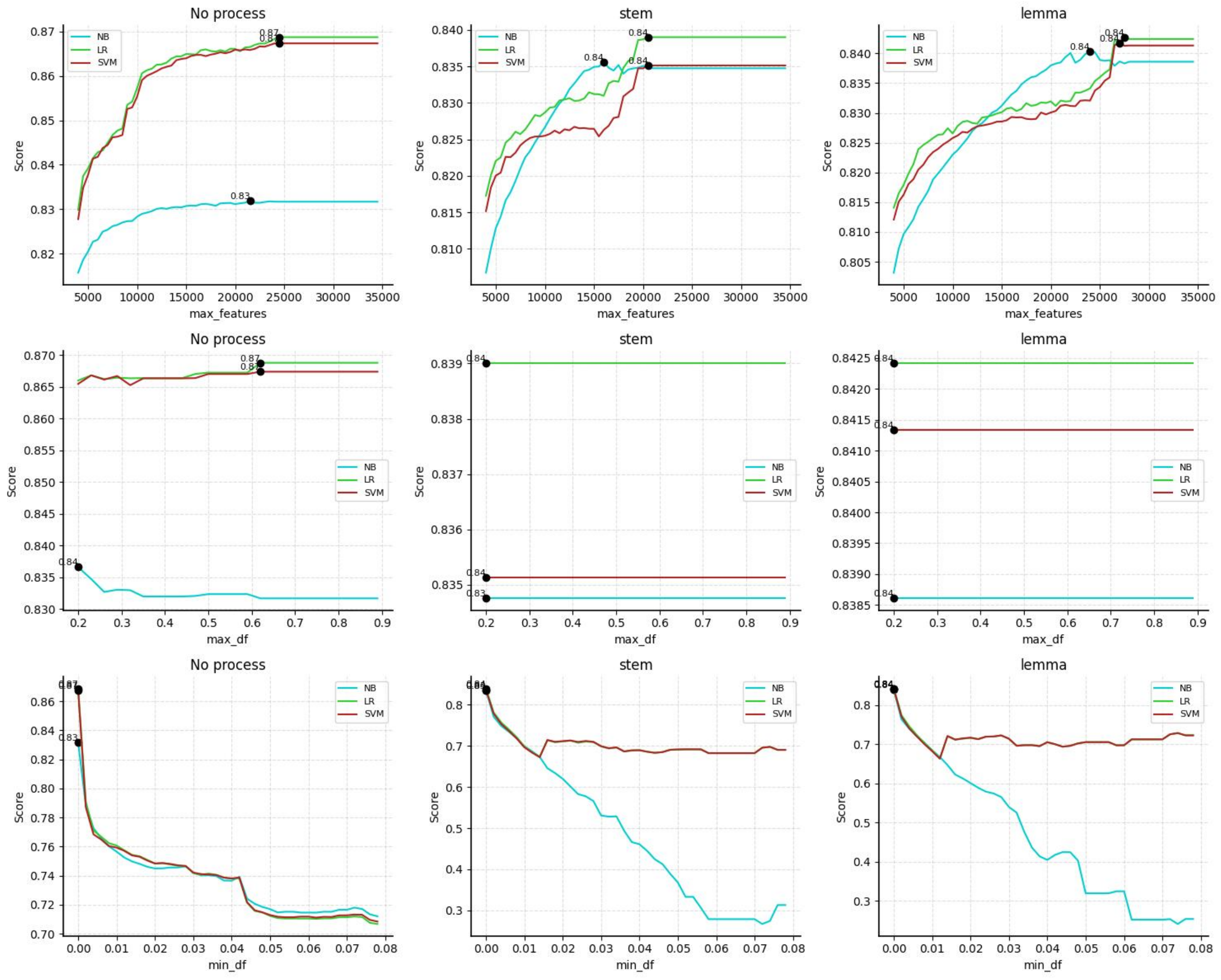


Figure 22: Parameter tuning results for (ii) Tfidf vectorizer *Presidents* dataset.