STATS 201B Final Report
Machine Learning Methods for Text Classification

Nicklaus Kim and María Elisa Ramos-Sepúlveda

March 18 2022

1 Introduction

Since the beginning of digital data, the easiest way to classify data is based on numeric keys. Nevertheless, with over two decades of digitized data, classification is complicated, especially when it comes to natural language data. Hence, the need for a classifier that identifies data based on text has become more evident in recent years. Text classification is a popular application of machine learning techniques that helps classify natural language data into different categories of interest to organize and clean databases. The purpose of this study is to test different text classification methods, evaluate their accuracy, and compare them to one another. Moreover, we aim to understand the simple classification approaches taught in class and how they might be applied specifically in the use case of text data. We will implement the Naive Bayes classifier (NB), Support Vector Machine (SVM), Random Forest (RF) and K-nearest neighbors (KNN) algorithms to classify different scientific research papers into various disciplines based on their titles. These are widely touted as some of the most popular and effective algorithms used in text classification in practice. Nowadays, different researchers may even use more elaborated and sophisticated methods, most notably large-scale neural networks and similar structures, to classify documents, emails, file and/or data into their proper categories. Nonetheless, such advanced and computationally expensive methods are outside the scope of this study.

Text classification techniques require thorough preprocessing of the data given that the natural language constituting the source data — English in this case — can contain many words with the same meaning that share the same word core (or root). Another obstacle to overcome is transforming the raw text into feature vectors (i.e. numeric variables) via text vectorization. This paper covers two of the most commonly used vectorization methods; (1) Count and (2) Term Frequency-Inverse Document-Frequency (TF-IDF). Count vectorizer will create a vector of frequencies that show how often the word appears in the text. The second, TF-IDF, will use a more elaborate measure, the multiplication of the word frequencies with a weighting factor. More details about the preprocessing, text vectorization, and classification models will be explained in later sections.

The original idea for this project was to replicate the findings from *Survey on supervised machine learning techniques for automatic text classification* by Kadhim (2019), but retrieving the author's original data proved to be quite complicated. Therefore, we created our own database and limited the study to comparing the efficiency of the methods implemented by the aforementioned author in their experiment versus our own. For instance, according to the previously mentioned paper, NB should be able to outperform the SVM method if the dataset is significantly small; otherwise, methods like SVM and KNN

should have a higher accuracy. We investigate this relationship as well as others across different methods by cross-examining the author's findings with our own.

The approach implemented in this study has its limitations. Based on *Research paper classification systems based on TF-IDF and LDA schemes* by Kim et al., (2019) creating a classification model based on only the titles of the articles might not be enough in some circumstances. A more robust approach would be to include the abstracts, keywords, and sometimes even the entire body of the paper. Nonetheless, due to computational limitations we could not acquire all the papers in their totality, hence we only used the titles for feasibility.

2 Data

2.1 Data Collection

The data is collected using a publicly available Python web scraping script *PyPaperBot* by Vito Ferrulli (2021). We visit the websites of prominent academic journals in four disciplines — statistics, civil engineering, biology, and education — and scrape the titles of published research papers. We also obtain other metadata such as the authors, DOI, and more, but these characteristics are not utilized in the final analysis. In total, the dataset consists of 1917 papers: 482 from *The Annals of Statistics*, 499 from *Geotechnics* (civil engineering), 499 from *The American Naturalist* (biology), and 437 from *The Journal of Higher Education* (education).

For purposes of modeling later, the data is split into a training set and a test set. We choose a 70-30 train-test split and end up with 1341 papers belonging to the training set and 576 papers in the test set, as depicted in Figures 1 and 2. In both the training and test sets, there is a roughly uniform distribution of papers belonging to each category.

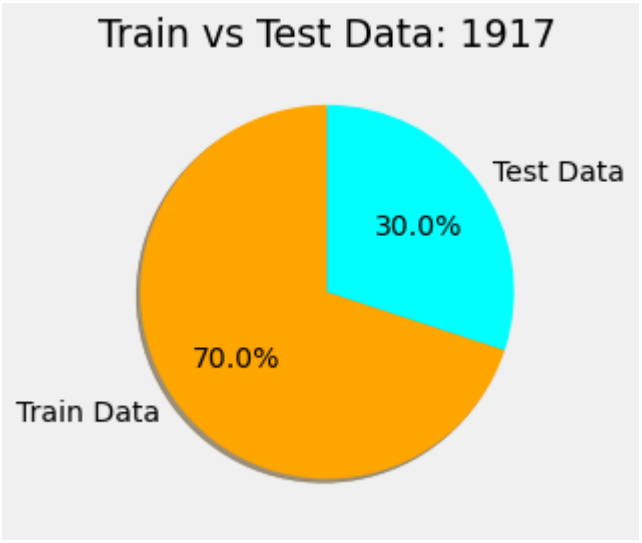

Figure 1: Amount of data split into train and test sets.

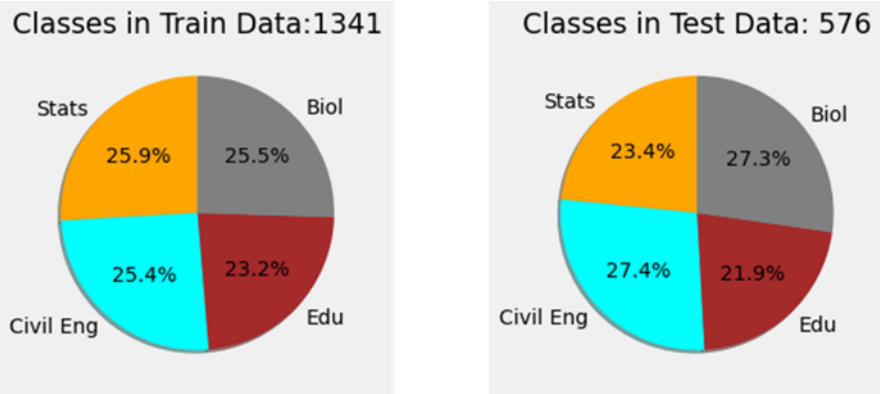

Figure 2: Distribution of data in each category.

2.2 Preprocessing & Exploratory Data Analysis

After importing our data, we proceed to apply several techniques of natural language processing in order to convert it into a format ready for modeling. This and subsequent methods are done in Python (rather than R) as this language allows for much smoother handling of character strings and contains some convenient packages for preprocessing text and designing the models. Our aim in this step is to extract numerical features from the raw text in our corpus in order to make the data intelligible to the computer. To accomplish this, we carry out four broad tasks: clean, vectorize, normalize, and reduce the dimensionality of the data.

First, we remove any titles that include words not in the English language in order to maintain uniformity in our data. Next we implement tokenization, splitting the strings of continuous text (i.e. the titles) into lists of individual words appearing in those strings; the delimiter used in this case is the empty whitespace character ' '. An example of this preprocessing step as well as future steps are illustrated in Table 1. Next we want to remove "noise" from the text, meaning any components of the text that are not significant to the overall meaning; these include things such as punctuation marks, upper-case letters, numbers, and stopwords. Stopwords in English are words like "of," "the," "a," etc. The last step is stemming, or the removal of any affixes so that different words can be grouped together according to their root. After all, there should be no intrinsic difference between, for example, "estimate" and "estimation." As a whole, in this step, we have transformed the raw text of each paper title into a list consisting of the stemmed individual words found in that title.

Table 1: Steps to obtain unique "core" words from original titles in the data.

Original Title	Tokenized	Noise Removed	Stemmed
On the adaptive elastic-net with a diverging number of parameters	On, the, adaptive, elastic-net, with, a, diverging, number, of, parameters	adaptive, elasticnet, diverging, number, parameters	adap, elasticnet, diverg, number, paramet

Once we have the list of (stemmed) words, rather than strings of continuous text, we need to decide how to capture this data in a numerical way. There are several options available which are commonly

used in natural language processing; we implement two of these, Count Vectorization (one type of "bag-of-words" methods) and Term Frequency-Inverse Document Frequency (TF-IDF) vectorization. Count vectorization tabulates a count of the number of times each word appears in the corpus, while TF-IDF involves a slightly more complicated measure. In TF-IDF, a "weighted average" of sorts is assigned to each word according to the following formulas:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \epsilon d} f_{t',d}}$$

$$idf(t,D) = log \frac{N}{|\{d \epsilon D : t \epsilon d\}|}$$

$$tf - idf(t,d,D) = tf(t,d) \times idf(t,D)$$

where $f_{t,d}$ is the number of times that term t occurs in document d and the denominator for tf is the total number of terms in document d. Also, N is the total number of documents in the corpus and the denominator for idf is the number of documents where the term t appears.

Essentially, *tf-idf* assigns less importance to words which appear very frequently in the entire set of documents as a way of balancing/weighting the relative impact a word may have in an individual document. The final values of *tf-idf* vectorized words will be from 0 to 1. An example of each vectorization method is demonstrated in Table 2 where two toy example sentences are converted accordingly.

Table 2: Example of two vectorized sentences data (count and TF-IDF)

		student	pass	difficult	class	work	hard	final	project
count	1. The students passed the difficult class	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00
	2. The students worked hard in the final project but did not passed the class	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
tf	1. The students passed the difficult class	0.25	0.25	0.25	0.25	0.00	0.00	0.00	0.00
	2. The students worked hard in the final project but did not passed the class	0.14	0.14	0.00	0.14	0.14	0.14	0.14	0.14
idf	1. The students passed the difficult class	1.00	1.00	2.00	1.00	2.00	2.00	2.00	2.00
	2. The students worked hard in the final project but did not passed the class	1.00	1.00	2.00	1.00	2.00	2.00	2.00	2.00
tf-idf	1. The students passed the difficult class	0.25	0.25	0.50	0.25	0.00	0.00	0.00	0.00
	2. The students worked hard in the final project but did not passed the class	0.14	0.14	0.00	0.14	0.29	0.29	0.29	0.29

3 Modeling

Before we proceed with model construction, we first want to see that the common words in each category make good intuitive sense. Hence we construct plots like Figure 3 and Figure 4 that show the most frequent (stemmed) words in statistics and biology. For example, the most common words in statistics articles are "estimation" or "estimate," "dimension," "linear," etc. Likewise, for biology, we have words like "evolution" or "evolve," "species," etc. In these cases, as well as in the other 2 categories not depicted here, we as humans can broadly see that these common words seem to display an obvious pattern, indicating that the task of identifying paper categories, given words like these, is one that is feasible. This analysis gave us confidence in the methodology used to preprocess and vectorize the text-data to move on with

the modeling phase.

At this point, we also need to normalize the vectorized data frames. This is done by enforcing a mean of 0 and a standard deviation of 1 for each column. We would also like to consider implementing dimensionality reduction, as we are essentially dealing with thousands of individual unique words, corresponding to thousands of columns in the vectorized data frames. Instead of the more typically used principal component analysis (PCA), we compute the singular value decomposition (SVD). Specifically, we use the sklearn implementation TruncatedSVD; "truncated" in this context simply refers to the fact that we are able to select the first n columns from the matrix factorization (analogous to the first n principal components). This linear dimensionality reduction technique does not center the data before computing the SVD and hence can work with sparse matrices more efficiently, making it more applicable to this project and to text analysis in general. Figure 5 shows the scree plots for both count (left) and TF-IDF (right) vectorization, which we use to inform our choice of the number of components to use in the reduced data. From a total of 2712 unique words, we reduce it to 500 columns as most of the variance is explained with roughly the first few hundred components in both cases. Although count and TF-IDF have different scree plots and maybe we could have used different numbers of components for each method, ultimately, we decide to use 500 for both of them to maintain consistency in the results and analysis.

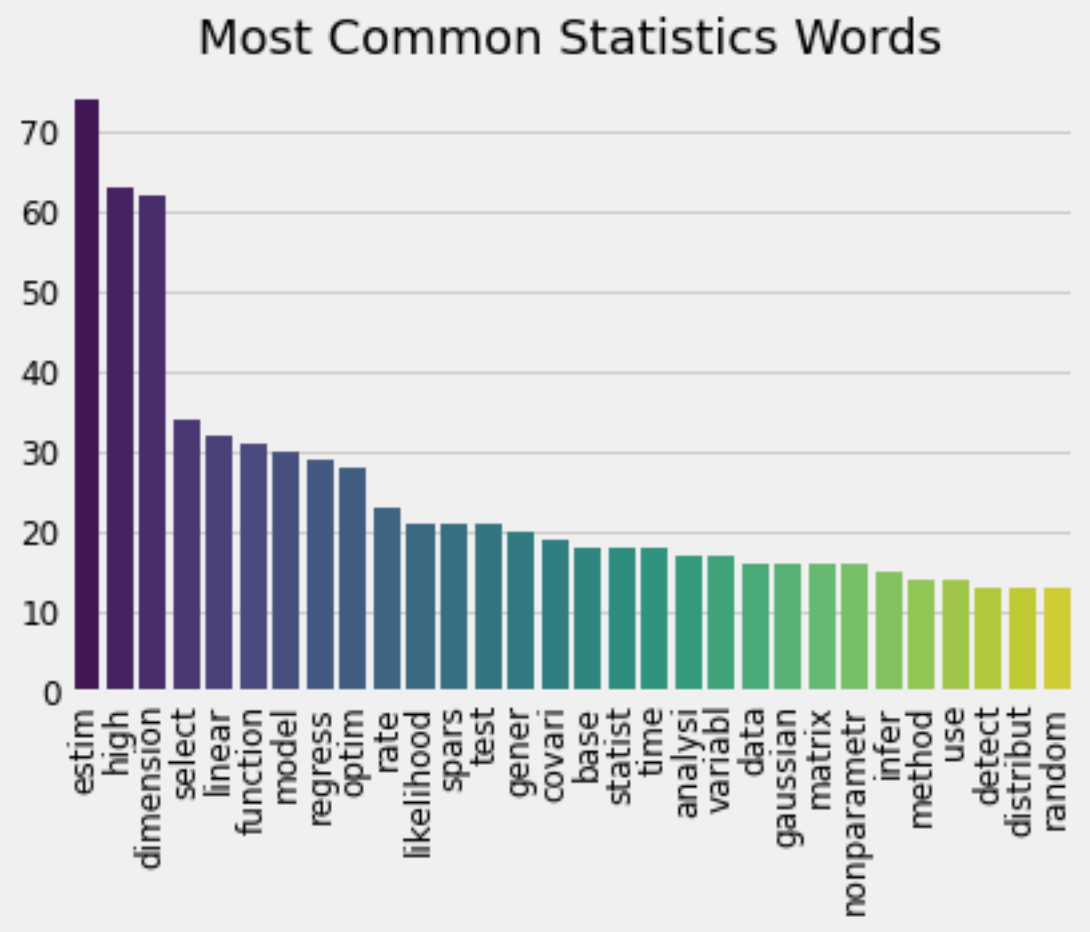

Figure 3: Most frequent (stemmed) words in the titles of statistics papers.

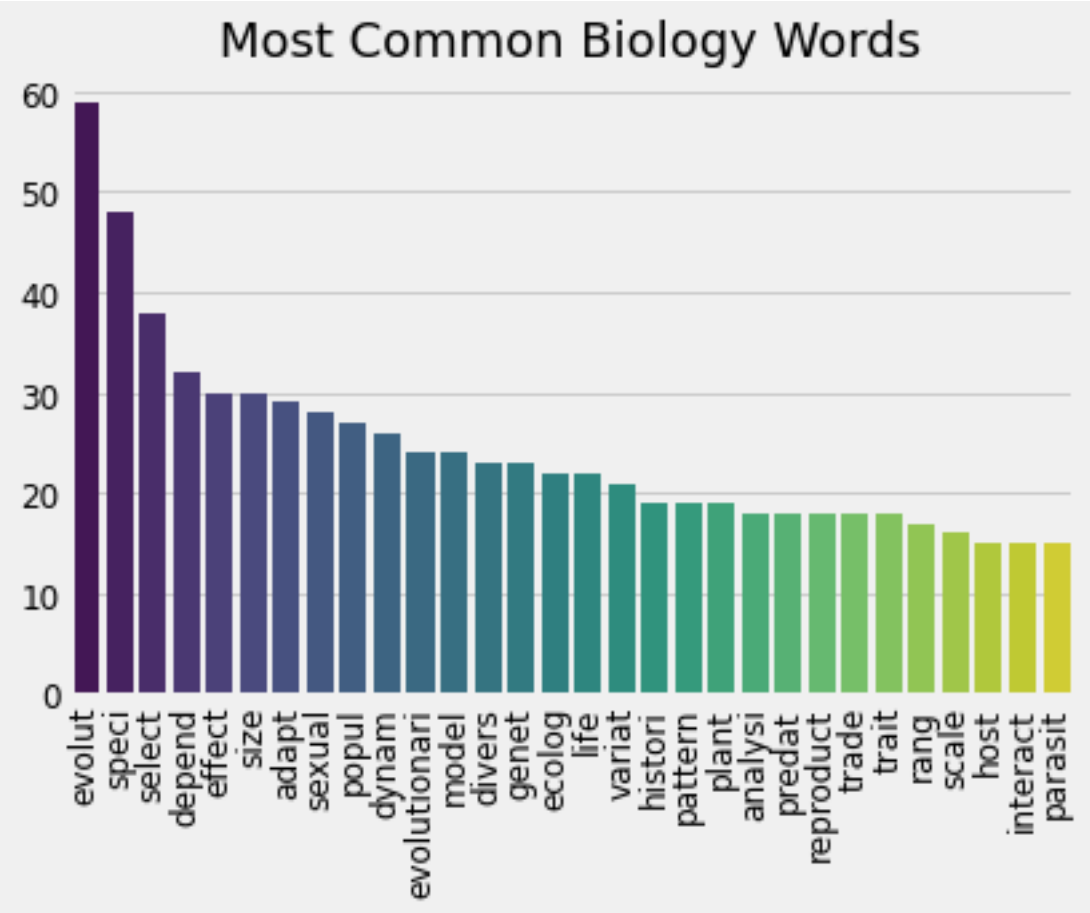

Figure 4: Most frequent (stemmed) words in the titles of biology papers.

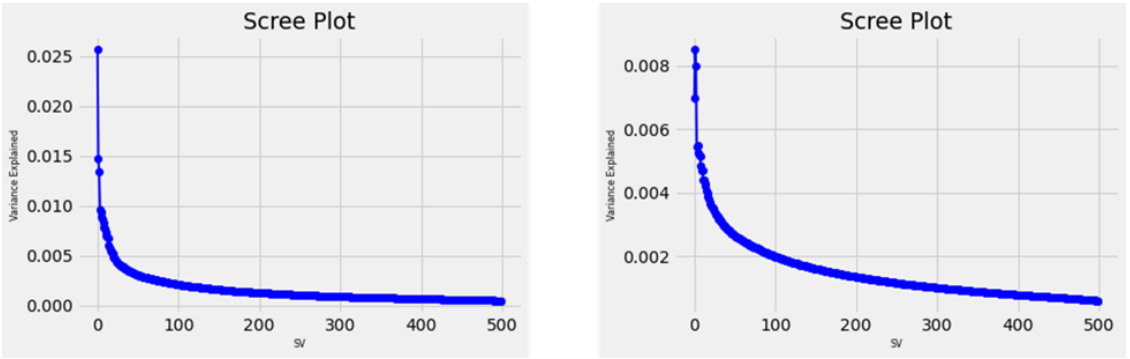

Figure 5: TruncatedSVD scree plots for count vectorization (left) and TF-IDF vectorization (right)

As discussed, we implement the Naive Bayes classifier (NB), Support Vector Machine (SVM), Random Forest (RF), and K-nearest neighbors (KNN) algorithms for classification. We decide to write the code for NB from scratch, given the nature of its relative simplicity; we do this to take the opportunity to explore this model in depth and we do not use pre-existing packages or functions such as those provided by sklearn in python. To compute the priors, we take an estimate of the probability using the proportion of papers in each category in the training set (this ends up being roughly a uniform prior). Our function

for the model (called nb_classifier) computes the following equation for the posterior probability:

$$P(class_i|title) = prior\ probabilities \times likelihoods\ of\ the\ words$$

$$P(class_i|title) = P(class_i) \times P(word_1|class_i) \times P(word_2|class_i) \times ... \times P(word_n|class_i)$$

where *class* could be statistics, civil engineering, etc., and *title* is a combination of words.

The final step is to calculate the accuracy of the model, which will be provided by counting the number of correctly classified titles divided by the total number of titles. In subsequent models, we use this same accuracy metric to keep consistency in comparing methods.

The steps for SVM, RF and KNN are slightly different as we incorporate a pre-existing sklearn package to run the data. The sklearn package contains built-in functions to fit the training data, predict on the test data, and verify the accuracy. As previously mentioned, as part of the standard procedure for using sklearn modules, we apply normalization; in addition, we reduce the dimensionality of the vectorized data frames to 500 components.

4 Results

We find and compare the accuracy of each text classification algorithm for both vectorization techniques. The final results are shown in Table 3. The NB had the best performance while RF had the lowest. However, RF accuracy is still significantly high. SVM, RF and KNN have a greater accuracy when we implement the TF-IDF vectorizer instead of Count vectorizer. The accuracy values are not significantly distant from each other, where SVM has an accuracy score of 0.88, while NB is slightly more efficient with an accuracy score of 0.90; this is expected based on the reported results from Kadhim (2019).

Using our NB code written from scratch, we are able to easily extract the misclassified titles along with the estimated probabilities of the titles belonging to each category. Actual probability values correspond to very small values, hence it is standard practice to show the natural logarithm of the probabilities instead. These values are shown in Table 4 for both Count and TF-IDF vectorization in Table 5. Values in green represent the true category while the values in red represent the misclassified category. These lists help us analyze the results and assess if the model is running correctly. In other words, a high accuracy score does not always mean the code is working properly, so we can evaluate these "failed classifications" to confirm it is running as intended. For example, Table 4 shows that the title "Model adequacy and the macroevolution of angiosperm functional traits" was misclassified into statistics because words like "model" and "function" appear in the title and those are more common to statistics, not biology. Another example is "Theory of disagreement-based active learning", which is a statistics paper and is classified as education because words like "theory" and "learning" appear in the title.

Table 3: Accuracy scores for each method based on different text vectorizations:

Method	Accuracy (using Count Vectorizer)	Accuracy (using TF-IDF Vectorizer)
Naïve Bayes Classifier	0.970	0.961
Support Vector Machine	0.943	0.962
Random Forest	0.887	0.909
K-Nearest Neighbors	0.910	0.935

Table 4: Examples of 4 titles incorrectly classified using NB and Count vectorizer.

Title	Stats	Ci Eng	Edu	Bio
Model adequacy and the macroevolution of angiosperm functional traits	-17.09	-20.60	-21.04	-17.32
Theory of disagreement-based active learning	-12.93	-15.78	-10.75	-16.39
Energy availability and density estimates in African ungulates	-29.23	-31.38	-32.20	-30.28
Plant age effects on soil infiltration rate during early plant establishment	-35.39	-30.45	-35.77	-27.36

The same titles that fail for Count vectorization fail for TF-IDF vectorization as well, in addition to a few others. Interestingly, Table 5 shows the title "Amino acids in nectar enhance butterfly fecundity: a long-awaited link" which are mostly natural science-biology words. This title has no words in the title that can explain the highest probability to be education. A possible explanation for this misclassification is the "zero-frequency" problem; this is discussed later.

Table 5: Examples of 4 titles incorrectly classified using NB and TF-IDF vectorizer.

Title	Stats	Ci Eng	Edu	Bio
Theory of disagreement-based active learning	-15.15	-17.99	-13.94	-17.31
Amino acids in nectar enhance butterfly fecundity: a long-awaited link	-30.22	-29.76	-28.61	-28.89
The Chief Diversity Officer: An examination of CDO models and strategies.	-27.68	-27.57	-26.07	-25.81
Plant age effects on soil infiltration rate during early plant establishment	-38.58	-35.50	-38.50	-34.19

A final concern regarding our results is the effect of title length on the accuracies. Longer titles may be more susceptible to being correctly classified as containing more words may lend to more opportunities to improve the estimation of the probability. Figures 6 and 7 illustrate however that the length of the titles do not seem to be overly significant. Most of the incorrectly classified titles are around 10 words

long for both types of vectorizers. Similarly, most of the *correctly* classified titles also have the same length, 10 words. However, it is worth noting that titles with more than 14 words are usually correctly classified and titles with 2 or 3 words are mostly incorrectly classified, so there could exist some possible bias here; normalizing the length of the titles may be an idea worth pursuing in the future for greater robustness.

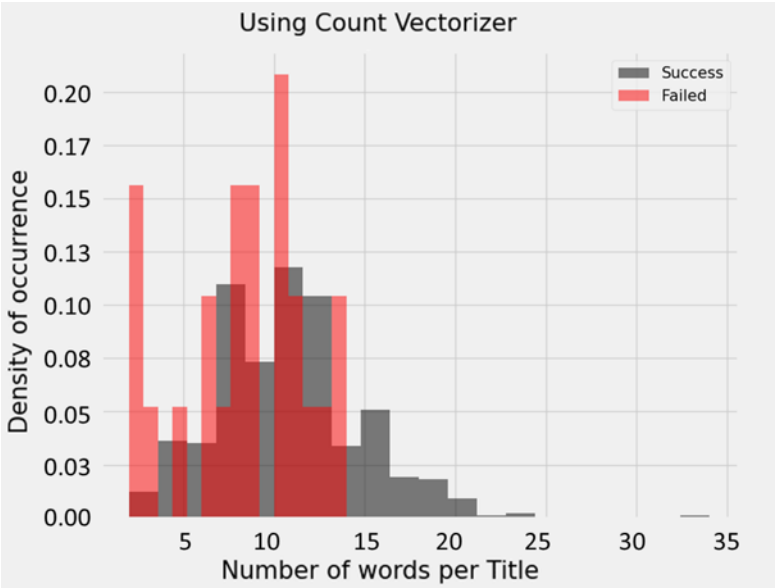

Figure 6: Normalized frequency of misclassifications vs. the word lengths of the titles, using Count vectorizer.

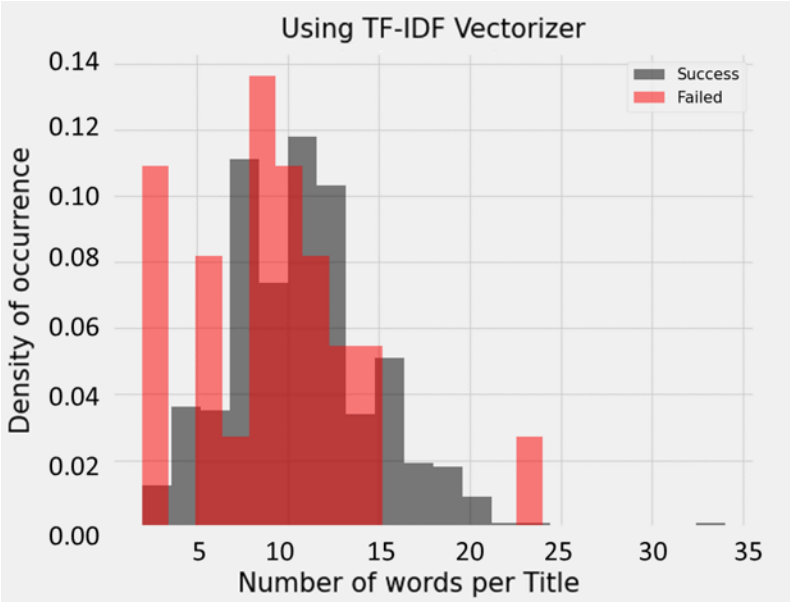

Figure 7: Normalized frequency of misclassifications vs. the word lengths of the titles, using TF-IDF vectorizer.

5 Discussion

Based on previous papers, we expected to get lower accuracy for NB as this high bias/low variance classifier method is not powerful enough to provide accurate models for large datasets. Nonetheless, it seems like the database is significantly small as NB performed better. Low bias/high variance classifier methods (e.g. SVM and KNN) overfit when used in small datasets and are more accurate for large training sets. This means that perhaps using only the titles does not create a significantly large dataset and more papers should be added.

We encounter some titles misclassified as education when none of the words seem to belong to education topics. This outcome could be explained by a combination of the "zero frequency" solution used in this project and the non-perfectly-uniform distribution of papers for each category. The "zero frequency" problem is what happens if a word does not exist in the corpus and the likelihood results in zeros. The method we use in this project to get around this issue is to add one when finding likelihoods of words to avoid zero probabilities. However, this is simply the most common and most straightforward approach to the zero-frequency problem, and there are many alternative schemes being proposed in the current literature. So for instance, going back to our earlier example from Table 5, the category of education has the smallest number of titles, hence the likelihood of words that do not appear in the corpus will always be higher for education compared to the other categories; this is why the classifier guesses that education is the right category.

Despite NB having the higher accuracy overall, we obtained very high accuracy across each of the different models. We now acknowledge some concerns surrounding this method in particular, despite its strong predictive performance. A powerful assumption of NB is the independence assumption which is often not realistic. Moreover, NB fails to produce good estimates for the correct class probabilities, so it may not be sufficient for some tasks or more advanced models building upon class probabilities.

We lastly touch upon possible explanations for the low accuracy of RF. One is the tuning of hyperparameters; for this project we did not specify any and simply used sklearn's default settings to keep things simple. RF in particular has a plethora of hyperparameters that could be quite impactful to the model's overall performance. Moreover, we could of course also use various hyperparameter optimization methods, such as cross-validation.

6 Conclusion and Future Work

In this project we classified research articles in different subjects based on their titles. We implemented the four most common text-classification techniques NB, SVM, RF and KNN. To transform the text-data into numeric features, we used two different types of vectorizers, Count and Term Frequency-Inverse Document Frequency (TF-IDF). From these we observe the highest accuracy when using NB and the lowest accuracy corresponds to RF. We also concluded that the length of the title will not help predict the category unless the title is very long (which most likely will be correctly classified) or if the length is too short (which most likely will be misclassified).

In order to improve our work, we can highlight a few additional tasks to consider. First, the dataset is relatively small, considering it is for text-classification. Hence, we can add more titles to the list. Moreover, we can improve the model if we add abstract, keywords, or even the body of the article to the corpus. Using more detailed text cleaning (e.g. different tokenization methods) can further help the efficiency of the model. Other linguistics/NLP-driven approaches can also be

incorporated for more informed, domain-backed modeling; for example, we could assess the power of lemmatization, taxonomy, etc. Similarly, many alternative vectorization approaches exist that can increase the accuracy. Additionally, we could evaluate the performance with alternative metrics. For example, precision and recall are two common accuracy metrics used in classification studies. Lastly, more sophisticated classification models altogether can be used, such as LDA/QDA, neural networks, ensemble methods to name a few.

7 References

Kadhim, A.I. Survey on supervised machine learning techniques for automatic text classification. Artif Intell Rev 52, 273–292 (2019). https://doi.org/10.1007/s10462-018-09677-1

Oliinyk, V., Vysotska, V., Burov, Y., Mykich, K., & Fernandes, V.B. (2020). Propaganda Detection in Text Data Based on NLP and Machine Learning. MoMLeT+DS.

Kim, SW., Gil, JM. Research paper classification systems based on TF-IDF and LDA schemes. Hum. Cent. Comput. Inf. Sci. 9, 30 (2019). https://doi.org/10.1186/s13673-019-0192-7

Pedregosa et al., Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830, 2011

Vito Ferrulli (2021) PyPaperBot https://github.com/ferru97/PyPaperBot