# Supervised Text Classification using Classical Natural Language Processing: A Comparative Study of Techniques

Connor Casey ccasey@bu.edu
Alex Melnick melnick@bu.edu
Loren Moreira ljmv@bu.edu
Bogdan Sadikovic bogdans@bu.edu

## ABSTRACT

*Text Classification is a classical machine learning problem with a long history. Natural Language Processing Techniques (NLP) were developed specifically to transform texts into usable sparse high-dimensional feature vectors. Since then, hundreds of techniques and methods to help improve performance on text classification have been developed. In this work, we conduct a comparative study on different foundational methods and techniques when used in conjunction with one another. We design a modular pipeline structure made up of components that are swapped out during evaluation to identify key strengths and weaknesses of specific approaches. We evaluate our pipeline on two separate real world datasets, 20newsgroups and Clickbait, in order to examine different behaviors or advantages when it comes to binary v.s. multiclass classification. In this work, we find that the optimal pairing is using TF-IDF and an SVM classifier without any feature selection, achieving 0.86 and 0.90 F1-score on the 20newsgroups and Clickbait datasets respectively..*

## 1. INTRODUCTION

Classical supervised machine learning is set on the foundation of possessing labels and feature vectors for single samples. This allows models to identify characteristics that allow them to predict the label of a sample it has never seen before. However, when it comes to text classification, it is not in a usable format and we must perform a specific kind of feature extraction. Being able to classify different texts is useful for applications such as email spam detection, sentiment analysis, and in this work, news topic prediction and clickbait detection.

Before utilizing NLP techniques, the first step is preprocessing the text data, in contexts where capitalization does not matter, or lots of variations of the same word are present, common steps are tokenization and lemmatization. Tokenization is the process of splitting a text up into a list of its words. Lemmatization is the process of transforming variations of the same word, like “run” and “ran” into the same word. This is useful because while humans reading can identify them as the same word, an algorithm reading them will treat them as completely separate words. Following this process, a common approach is removing “stop words”, stop words are essentially filler or connecting words. Words like “and, if, they, but” are vital to language, but when it comes to figuring out what a text could be about or how angry the person writing it was, they are not helpful. By removing stop words, the remaining words are key words. Key words are possibly important words that can help identify the label of the text being read.

Following this, is where major NLP techniques like Bag of Words (BoW) or TF-IDF come in, which are cornerstones of this work. In modern day natural language processing, they have stepped away from this approach with techniques such as word embeddings combined with transformers. For a foundational understanding and for lightweight applications however, these NLP techniques are still useful. Especially in the context of the pipeline; which will experiment with BoW/TF-IDF feature extracted data, feature selection methods, and different classification techniques to obtain a deep understanding of what works well together. It will be tested with the *20newsgroups* dataset, consisting of 18,000 articles with 20 different classes, and the *Clickbait* dataset, for binary classification with over 30,000 headlines.

## 2. RELATED WORKS

Previous foundational NLP works essentially all use TF-IDF and BoW, as they have been studied and used for more than 20 years (Fabrizio, 2002). TF-IDF is a well-suited and foundational method particularly with the *20newsgroups* dataset. As Dadgar et al. show in their 2016 paper, they use TF-IDF in conjunction with an SVM to obtain high F1-scores on subsets of the 20 newsgroups dataset. Particularly because we already planned on implementing an SVM, we believe this particular pairing in the study may perform very well. However, TF-IDF has been used for a multitude of other techniques, such as clustering with K-means (Ibrahim et

al. 2021) where they also cover the 20 newsgroups dataset, or KNN and Naive Bayes (Hakim et al. 2014). BoW is one of the oldest methods available for extracting features from a text, and has been somewhat passed up by newer methods. However in foundational applications it is helpful to include as a baseline. For example, in 2017, BoW was used in conjunction with a neural network for Vietnamese News classification (Van et al.). It still is a useful technique, as Alyamani et al. show that BoW is able to beat out TFIDF on smaller datasets.

Due to extremely high dimensionality often produced by BoW and TF-IDF, the need for a feature selection algorithm became evident. Chi-squared ($\chi^2$) is a well-established feature selection technique used in natural language processing (NLP) to reduce dimensionality by selecting the most relevant features for classification tasks. For example, a study on Arabic sentiment analysis compared Chi-squared with Information Gain, demonstrating that Chi-squared effectively reduced the feature space and improved classification accuracy when used with models like SVM and KNN (Yousef and Alali, 2022). Another study on fake news detection found that combining Chi-squared with bigram feature extraction significantly enhanced the performance of machine learning models such as Random Forest and Extreme Gradient Boost (Mohan, Assi, & Mohammed, 2023). Additionally, research on text classification using SVM showed that applying Chi-squared with N-grams improved accuracy by reducing irrelevant features, achieving an accuracy increase of up to 1.58% (Fachrurrozi et. al. 2023).

Mutual Information (MI) is a pivotal feature selection technique in Natural Language Processing (NLP), quantifying the dependency between variables to identify features that significantly influence classification outcomes. For instance, Novovičová et al. (2004) introduced algorithms utilizing improved MI for text classification, demonstrating enhanced effectiveness in selecting informative features. Similarly, Zhou et al. (2021) proposed a feature selection method combining MI with correlation coefficients, effectively reducing redundancy and improving classification accuracy. Additionally, Beraha et al. (2019) provided theoretical insights into MI-based feature selection, offering guarantees on the solutions proposed by these algorithms. These studies underscore MI's versatility and efficacy in enhancing NLP tasks through strategic feature selection.

Regarding the classifiers, it has been previously shown that SVM algorithms work well in the context of text classification because of its sparse, high dimensional nature. Liu et Al. state in their 2010 paper that SVM is based on the “structural risk minimization principle and the limited sample of information”, which is inferred to work well with the sparse datasets generated from TF-IDF and BoW. The Random Forest classifier will also be tested, as it generally performs well with sparse high dimensional datasets, as it has been shown to remove noise in text datasets (Shah et al., 2020). For the purpose of the experiment, a multinomial Naïve Bayes algorithm will also be trained, as a baseline for the performance of all trained models, similar to previous work cited (Hakim et al. 2014).

# 3. METHODOLOGY

## 3.1 NLP TECHNIQUES

Bag of Words is a classical natural language processing technique that has the goal of taking unstructured human text data, and processing it into structured numerical data via tokenization, removal of stop-words, and lemmatization. It places all words in an entire dataset into a vocabulary which form the feature space, and then checks how many times a word appears in each document, and assigns that value to that document's corresponding feature. This allows for data to be analyzed quicker, and for the most important information (the number of occurrences of words per class) to be highlighted.

The BoW technique is best when it is integrated into other techniques, as it allows for a more readable format, and potentially better analysis of datasets. On its own, it creates a large sparse matrix of data that displays all the corresponding appearances of words.

TF-IDF is a frequently-used technique for text feature extraction. It computes a statistical measure to evaluate the importance of each word (term) in a document relative to a corpus of documents It is the product of two measures, Term Frequency (TF), and Inverse Document Frequency (IDF). TF is defined as for a corpus with N documents, and a vocabulary with M terms, the TF for the *i*th term and *j*th document is:

$$TF(t_i, d_j) = \frac{term\ frequency\ in\ d_j}{total\ term\ count\ in\ d_j}$$

This allows us to see how important a word is for a particular document, which can highly assist with identifying the label. We can measure the Document Frequency (DF), which is defined as:

$$DF(t_i) = \frac{\#\ of\ documents\ t_i\ is\ in}{N}$$

Thus, by inverting it we obtain:

$$IDF(t_i) = \frac{N}{\#\ of\ documents\ t_i\ is\ in}$$

However, it is possible that a term in the vocabulary is so rare across the corpus, that the IDF value could explode. So for smoothing and preventing extreme values, it is commonly defined as:

$$IDF(t_i) = log(\frac{N}{\#\ of\ documents\ t_i\ is\ in + 1})$$

TF-IDF is simply the product of these two, giving us an effective statistical measurement of how important a word is.

$$TFIDF(t_i, d_j) = TF(t_i, d_j) * IDF(t_i)$$

This is very effective at isolating important words, as unlike BoW we can set a multitude of thresholds to effectively perform feature selection as we extract them. By only keeping terms that are within a particular DF range, we can limit popular keywords and remove anomalous keywords. By simply thresholding the average TF-IDF score for a term, you can select the top-K features. We used the scikit- learn TfidfVectorizer() for all of our experiments.

### 3.2 FEATURE SELECTION

For feature selection, two techniques were utilized, Chi-Squared and Mutual Information. Chi-squared ($\chi^2$) is a well-established feature selection technique used in natural language processing (NLP) to reduce dimensionality by selecting the most relevant features for classification tasks. It is a statistical method for analyzing if there is a significant association between categorical variables. It is calculated using the below formula in conjunction with a contingency table:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

$\chi^2$ is the Chi-Squared test statistic. $O_i$ is the observed frequency for a feature-class combination. $E_i$ is the expected frequency for the same feature-class combination.

Chi-Squared assumes independence of variables and tests against this assumption. The expected frequency of a feature-class combination is simply the amount of expected observations if the relationship were defined by a uniform random variable (i.e., if the observations are evenly distributed). If there is a significant difference between the observed and expected frequencies of a specific feature-class combination, the Chi-Squared statistic will be greater than a specified threshold and the assumption is rejected. This indicates that the two variables are not independent of each other. Chi-Squared is considered to be very efficient, with a time complexity of O(n).

Mutual Information (MI) is a powerful technique often used in feature selection to measure the dependency between two variables. Unlike Chi-Squared, which assumes independence as a null hypothesis, Mutual Information quantifies how much knowing one variable reduces the uncertainty of another. It is based on information theory and captures both linear and non-linear dependencies. The MI score is calculated as follows:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

I(X, Y) represents the mutual information between variables X (feature) and Y (class). P(x, y) is the joint probability of X and Y, and P(x) and P(y) are their marginal probabilities.

Mutual Information is especially useful in situations where relationships between variables are complex and not easily captured by simpler statistical measures. A higher MI score for a feature indicates a stronger dependency on the target class, making it more relevant for predictive modeling. This versatility makes Mutual Information a robust and widely used feature selection method. Unlike Chi-Squared, however, Mutual Information is more computationally expensive, with a time complexity of O(n*logn).

Both Chi-Squared and Mutual Information have a hyperparameter, *k*, which is the number of features selected. This value was picked arbitrarily as there is a tradeoff between accuracy and compute speed. Higher values of *k* result in higher accuracy, but also increase the runtime of the classification model. The values chosen appear to yield good results in both categories.

### 3.3 CLASSIFICATION

The multiclass SVM learning problem is described by the following equation and set of constraints:

$$\boldsymbol{\theta_{svm} = argmin \frac{1}{2} \sum_{k=1}^{m} \|w_k\|^2 + penalty(\xi_1, \dots, \xi_n)}$$
$$\boldsymbol{(\theta, \xi_1, \dots, \xi_n):}$$
$$\boldsymbol{\forall j, \forall l \neq y_j, \left(\theta_{y_j} - \theta_l\right)^T x_j^{ext} \geq 1 - \xi_j}$$
$$\boldsymbol{\xi_j \geq 0}$$

The objective function looks to find the best **Θ** that separate the classes in the feature space. The argument of the sum – $\| \mathbf{w_k} \|^2$ – looks to regularize the parameters of

$\mathbf{w_k}$ for each class to deal with potential overfitting, while the penalty argument in the sum allows for soft margins in the SVM classifier, which handles cases when the data cannot be linearly separated. For our purposes, the SVM model was used with an RBF kernel that would allow for optimal decision making in higher dimensional space. This entails the tuning of a parameter γ that dictates the "smoothness" of the decision boundary of the trained model. A high γ value may lead to overfitting as a result of a boundary that is too smooth. As for the run-time complexity of this algorithm, it is hard to characterize, but a previous study showed it to be in O(nSV × d), where nSV indicates the number of support vectors, and d is the dimension (Claesen et. al., 2010).

Random Forest classification is described by the following learning problem:

$$h_{\mathcal{T}}(x) \;=\; argmax\, \hat{p}\,(\, y \mid leaf_{\mathcal{T}}(x)\,)$$

$$y \in Y$$

This function essentially is traversing the nodes of the tree based on the feature values of x, and at the end of the traversal, x is in a specific leaf node. The tree then computes the probabilities of each class using the training data samples in that leaf, and the predicted class is the highest of said probabilities. It is important to note that there are a myriad of hyperparameters that can be set when utilizing the black box function in scikit-learn, including but not limited to: max_leaf_nodes, max_depth, min_samples_leaf, max_features, etc. These were all left unset, and the parameter used and kept constant all throughout testing was n_estimators and random_state. These were set to 100 and 42 respectively, and they were tested very briefly at the start of the experiment on small data, and it was found that a change in either of these resulted in very minimal change, so they were kept constant for the sake of reproducibility. The exact time complexity for the building of the random forest algorithm is hard to determine, however it is bounded by the following expression:

$$\mathbf{\Theta}(\, knlog(n)\,) \leq \mathbf{\Theta} \leq \mathbf{\Theta}(kn^2)$$

Where k denotes the number of elements, and n denotes the depth of said tree (Štepánek et. al., 2021).

As for the multinomial Naive Bayes algorithm, it relates the conditional probabilities of events utilizing Bayes' theorem. It acts under the assumption that features are conditionally independent of each other, such that the algorithm is modeled after the following:

$$\hat{y} \;=\; argmax\, P(C_k)\, \mathbf{\Pi}_{i=1:n} P(X_i \mid C_k)$$

$$k \in \{1, K\}$$

Where C is the class, and k is the number of classes. In the context of text classification, Naive Bayes models the probability of a feature belonging to a class based on the frequency of the words within the document. A black box function from scikit-learn was also used for this purpose, which uses Laplace Smoothing by default to handle cases where a term doesn't occur in the data. Naive Bayes is denoted to have an O(nd) training time, with a testing time complexity of O(kd).

# 4. EXPERIMENTAL METHODOLOGY

Our experiment's goal was identifying the best combination of methods/techniques in our pipeline. For our experiments, due to the size and dimensionality of the dataset performing grid search CV for hyperparameter tuning each stage of the pipeline was seen as untenable. Thus for the NLP parameters, and baseline classifier's hyperparameters we left them at default or recommended values. We used scikit-learn implementations for TFIDF, BoW, Chi-Square, Mutual Information, SVM, RandomForest, and Naive Bayesian. The pandas dataframes were utilized for loading and saving datasets.

For identifying the best performing pipeline configuration on each dataset, two main experiments were ran using TF-IDF and BoW feature extracted data respectively. *20newsgroups* contains both articles and their metadata. For the experiments, there was assumed access to the metadata of the dataset, which is responsible for the better-than-usual performance. An 80/20 training split was utilized alongside a 5-fold cross validation using stratified sampling solely for TF-IDF case to maintain any class imbalances: this was done with sci-kit-learn's stratifiedKFold. The macro average F1-score was utilized to accommodate for the multi-class performance for *20newsgroups*. For feature selection, the top-k features chosen were 1000 and 500 respectively for *20newsgroups* and *Clickbait*. Following this, the best performing pipeline was selected from these large baseline competitions and further tuned its hyperparameters and the amount of features selected for it.

Feature selection was implemented using the open-source Python sklearn.feature_selection.chi2 library for Chi-Square and sklearn.feature_selection.mutual_info_classif for Mutual Information. Threshold Chi-Square or Mutual Information scores were not utilized for selecting features. Instead, the feature vectors were sorted according to the Chi-Square and Mutual information scores. The top $k$ feature vectors were then extracted with the highest Chi-Square and Mutual Information

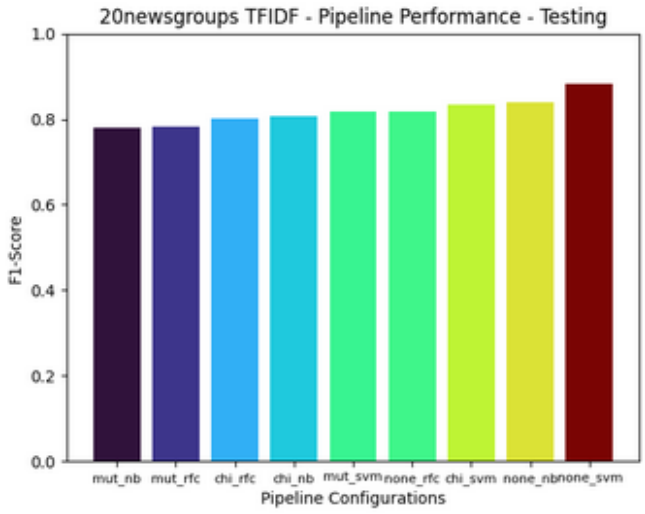


**Figure 1.** F1-scores of each pipeline on the testing 20 Newsgroups dataset. The prefix indicates the feature selection method used (Chi-squared, Mutual Information, or None), and the suffix corresponds to the model applied (Random Forest, Naive Bayes, or SVM).

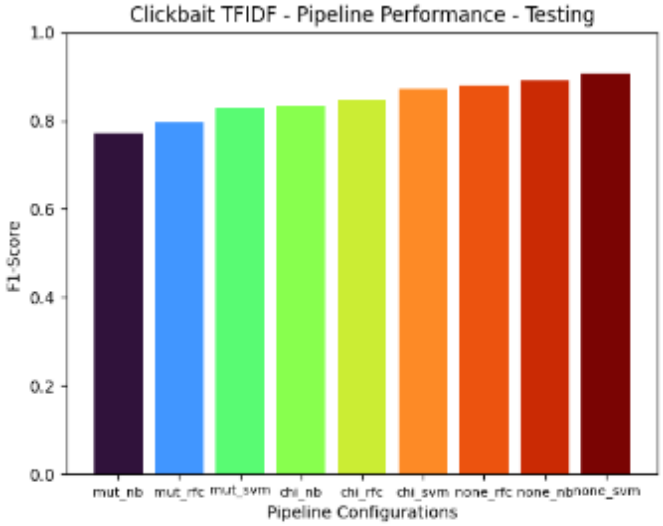


**Figure 3.** F1-scores of each pipeline on the testing Clickbait dataset. The prefix indicates the feature selection method used (Chi-squared, Mutual Information, or None), and the suffix corresponds to the model applied (Random Forest, Naive Bayes, or SVM).

scores, where *k* is a tunable hyperparameter. For the *20newsgroups* dataset, *k* was chosen to be *1000*. For the *Clickbait* dataset, *k* was chosen to be 500.

## 5. RESULTS

For TF-IDF feature extracted data, a maximum F1-score of 0.86 and 0.90 was observed respectively for *20newsgroups* and *Clickbait*. In Figures 1 and 3, we can observe that for TFIDF, the SVM with no feature selection performed the best. Consistently, pipelines that used no feature_selection had the highest performance, with the exception of the chi-square SVM pipeline in *20newsgroups*. SVM is consistently the highest performing model, though all are comparable. This is because SVMs excel at learning from high-dimensional

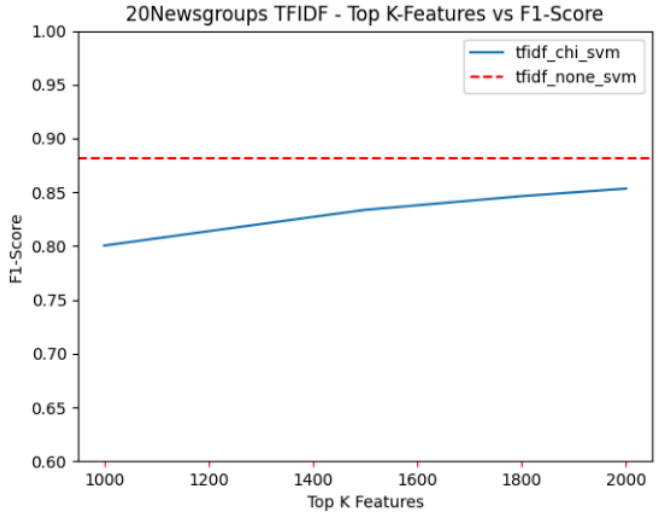


**Figure 2.** F1-scores increase with diminish returns as the amount of selected features approaches the original feature count.

sparse feature spaces, which is exactly the type of data that TFIDF produces. RandomForest and Naive Bayesian fall behind because of RandomForest's tendency to overfit the training data, hurting its generalizability and Naive Bayesian's assumption that key words are conditionally independent. When analyzing the feature selection methods, it is observed that any feature selection hurts performance, with chi-square beating out mutual information. When observing how the F1-score changes with an increase in selected metrics in Figure 2, it is determined that TFIDF already selects the most relevant features, and that any reduction in features leads to a decrease in model performance. Though it is important to note that this reduction in features does reduce training time. For BoW, there was a discrepancy with the data after it was processed. It would possess missing features, reading as NaNs. In order to counteract this, we filled in those missing features with zeros. This could have been possibly related to the dubious results on the BoW. Due to the preprocessing done, the data could have been altered in way that created words that never existed within the dataset due to the way, thus potentially increasing the number of zeros in a sparse matrix

The results for the Bag of Words analysis were very unexpected. The F1 Scores ranged from below 0.10 to approaching 1.0 . It is reasonable to assume that there is a methodological issue at work here. The result for the SVM was good, and performed as expected at around an 0.84 F1 score, which indicates the model was able to find relationships between the words within an article, and the article's topic. The result for the Naive Bayes was very low, and was likely a result of the distribution of the data. Since lots of preprocessing was used, there could have been too many words removed, which resulted in the model not being as accurate due to the data being too distributed, and thus not well represented

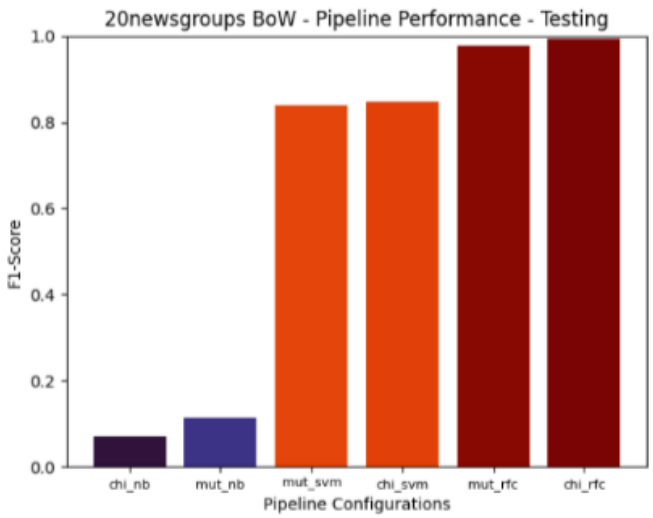


**Figure 11.** F1-scores of each pipeline on the testing 20 Newsgroups dataset. The prefix indicates the feature selection method used (Chi-squared, Mutual Information, or None), and the suffix corresponds to the model applied (Random Forest, Naive Bayes, or SVM). However, results are suspect as no previous work has been shown to obtain a perfect score and the naive bayesian has significantly low performance.

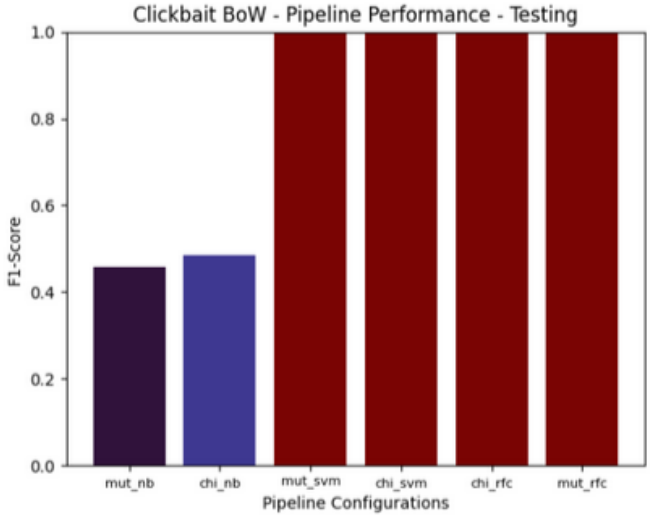


**Figure 12.** F1-scores of each pipeline on the testing 20 Newsgroups dataset. The prefix indicates the feature selection method used (Chi-squared, Mutual Information, or None), and the suffix corresponds to the model applied (Random Forest, Naive Bayes, or SVM). However, results are suspect as no previous work based on the other dataset and the naive bayesian has significantly low performance.

to the Naive Bayes model. The Random Forest model's performance was way too high, approaching a 1.0 F1-score. This could be plausible for the *Clickbait* dataset, but not for the *20newsgroups* dataset. Likely, our inclusion of certain metadata contributed to RandomForest's behavior. Similar to TF-IDF, Chi-squared had consistently higher performance compared to Mutual Information. This is because Chi-square relies on using data that is already within the label, whereas Mutual Information relies on getting information about one variable using others, which could have been hindered as a result of BoWs matrix, and the dataset used.

## 6. CONCLUSION

This work conducted a thorough comparative study of the strengths and weaknesses of methods used at every stage of the designed pipeline. It was discovered that using feature selection for TF-IDF extracted data can be detrimental to performance unless your aim is to reduce time complexity. When using BoW however, using feature selection can be beneficial as BoW does not have a way to get rid of non-important keywords like TF-IDF has. For both NLP methods, the data extracted is very high-dimensional and extremely sparse which makes it well suited for essentially all of the tested models, each having comparable performance. SVM paired with an RBF-kernel was the best out of the three models because the support vectors formed from the sparse dataset representation are able to better encapsulate the behavior of the important words thanks to the kernel.

When it comes to feature selection, Chi-Squared feature selection outperformed Mutual Information. This is likely because Chi-Squared is well-suited for categorical data and measures the strength of association between features and the target based on observed and expected frequencies. The 20 Newsgroups dataset, represented as sparse bag-of-words vectors, aligns well with Chi-Squared's frequency-based approach. In contrast, Mutual Information, while capturing both linear and non-linear dependencies, may be more sensitive to noise and sparsity, reducing its effectiveness in this context.

When designing a pipeline for supervised text classification with classical techniques, your best choice is utilizing TF-IDF along with SVM. Contrary to the original assumption, it is best to not undergo any feature selection. Using these configurations, an F1-Score of 0.86 and 0.90 on the 20newsgroups and *Clickbait* data was obtained.

For future work, it would be best to focus on tuning each individual pipeline configuration before selecting the best configuration. Since this was not a viable alternative for the purpose of this experiment, the parameters were selected heuristically and then tuned, thus leaving the possibility that another configuration could surpass the current best performance.

## 9. REFERENCES

S. M. H. Dadgar, M. S. Araghi and M. M. Farahani, "A novel text mining approach based on TF-IDF and Support Vector Machine for news classification," 2016 IEEE International Conference on Engineering and

Technology (ICETECH), Coimbatore, India, 2016, pp. 112-116, doi: 10.1109/ICETECH.2016.7569223.

R. K. Ibrahim, S. R. M. Zeebaree, K. Jacksi, M. A. M. Sadeeq, H. M. Shukur and A. Alkhayyat, "Clustering Document based Semantic Similarity System using TFIDF and K-Mean," 2021 International Conference on Advanced Computer Applications (ACA), Maysan, Iraq, 2021, pp. 28-33, doi: 10.1109/ACA52198.2021.9626822.

A. A. Hakim, A. Erwin, K. I. Eng, M. Galinium and W. Muliady, "Automated document classification for news article in Bahasa Indonesia based on term frequency inverse document frequency (TF-IDF) approach," 2014 6th International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia, 2014, pp. 1-4, doi: 10.1109/ICITEED.2014.7007894.

Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47.

Xu, Baoxun, et al. "An improved random forest classifier for text categorization." J. Comput. 7.12 (2012): 2913-2920.

Yousef, M.W., \& Alali, A.M. (2022). Analysis and Evaluation of Two Feature Selection Algorithms in Improving the Performance of the Sentiment Analysis Model of Arabic Tweets. International Journal of Advanced Computer Science and Applications.

Mohan, S., Assi, J., \& Mohammed, A.H. (2023). Using Supervised Machine Learning Models and Natural Language Processing for Identification of Fake News. International Journal of Data Science and Advanced Analytics.

Fachrurrozi, S., Muljono, Shidik, G.F., Fanani, A.Z., Purwanto, \& Zami, F.A. (2021). Increasing Accuracy of Support Vector Machine (SVM) By Applying N-Gram and Chi-Square Feature Selection for Text Classification. 2021 International Seminar on Application for Technology of Information and Communication (iSemantic), 42-47.

Toan Pham Van, Ta Minh Thanh(2017). Vietnamese News Classification based on BoW with Keywords Extraction and Neural Network

Hasan J. Alyamani(2022). Determining Feature-Size for Text to Numeric Conversion based on BOW and TF-IDF

L. Štěpánek, F. Habarta, I. Malá and L. Marek, "A random forest-based approach for survival curves comparison: principles, computational aspects, and asymptotic time complexity analysis," 2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS), Sofia, Bulgaria, 2021, pp. 301-311, doi: 10.15439/2021F89. keywords: {Computer science;Computational modeling;Robustness;Hazards;Complexity theory;Time complexity}

Python Course. (n.d.). *Random forests in Python*. Retrieved December 11, 2024, from https://python-course.eu/machine-learning/random-forests-in-python.php

F. Knyszewski. (2020, April 18). Implementing Naive Bayes for Sentiment Analysis in Python. AI Time Journal. https://www.aitimejournal.com/implementing-naive-bayes-for-sentiment-analysis-in-python/2820/

Novovičová, J., Malík, A., Pudil, P. (2004). Feature Selection Using Improved Mutual Information for Text Classification. In: Fred, A., Caelli, T.M., Duin, R.P.W., Campilho, A.C., de Ridder, D. (eds) Structural, Syntactic, and Statistical Pattern Recognition. SSPR /SPR 2004. Lecture Notes in Computer Science, vol 3138. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-27868-9_111

Zhou, H., Wang, X. & Zhu, R. Feature selection based on mutual information with correlation coefficient. *Appl Intell* **52**, 5457–5474 (2022). https://doi.org/10.1007/s10489-021-02524-x

M. Beraha, A. M. Metelli, M. Papini, A. Tirinzoni and M. Restelli, "Feature Selection via Mutual Information: New Theoretical Insights," *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary, 2019, pp. 1-9, doi: 10.1109/IJCNN.2019.8852410.

# 10. DESCRIPTION OF INDIVIDUAL EFFORT

## Connor Casey:

**Code Written**: pipeline.py, tfidf.py, plotter.py

**Presentation:** Designed slide deck, Created all Slides except BoW Slide, Feature Selection Slides, and Classification Model Slides

**Experiments Run:** TF-IDF Pipeline with Clickbait

**Final Report Sections Written:** Abstract, Introduction, Related Work,3.1 TFIDF, Experimental Methodology, Results, Conclusion, Appendix Pseudocode, Appendix Data Table

## Alex Melnick:

**Code Written**: FeatureSelection.py, pipeline.py

**Presentation:** Feature Selection Slides

**Experiments Run:** TF-IDF Pipeline with 20newsgroups, BoW Pipeline with 20newsgroup and Clickbait.

**Final Report Sections Written:** 3.2 Feature Selection, Experimental Methodology, Results, Conclusion

## Loren Moreira:

**Code Written**: SVM_decision.py, RFC_decision.py, NB_decision.py, pipeline.py

**Presentation:** Made Classification Model Slides

**Experiments Run:** TF-IDF Best Model Tuning

**Final Report Sections Written:** Related Works, 3.3 Classification, Appendix Pseudocode, Conclusion, extensive final report proofreading

## Bogdan Sadikovic:

**Code Written**: NLP_BoW.py, NLP_BoW_Clickbait.py

**Presentation:** BoW Slides

**Experiments Run:** BoW Pipeline Experiments with 20newsgroups and Clickbait Dataset

**Final Report Sections Written:** 3.1 BoW, Results BoW, Potential Problems, Proofreading

# 11. APPENDIX

## Github Repository

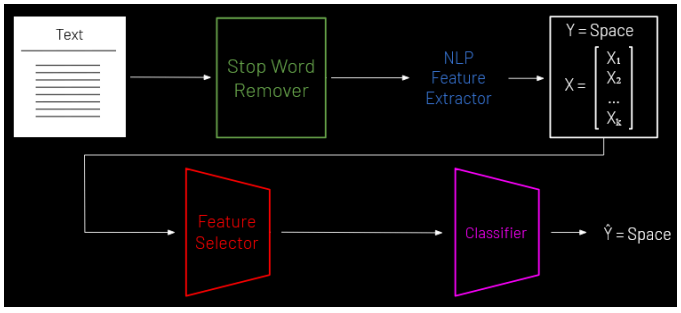


Figure 0: Pipeline Architecture

| Dataset | Samples | Distribution | Classes |
|---|---|---|---|
| 20newsgroups | 18836 | Uniform | 20 |
| Clickbait | 30296 | 53% Clickbait<br>47% Not | 2 |

Figure 4: Dataset Description Table

**Algorithm 1:** TF-IDF

**Result:** Sparse Dataset of TFIDF Scores
Given Corpus of N documents;
Given Vocabulary of M terms;
**for** *i = 1:M* **do**
  IDF(i) = log(N/(1 + of Documents Containing ith term));
**end**
**for** *j = 1:N* **do**
  **for** *i = 1:M* **do**
    TF(i,j) = (ith term count in jth document)/(total term count in jth document);
    TFIDF(i,j) = TF(i,j)*(i);
  **end**
**end**

Figure 5: TF-IDF Pseudocode

**Algorithm 2:** BoW Algorithm

**Result:** Sparse Dataset of BoW Scores
**Input:** Corpus of $N$ documents
**Output:** Document-term matrix $A$ of size $N \times$ Vocabulary Length
**Step 1: Build Vocabulary**;
**for** $j = 1$ *to* $N$ **do**
  **foreach** *word* $w$ *in the* $j$*-th document* **do**
    **if** $w \notin V$ **then**
      Add $w$ to $V$;
    **end**
  **end**
**end**
**Step 2: Initialize Document-Term Matrix**;
Initialize matrix $A$ of size $N \times |V|$ with all zeros;
**Step 3: Populate the Matrix**;
**for** $j = 1$ *to* $N$ **do**
  **foreach** *word* $w$ *in the* $j$*-th document* **do**
    **if** $w \in V$ **then**
      Increment $A[j,$ index of $w$ in $V]$;
    **end**
  **end**
**end**
**return** $A$;

Figure 6: BoW Pseudocode

**Algorithm 3:** Chi-square Feature Selection Algorithm

**Result:** Top-K Features based on Chi-square Scores
**Input:** Dataset $D$ with $N$ samples, $M$ features, and Target Label $Y$
**Output:** Selected features based on Chi-Square scores
**Step 1: Initialize List**;
Initialize an empty list '$chi_scores$'
**Step 2: Compute Chi Scores**;
**for** *each feature* $F$ *in* $D$ **do**
  Build contingency table for $F$ and $Y$;
  **for** *each cell* $(i, j)$ *in the contingency table* **do**
    $E_{ij} = \frac{\text{Row\_Total} \times \text{Column\_Total}}{\text{Dataset\_Total}}$;
    **if** $E_{ij} == 0$ **then**
      Skip this cell;
    **end**
    $\chi_{ij} = \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$;
  **end**
  Compute $\chi_F = \sum \chi_{ij}$ across all cells;
  Append $\chi_F$ to '$chi_scores$' **end**
**Step 3: Rank Features**;
Sort features by their Chi-square scores in descending order;
**Step 4: Select Top-K Features**;
Select the top $k$ features from the ranked list;
**return** *selected_features*;

Figure 7: Chi-square feature selection Pseudocode

**Algorithm 4:** Mutual Information Feature Selection Algorithm

**Result:** Top-K Features based on Mutual Information
**Input:** Dataset $D$ with $N$ samples, $M$ features, and Target Label $Y$
**Output:** Selected features based on Mutual Information scores
**Step 1: Initialize List**;
*Initialize an empty list 'mi_scores'*
**Step 2: Compute Mutual Information Scores**;
**for** *each feature $F$ in $D$* **do**
Compute the joint probability distribution $P(F,Y)$ for feature $F$ and target $Y$;
Compute the marginal probability distributions $P(F)$ and $P(Y)$;
Compute the Mutual Information score for feature $F$ and target $Y$:

$$MI(F,Y) = \sum_{f \in F} \sum_{y \in Y} P(f,y) \log\left(\frac{P(f,y)}{P(f)P(y)}\right)$$

Append $MI(F,Y)$ to 'mi_scores' **end**
**Step 3: Rank Features**;
Sort features by their Mutual Information scores in descending order;
**Step 4: Select Top-K Features**;
Select the top $k$ features from the ranked list;
**return** *selected_features*;

Figure 8: Mutual Information feature selection Pseudocode

**Input:** Original data
**Output:** Random Forest Outcome
**Tree_Outcomes** ← [ ]
**for** $i = 1$ to $n$ **do**
    Create a bootstrap sample from the original data
    Train a tree-model on this bootstrap data using the common stopping criteria
    **for** each split **do**
        Subspace sample a number of $m = \sqrt{p}$ features from the feature space $p$ at this node
        Choose the best feature (highest Information Gain, lowest variance) to split the data on
        Split the data along the feature
        Remove the feature
    Add the outcome of each tree to **Tree_Outcomes**
**if** Classification **then**
    **Random_Forest_Outcome** ← Majority vote (mode) of the elements in **Tree_Outcomes**
**else if** Regression **then**
    **Random_Forest_Outcome** ← Mean/Median of the elements in **Tree_Outcomes**
**Return Random_Forest_Outcome**

Figure 9: RandomForest Pseudocode (from python-course.eu)

**for each** class $c \in C$ # Calculate $P(c)$ terms
$N_{doc}$ = number of documents in D
$N_c$ = number of documents from D in class c
$logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
$V \leftarrow$ vocabulary of D
$bigdoc[c] \leftarrow$ **append**(d) **for** d ∈ D **with** class $c$
**for each** word $w$ in V # Calculate $P(w|c)$ terms
$count(w,c) \leftarrow$ # of occurrences of $w$ in $bigdoc[c]$
$loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
**return** *logprior*, *loglikelihood*, $V$

Figure 10: Naive Bayesian Algorithm Pseudocode

(Knyszewski, Implementing Naive Bayes for Sentiment Analysis in Python. AI Time Journal))