Comparing Machine Learning Methods in News Classification

Hao Li hli662@wisc.edu
Hao Liang hliang56@wisc.edu
Katherine Fu jfu57@wisc.edu
Yunzikai Chen ychen857@wisc.edu

Abstract

With the traditional newspaper companies starting to offer online subscriptions and the propensity of digital news providers such as Blooberg and HuffPost, the major sources of news for us gradually shift from the traditional newspapers to the online news providers. With the intention to push customized and up-to-dated news pieces to their subscribers, these news providers need to develop a mechanism which can categorize the news pieces accurately into different categories such that that news can be pushed to the subscribers who are most likely be interested in. In this paper, we utilized the dataset obtained from HuffPost, which contains 30,000 news pieces posted on HuffPost from 2012 to 2019, with each news' corresponding categories and text features, and developed seven machine learning models, including K-Nearest Neighbor (kNN), Mltinomial Naive Bayes (MltinomialNB), Random Forest, LinearSVC, etc. to categorize the news into one of the six categories. After a thorough analysis of the accuracy achieved by each model, we determined that the Linear SVC model was a relatively better model for categorizing news pieces based on its higher classification accuracy. Also, by comparing different input values for the machine learning models, we found that keywords and headlines were more useful features than the short descriptions in the process of text classification. Admittedly, the dataset we used was outdated while the news industry is a fast-growing field, our model could still serve as the baseline model for the reference of the future works.

1. Introduction

Digitalization is an unstoppable trend we are facing in the 21st century. Being in this unprecedented era, we are observing more and more applications gradually switch to the mobile end. The news industry, for example, is one of the pioneering fields moving towards digitalization. *The Economics*, an international weekly newspaper, used to be printed only in magazine format, but now, it is published both in online and magazine formats. For the news industry, publishing news on time is crucial, since only the most recent news is newsworthy. Especially for the financial news field, timeliness is very important. For instance, Bloomberg must report finance-related news online promptly so that the stock traders can make adjustments to their portfolio on time. Therefore, presenting and categorizing the content in a timely manner is very important for those news providers to create a faster and well-customized news-browsing experience for their readers.

In this project, we utilized the *News Category Dataset* obtained from the Kaggle website [9], which was collected from the HuffPost website [1] between 2012 and 2019. HuffPost is an American news aggregator and blog. In this dataset, there are 30,000 samples categorized into six different categories and four features (*headline*, *short_description*, *date*, and *keywords*). The categories in the dataset include *Business*, *Food & Drink*, *Parenting*, *Politics*, *Wellness*, and *World News* and we utilized three features (*category*, *headline*, and *short_description*) for training the models. The performances of these features (accuracy, weighted recall, weighted precision, and weighted F-1 score) were compared in the end across all the models. A detailed classification procedure was discussed in the section 5. In short, we first conducted data cleaning process and then split our dataset into the testing, training, and validation datasets. In order to feed our data into seven proposed model displayed in section 4, we employed Python packages such as **Scikit-learn** and evaluate the model using our validation and test sets. The final results were discussed in section 6 and can be seen more clearly in Table 2.

In terms of model development, we selected machine learning models that were most typical and used frequently in the industry. We performed hyperparameter-tuning process in order to get the better performance for each models. Since we experimented in all the useful features, we can be sure that the results we generated were representative and accurate.

2. Motivation

According to the 2021 News Consumption Statistics [2], 2020 is the year that was rated having the most viewers of U.S. news outlets. For example, over 30 million Americans received news from the Evening News Cables, and

roughly half of them obtained information from social media in 2020. Unsurprisingly, young generations, between the age of 18 to 29 years, received the latest news on digital platforms through the internet. With the emergence of the internet, people can easily immerse themselves in tons of information every day. They can access all kinds of news through their cell phones, laptops, even smartwatches. Because of the skyrocketed trend of increasing viewers in digital information, the news media, like Bloomberg, Thomson Reuters, and HuffPost, put more effort into bringing better customer experiences. One of the most challenging aspects of promoting users' experiences is to help customers locate specific news they are interested in and enjoy reading. In the past, newsreaders could choose the news they wanted by subscribing to particular magazines or newspaper. However, being presented with tons of information every day on their phone with unclear categories would make it harder for users to locate the news they are interested in. It is important for news providers to categorize news in an automated and organized format. Moreover, due to the soar of unstructured data and new formats of text resources, the demand for refining models in news classification has also risen drastically in recent years [11]. Unfortunately, the large amount of information makes manual classification inefficient and could easily lead to misclassifications. For example, during a summer internship role at a media company, one of our group members spent three weeks manually classifying the news based on the subtitles and contents. According to him, this inefficient way of classifying is a waste of time and human sources. Nowadays, with the technological advancements and border application of machine learning algorithms and models, we can design models with a faster and much more efficient method in classifying news. With the help of automated machine learning models, such as the models we are going to propose below, media companies could quickly process tons of information and publish the news with correct labels promptly. Through iterative training processes, machine learning models will ensure a high level of prediction rates and effectiveness so that the models can perform well in categorizing each piece of news provided in a shrot period of time.

3. Related Work

Many previous researchers have conducted studies on the topic of applying machine learning models to classify text into different categories, including the classification of news. This section aims to find what other research was conducted on related topics and what can be done on news classification.

When classifying news based on its description or original text, the first problem one needs to work through before building any machine learning classifiers is how to extract the key information out of the descriptions or the original text and use that to classify the news accordingly. Mulahuwaish, A., Gyorick, K., and Ghafoor, K. Z. (2020) [10], for example, deployed the data mining techniques, which can crawl the web and extract specific information to classify them into different categories. They then compared four different machine learning classifiers, Support Vector Machine (SVM), K-Nearest Neighbor (kNN), Decision Tree (DT), and Long Short-Term Memory (LSTM), on classifying web news. Their results suggest that SVM is the best-performing classifier in news classification, while kNN is the worst. In our dataset, each piece of news is categorized into one of the six categories, which is more closer to the real-world setting, where the news categories are more than four. Moreover, we delved into other potential machine learning algorithms, such as Random Forest, GridSearch, to check if we can develop a better-performing model.

By far, we only talked about classifying English-text news, of which there are abundant resources for us to perform analysis and classification. However, it would be much more difficult to apply machine learning methods to classify news written in non-mainstream languages, such as Amharic. According to Dndalie and Haile (2021) [8], lacking language resources is one of the major reasons impeding machine learning techniques to be applied in Amharic document classification. They, therefore, proposed a deep learning model for classifying Amharic news documents; specifically, their model utilized the text vectors matrix to represent the semantic meaning of texts, which were then fed into the embedding layer of a convolutional neural network (CNN) to extract features automatically. Their studies focused on classifying news into six categories and achieved a 93.79% classification accuracy. Based on such high classification accuracy, we could expect machine learning algorithms to be applied in classifying not only English-text news but also news written in other forms of language, given that we have sufficient language resources and can build well-developed models.

The term frequency-inverse document frequency (TF-IDF) is one of the most popular methods in feature extraction. It is useful in that it emphasizes the relationship between each word with the whole text [14]. However, it has been noted by some scholars that the TF-IDF machine learning method does not have the capability to analyze text with its semantic meaning. [4] Specifically, TF-IDF cannot derive the context of the words, and it does not consider the order of the words. In other words, this model employing TF-IDF cannot relate to the sentimental meaning of these texts. This might raise concerns when we are trying to develop machine learning models for all-text data. TF-IDF's downsides do not stop researchers from keeping using them due to their advantages in effectively analyzing the importance of each text.

4. Proposed Method

4.1. Feature Extraction

4.1.1 Bag of Words

We utilized CountVectorizer in scikit-learn for tokenize and vectorize text samples into vectors. CountVectorizer simply employs bag of words algorithm which counts the number of times each word appears in the text summary and puts them into an array of vectors. This is a naive approach, meaning it is not sufficient for analyzing text because it only highlights the importance of word counts instead of their relationship with the entire texts.

4.1.2 TF-IDF

Since counting only the frequency of each word will give some words more weights than others, we experimented Term Frequency Times Inverse Document Frequency (TF-IDF) to resolve this issue. TF-IDF vectorizes words by taking not only frequency into consideration but also their inverse document frequency (how much valuable information a word contributes to the text). The TF-IDF formula can be calculated as below [14]

$$tfide(w,t,T) = tf(w,t) * idf(w,T)$$

where w is each word, t is each text, and T is a set of text.

$$tf(w,t) = log(1 + freq(w,t))$$

$$idf(w,T) = log(\frac{N}{count(w \in t : t \in T)})$$

4.2. Machine Learning Method

In this section, we presented seven models we utilized in classifying three text features into news categories. We explained what each model does, what parameters we employed for building models, and what results each model has generated. We displayed Table 1 to show all the parameters and Table 2 for the results.

4.2.1 K Nearest Neighbor Classifier

For our baseline model, we utilized the k-nearest neighbors non-parametric algorithm (KNN) for text-based classification. This method classifies data samples by checking the labels of their nearest neighbors and then predicting labels by the most majority vote. We believed the KNN model is the most straightforward and intuitive model to classify our text information. We chose K (=3) as the parameter for the number of neighbors considered. The result can be referred to in the summary table.

Table 1. Hyperparameters setting

Models	**Parameters**
KNeighborsClassifier	n_neighbors=3
MultinomialNB	default
OneVsRestClassifier	default
LinearSVC	tol=1e-5
RandomForestClassifier	n_estimators=100
StackingCVClassifier	RandomForestClassifier: n_estimators=100 XGBClassifier: verbosity=0 use_label_encoder=False LogisticRegression : max_iter=1000
GridSearchCV 1	StackingCVClassifier: use_probas=[True, False] drop_proba_col=[None, 'last'] xgbclassifier_max_depth=[4, 6] randomforestclassifier_ n_estimators =10; cv=2; n_jobs=1; verbose=2
GridSearchCV 2	DecisionTreeClassifier: max_depth=[1, 2, 3, 4, 5, 10, 15, 20, None] criterion=['gini', 'entropy'] scoring='accuracy' cv=10

4.2.2 Naive Bayes Classifier

We applied the Naive Bayes classifier (MultinomialNB) to compare the classification accuracy of each feature on predicting the labels. This classifier assumes that all the features are independent, and it predicts class outcomes by computing the conditional probabilities and "space of hypotheses." Conditional probabilities are calculated based on the Bayes' theorem and "space of hypotheses" by locating the possible hypotheses training data [6, 12]. The advantage of using this method is that it performs well in classifying multiple classes, which is why we used this model; however, one disadvantage is that it is computationally inefficient and the assumption of independence cannot hold all the time.

4.2.3 OneVsRest Classifier

To classify data samples into multiple classes, we can employ one way by converting multiple-classification problems into binary-classification algorithms. OneVsRest Classifiers achieves this by splitting different classes into multiple binary classes to train binary models on the datasets [7]. It does require us to create models for each class, thus, requires more computational resources.

4.2.4 LinearSVC

We utilized Linear Support Vector Classifier (LinearSVC) because this method has shown good performance in working with a large number of datasets. LinearSVC is more advantageous than the regular SVC because it applies a loss function to penalize incorrect class predictions for getting a better result. In our example, we set tol to e-5. It represented tolerance for the stopping criteria.

4.2.5 RandomForest Classifier

Random Forest model is another method we applied. This method combines multiple single classification decision trees. It trains the dataset through sampling with bootstrapping and builds the trees by randomly selecting features from dataset in each splitting step. In our experiment, we utilized default number of estimates (=100) for the number of decision trees trained and Gini Impurity as the metrics for information gain, which would give us the best split when Gini is the maximum. The formula for Gini is shown below

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

4.2.6 XGB Classifier

In our project, we applied XGBClassifier inside of 4.2.7 StackingCVClassifier. It implements "gradient boosted decision trees" [5]. Gradient boosting learns from and corrects the previous mistakes and builds models that average the effects of the wrong predictions. More information about how we utilized this classifier can be referred in the next subsection.

4.2.7 StackingCV Classifier

We fit the StackingCVClassifier when doing 4.2.8 GridSearchCV. In this subsection, we first introduced StackingCVClassifier and in the next one, we will talk more about GridSearchCV. Stacking cross-validation classifier combines both stacking ensemble method and cross-validation re-sampling processes to yield the best prediction. The stacking method helps us obtain the final result by going through two-level of prediction with different models utilized [3]. In our case, we employed 4.2.5 RandomForestClassifier, 4.2.6 XGBClassifier, as the first-level classifiers to get multiple predictions, and then we fit them into the second-level meta-classifier which was LogisticRegression in our case to get the final prediction. Cross-validation enables us split dataset into multiple folds so that we can better extract information from the features. In our project, we utilized the default cv (=2) iterations.

4.2.8 GridSearchCV Classifier

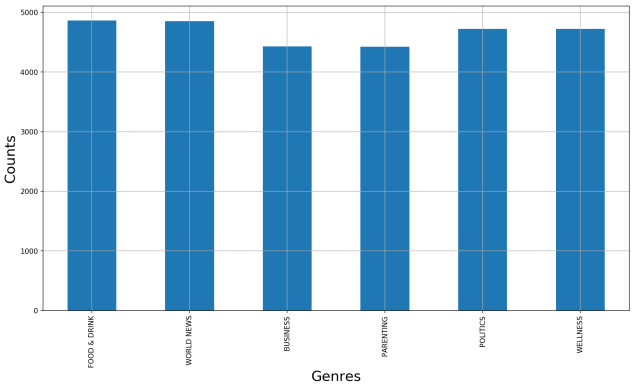
GridSearchCV exhaustively runs models with all the combinations of parameters provided and compares their performances to find the one with the best score [13]. It could be time-consuming to complete the entire process. In our projects, we presented two GridSearchCV with different parameters and classifiers – one simply using DecisionTreeClassifier while the other 4.2.7 StackingCVClassifier. For each classifiers, we provided a set of hyper parameters for tuning.

5. Experiments

In our experiments, we utilized three features including *short_description, keywords,* and *headline* in the dataset to fit into all the models listed in section 4. Thus, we have conducted in total 3 x 7 = 21 model classification steps. Each of the features were processed in the identical ways: being split into training, test, and validation dataset, being fit into the seven models with the same parameters, being analyzed and evaluated using accuracy, F-1 scores, precision, and recall. The reason of ensuring the similarity of the processes is to help us compare the effectiveness of each feature and models in classifying the news categories. In the following sub-sections, we will detailedly illustrate each step.

5.1. Dataset

Figure 1. Dataset Class Bar Chart



The dataset we employed here is from the News Category Dataset on the Kaggle website. This dataset was collected from HuffPost over the years from 2012 to 2018. It contains 30000 original samples and 5 columns which include *category*, *short_descriptions*, *headline*, *links*, and *keywords*. There are overall six categories, and we utilized all of them as the labels for classification. By visualizing the Figure 1, we can see that each category contains relatively the same amount of data samples, ensuring a balance. We applied *headlines*, *short_descriptions*, and *keywords* as features as we believed all of them contain useful information

for classification. In the end, we compared their performance to see which one could yield us better results. Even though the dataset was cleaned up previously, we still conducted pre-processing steps for better usage and analysis.

5.2. Data Preprocessing

5.2.1 Drop NaN values

We first dropped the lines in the dataset that contains the NaN values. By reading the data, we identified 1982 rows that contain NaN values in their "keywords" column. After deleting these rows, we remained 28018 samples.

5.2.2 Remove special characters

We remove characters including all punctuation marks and other special signs for all the features. One example text before the removal process is *If you want to be busy, keep trying to be perfect. If you want to be happy, focus on making a difference.* After the removal process, it becomes *If you want to be busy keep trying to be perfect If you want to be a happy focus on making a difference.*

5.2.3 Remove stop words

The last step in the pre-processing step was to remove stop words including "a", "the", "is", "are" and etc. One example could be *Resting is part of the training I have confirmed what I sort of already knew I m not built for running streaks I m built for hard workouts three to five days a week with lots of cross-training physical therapy and foam rolling But I have also confirmed that I m stubborn with myself* and it changes to *resting part training confirmed sort already knew built running streaks built hard workouts three five days week lots cross-training physical therapy foam rolling also confirmed stubborn.*

5.3. Data Splitting

We shuffled our data set and split it into training and test datasets. Specifically, 70% of whole dataset was our training set (which 14% is our validation set), and 30% of whole dataset was our test set. The step allowed us to estimate performances of each model we applied in section 4.

5.4. Model Fitting

After data pre-processing and data splitting process, we prepared a training data set and we began to fit the training data into our previous mentioned classifiers, including the K-NN (K Nearest Neighbors), Mutinomial NB (Naive Bayes), OneVsRest, Linear SVC, Random Forest, Stacking CV, Grid Search CV, XG Boosting. For each model, we ran these classifiers on all three features — *short_descriptions*, *keywords*, and *headline*. After running the models, we calculated the accuracy on the training set, evaluated the accuracy, f-1 score, precision and recall results with our validation set and finally tested our models on the test set. The performances of each model with each feature fitted in were presented in the section 6.

5.5. Software and Hardware

We mainly utilized Pandas, Numpy, and Scikit-learn from Python to implement all the models and preprocessing steps. In addition, we applied Jupyter notebook, Google Colab, and terminal to run the scripts on our personal laptops. All the scripts were uploaded to our project GitHub.

We found and cited online machine learnings sources and scholarly articles from Google Scholar, Medium, Science Direct, and others to learn more about various models and methods.

6. Results and Discussion

We fitted three features into seven models and ensemble models. The final results with all the performance metrics are displayed in Table 2. We also reported data both by features and models individually. By observing the results by features (Table 3), we calculated the average accuracy performances of all the models. The result showed that *keywords* had the largest accuracy results in all the training, validation, and test datasets. Since *keywords* were the most useful text features extracted from the longer summaries, we can conclude that the large accuracy makes sense. By looking at the data from Table 2, we can also observe that LinearSVC model showed the highest test accuracy in *keywords* (Accuracy=78.33%; F-1=78.33%; Precision=78.37%; Recall=78.33%). *Short description* gave the worst results for both training and test sets. Even though this feature contained more information, the bad performance could result from the noise in the text data that interfered with the classification processes.

By averaging the results by models (Tabel 4), we can see that KNeighborsClassifier gave the worst performance results across three features for training, validation, and test sets. While DecisionTreeGridSearchCV displayed extremely high accuracy on training and validation, its test accuracy was really low. The reason for this result might associate with over-fitting. LinearSVC gave the best on the test sets.

If we look at models and features together (Table 2), we can see that using *headlines* as the feature and fitting into the DecisionTree + GridSearchCV gave us the best result on the validation dataset (Accuracy=F-1=Precision=Recall=99.90%) and LinearSVC displayed the best result on the test data (Accuracy=80.56%; F-1=80.59%; Precision=80.69%; Recall=80.56%).

Table 2. Performance Comparison

Features	Models	Train	Valid	Test
		Accuracy	Accuracy F-1 Score Precision Recall	Accuracy F-1 Score Precision Recall
Headlines	KNeighborsClassifier	59.26%	30.92% 27.61% 66.62% 30.92%	30.16% 26.62% 68.24% 30.16%
	MultinomialNB	87.87%	77.72% 77.54% 77.53% 77.72%	78.46% 78.35% 78.36% 78.46%
	RandomForestClassifier	94.99%	75.45% 75.61% 75.99% 75.45%	75.57% 75.69% 76.21% 75.57%
	OneVsRestClassifier	90.37%	90.82% 90.77% 90.83% 90.82%	79.47% 79.35% 79.34% 79.47%
	LinearSVC	97.67%	97.45% 97.45% 97.45% 97.45%	80.56% 80.59% 80.69% 80.56%
	StackingCV GridSearchCV	95.86%	95.87% 95.83% 95.90% 95.87%	76.71% 76.92% 77.76% 76.71%
	DecisionTree GridSearchCV	99.88%	99.90% 99.90% 99.90% 99.90%	71.38% 71.36% 71.80% 71.38%
Keywords	KNeighborsClassifier	80.51%	80.30% 80.31% 81.43% 80.30%	65.05% 65.07% 66.71% 65.05%
	MultinomialNB	88.20%	88.10% 88.10% 88.21% 88.10%	76.98% 76.98% 77.09% 76.98%
	RandomForestClassifier	99.56%	99.75% 99.75% 99.75% 99.75%	73.59% 73.70% 74.88% 73.59%
	OneVsRestClassifier	88.93%	88.81% 88.80% 88.88% 88.81%	77.69% 77.69% 77.75% 77.69%
	LinearSVC	96.47%	96.15% 96.15% 96.17% 96.15%	78.33% 78.33% 78.37% 78.33%
	StackingCV GridSearchCV	96.27%	96.69% 96.67% 96.70% 96.69%	74.02% 74.01% 74.39% 74.02%
	DecisionTree GridSearchCV	99.56%	99.62% 99.62% 99.62% 99.62%	67.93% 67.91% 68.45% 67.93%
Short descriptions	KNeighborsClassifier	28.63%	29.24% 23.02% 80.73% 29.24%	21.20% 12.90% 77.41% 21.20%
	MultinomialNB	86.35%	86.57% 86.58% 86.77% 86.57%	72.82% 72.84% 73.53% 72.82%
	RandomForestClassifier	99.90%	99.77% 99.77% 99.77% 99.77%	69.51% 69.65% 70.57% 69.51%
	OneVsRestClassifier	87.28%	87.56% 87.55% 87.69% 87.56%	73.82% 73.80% 74.28% 73.82%
	LinearSVC	97.67%	97.55% 97.55% 97.56% 97.55%	73.16% 73.10% 73.09% 73.16%
	StackingCV GridSearchCV	95.05%	95.44% 95.42% 95.43% 95.44%	70.91% 70.96% 71.23% 70.91%
	DecisionTree GridSearchCV	99.91%	99.87% 99.87% 99.87% 99.87%	60.08% 60.06% 60.14% 60.08%

We also observed from the table that some evaluation methods displayed really disparate results. For example, the accuracy for KNeighborsClassifier using *headlines* is 30.92% but the precision is 66.62%, which is way different from the rest of the metrics. The similar situation can be seen in KNeighborsClassifier with *short description*. This could be due to the fact that either there are fewer false negatives or the true positives are large.

Table 3. Average Accuracy Results across Features

Features	Training	Validation	Test
Headlines	89.41%	81.16%	70.39%
Keywords	92.79%	92.77%	73.37%
Short Descriptions	84.97%	85.14%	63.07%

Table 4. Average Accuracy Results across Models

Models	Training	Validation	Test
KNeighbors Classifier	56.13%	46.49%	38.80%
MultinomialNB	87.47%	84.13%	76.09%
RandomForest Classifier	98.15%	92.32%	72.89%
OneVsRest Classifier	88.86%	89.06%	76.99%
LinearSVC	97.27%	97.05%	77.35%
StackingCV GridSearchCV	95.73%	96.00%	73.88%
DecisionTree GridSearchCV	99.51%	99.79%	66.46%

7. Conclusions

In conclusion, by combining results from the above analysis, we can infer that LinearSVC was the best model for our project. Both *keywords* and *headlines* were the most useful features for classification. It is important to notice the difference among different feature variables as well as their relationships to the models. Since *keywords* and *headlines* were both generated from the *short description*, we cannot conclude that *short description* was not essential in our classification process. Instead, it provided valuable information and was the basis for the classification. Our project could be generalized to and used by other text information. Based on our results, we discovered that it is of great significance to extract keywords or short sentences from the summary texts before conducting the classification procedure instead of fitting the summary texts directly into the models

8. Future Direction

Our model in news classification only provided a baseline for further exploration. Our datasets were outdated, as each line was news from 2012-2019. With fast-developed technology, there are much more news categories than before. Also, in the actual scenario, it is very common to have more than one classification for each line of news. However, in our model, the datasets we chose contained one classification for each news. As discussed in Relation Work section, it is impotant to classify text from different languages but our project only dealt with English texts. Thus, a more practical model could be developed when we have more real-time, up-to-date, and international datasets. What's more, in our projects, we only tried out certain set of parameters and models but it is possible that by further tuning the hyperparameters, we can get a more favorable result.

9. Acknowledgements

We want to thank our great Prof. Sebastian Raschka for his great feedback and constructive suggestions in model selections and training process during the hour offices. Additionally, we want to thank Prof. Raschka's well-structured lectures and lecture notes as well as the helpful coding samples, homework, and quizzes. In addition, we want to thank every teammate in this group, where everyone made significant contributions in the course of this semester. Lastly, we would like to thank everyone who will read our report and give our peer review feedback.

10. Contributions

In this project, we all demonstrated our passions and responsibilities in contributing to this project. Yunzikai Chen and Hao Li took responsibility in fitting, and running models. Katherine Fu was mainly responsible for conducting data cleaning process, summarizing, analyzing, and reporting the results. Hao Liang attempted to find resources on each topics and helped set up the models. All of us engaged in writing the final report. Although we had various challenges in the course of accomplishing our project, everyone fully showed their competencies in solving every problem we encountered and finally completed this project promptly.

References

[1] Huffpost website.
[2] News consumption across social media in 2021.
[3] Stackingcvclassifier.
[4] Understanding tf-idf for machine learning.
[5] J. Brownlee. How to develop your first xgboost model in python. 8 2016.
[6] J. Brownlee. A gentle introduction to the bayes optimal classifier. 12 2019.
[7] J. Brownlee. One-vs-rest and one-vs-one for multi-class classification. 4 2020.
[8] D. Endalie and G. Haile. Automated amharic news categorization using deep learning models. 7 2021.
[9] R. Misra. News category dataset.
[10] A. Mulahuwaish, K. Gyorick, K. Z. Ghafoor, H. S. Maghdid, and D. B. Rawat. Efficient classification model of web news documents using machine learning algorithms for accurate information. 8 2020.
[11] M. I. Rana, S. Khalid, and M. U. Akbar. News classification based on their headlines: A review. In *17th IEEE International Multi Topic Conference 2014*, pages 211–216, 2014.
[12] I. Rish. An empirical study of the naive bayes classifier. Technical report, 2001.
[13] R. Shah. Tune hyperparameters with gridsearchcv. 6 2021.
[14] B. Stecanella. Understanding tf-id: A simple introduction. 5 2019.