A Thesis

Submitted to the Graduate Faculty of the

National University, Department of Engineering & Computing

in partial fulfillment of the requirements for the degree of

Masters of Science in Data Science

Prepared By:

Arun Kumar Mallikarjuna

Ryan Paw

Catherine Rohrer

National University

October 2021

MASTER'S THESIS APPROVAL FORM

We certify that we have read the project of Arun Kumar Mallikarjuna, Ryan Paw, and Catherine Rohrer entitled DEFEATING FAKE NEWS USING NLP AND MACHINE LEARNING TECHNIQUES and that, in our opinion, it is satisfactory in scope and quality as the thesis for degree of Master of Science in Data Science at National University.

Approved:

Jennifer Rabowsky, Capstone Project Advisor Date
Machine Learning Engineer, NLP

Dr. Kevin Duffy-Deno, Capstone Project Advisor Date
Adjunct Professor, Department of Engineering and Computing
National University

Dr. Jodi Reeves, Program Director Date
Associate Dean, Department of Engineering and Computing
National University

Dr. Mario Missakian, Capstone Instructor Date
Adjunct Professor, Department of Engineering and Computing
National University

Abstract

In a digital age where people are connected through technology, our worldview is shaped by the information we see, read, and hear. The objective for this study was to create a predictive model that explores textual properties to classify news articles as fake or fact. The first model (Model 1) investigated only the news headline and the second model (Model 2) investigated the news headline and the body of the article as features. Before text was analyzed by the model, we implemented data cleaning techniques, such as stop word removal and stemming, and a feature extraction technique, term frequency-inverse document frequency (TF-IDF). The project trained multiple real-world datasets using the following machine learning algorithms and ensemble method: logistic regression, support vector machine, k-nearest neighbors, and voting ensemble. For Model 1 and 2, support vector machine performed the best at classifying fake news with an accuracy of 86% and 94%, respectively.

Table of Contents

Chapter 1: Introduction 1

Background 1

Problem Statement 3

Objectives 3

Limitations of the Study 4

Chapter 2: Literature Review 5

Previous Fake News Models 5

Data Preprocessing 6

Algorithms 8

Voting Ensemble 10

Chapter 3: Methodology 15

Data Description 16

Data Preprocessing 17

Stemming 20

Term Frequency-Inverse Document Frequency (TF-IDF) 21

Train-Validate-Test Split 22

Logistic Regression 22

Support Vector Machine (SVM) 23

K-Nearest Neighbors (KNN) 24

Voting Ensemble 24

Parameter and Hyperparameter Tuning 25

Performance Metrics 25

Accuracy 26

Recall 26

Precision 26

F1-score 27

AUC-ROC Curve 27

Feature Weights Using Bi-grams 27

Chapter 4: Results & Analysis 30

Model 1 Bi-gram Results 30

Model 2 Bi-gram Results 32

Accuracy Results 34

Precision, Recall, F1-score Results 36

AUC-ROC Curve Results 39

Chapter 5: Conclusions and Recommendations 41

Appendix A1: Python Code - Model 1 - Data Preparation/Data Preprocessing 43

Appendix A2: Python Code - Model 1 - Data Vectorization/Train-Validate-Test Split 45

Appendix A3: Python Code - Model 1 - Model Training 46

Appendix A4: Python Code - Model 1 - Feature Weights 48

Appendix A5: Python Code - Model 1 - Model Hypertuning 49

Appendix A6: Python Code - Model 1 - Model Testing 51

Appendix B1: Python Code - Model 2 - Data Preparation/Data Preprocessing 52

Appendix B2: Python Code - Model 2 - Data Vectorization/Train-Validate-Test Split 54

Appendix B3: Python Code - Model 2 - Model Training 55

Appendix B4: Python Code - Model 2 - Feature Weights 56

Appendix B5: Python Code - Model 2 - Model Hypertuning 57

Appendix B6: Python Code - Model 2 - Model Testing 59

References 61

List of Tables

Table 1. Previous research on fake news detector models.. 11

Table 2. Accuracy percentages for each algorithm between Model 1 and Model 2.................... 35

Table 3. Precision, recall, and F1-score percentages for each algorithm. 37

List of Illustrations

Figure 1. Workflow diagram for our fake news detector model 15

Figure 2. Top 10 fake features and their corresponding weights in Model 1. 31

Figure 3. Top 10 fact features and their corresponding weights in Model 1. 31

Figure 4. Top 10 fake features and their corresponding weights in Model 2. 33

Figure 5. Top 10 fact features and their corresponding weights in Model 2. 34

Figure 6. Accuracy comparison between Model 1 and Model 2 for each algorithm 36

Figure 7. Precision, recall, and F1-score comparison between Model 1 and Model 2. 38

Figure 8. AUC-ROC curves for each algorithm in Model 1 (Headline Only). 39

Figure 9. AUC-ROC curves for each algorithm in Model 2 (Headline and Body). 40

Chapter 1: Introduction

Background

News serves as an important part of our lives. The American Press Institute defines the purpose of journalism as providing “citizens with the information they need to make the best possible decisions about their lives, their communities, their societies, and their governments” (Dean, 2021, para. 3). The news we receive can impact our careers, financial decisions, and relationships. While it is not a new concept, the term “fake news” exploded on the scene following the 2016 presidential election. McGonagle (2017) defines fake news as, “information that has been deliberately fabricated and disseminated with the intention to deceive and mislead others into believing falsehoods or doubting verifiable facts” (para. 3).

In our digital age, worldview is shaped by the information we are deluged with. A study by Ahler (2006) explored the shift of news consumption from the traditional media to online news media. This includes, but was not limited to elections, celebrity gossip, natural disasters, and health tips. It has become easier to acquire the latest information with a swipe of a finger. The spread of COVID-19 is an example where fake news spread about the virus’ origins, how the virus behaved, and how people should respond to the virus. A 2020 study by Hua, analyzed the pandemic from a data perspective and described two wars being fought: a COVID-19 epidemic and an infodemic. When there is conflicting information, misinformation, or manipulative information, it is challenging to discern between fake or fact.

Misinformation can lead to immensely dangerous outcomes. One illustration of this was Pizzagate. Pizzagate was a malicious story that linked politics to a sex trafficking ring that reportedly ran out of a pizza shop in Washington, D.C. This fake story first appeared on Facebook on October 29th, 2016. On December 4th, 2016, Edgar Maddison Welch opened fire in the pizza shop with an AR-15. Over a five-week period, the Pizzagate story was shared on Twitter approximately 1.4 million times (Bleakley, 2021). Recent examples of fake news include stories about COVID-19 and the events leading up to the storming of Capitol Hill.

Dissemination of information rapidly occurs on social media. People unknowingly participate in sharing false stories by retweeting, liking, and resharing the information (Liu, 2019). The perception exists that social media accounts for the majority of fake news and research supports this. A study conducted by Naeem et al. (2021) concluded that social media accounted for 50.5% of the fake news articles they studied. The study explored 1,225 fake news stories surrounding COVID-19 and concluded they were a risk to public health.

According to Mitchell et al. (2020), one-in-five U.S. adults obtain their political news primarily on social media. While this is slightly lower than the percentage of adults relying on news websites for this information, this number includes all age brackets. The percentage increases for people between the ages of 18 and 49. The same study concluded that 48% of people between the ages of 18 and 29 and 40% of people between the ages of 30 and 49 received their political news from social media. This increases the likelihood that people under the age of 50 will be exposed to fake news.

After the 2016 US election, fake news on social media was being discussed more than ever. Some argued the best way to defend oneself from fake news on Facebook was to trick the existing ML algorithms (Patterson, 2016). It is known that Facebook adds content similar to what has already been viewed. So, if people continued to view articles from reputable sources, Facebook would show more reputable sources and vice versa. When Facebook first started to support detecting fake news, they partnered with third-party companies such as Snopes and Politifact (Bloomberg, 2017). These companies employ trackers to research articles. Twitter and Facebook also allow people to identify and tag news as fake. However, because this method still relied on human intervention, there was still a large margin of error.

Problem Statement

Fake news is a major problem and there are social and economic repercussions are caused by this. As technology improves, it also means fake, manipulative content also increases. It makes us question if the information we receive is credible. While there have been many improvements in existing algorithms, the issue still exists. Identifying fake information online is difficult. This issue requires more knowledge and tools to help people fight this infodemic.

Objectives

The aim for this study was to create a predictive model to classify news articles as fake or fact using machine learning algorithms and a public dataset. One objective was to describe how the dataset features impact the model's output. Another objective was to understand the difference in the performance of the individual algorithms. The final objective was to identify features or processes that would improve the model.

Limitations of the Study

This study was confined by the datasets chosen. News on social media can contain text, images, audio, and videos, but in this study, the model analyzed only text. Existing studies included more features such as news sources, history of the news spreading, and date/time data. However, our study was limited to headline news titles and several lines of the article's body paragraphs. While the dataset included the author as a feature, there was too much missing data to incorporate into the model. Downsampling the model for the author feature would reduce the number of records to a set that would be too small. Also, further investigation found that imputing the author feature made minimal impact to the model. The same issue existed for the source and URL of the news article as features. The chosen datasets included a label feature identifying articles as fake or fact. Therefore, the assumption was made that the data originator accurately labeled the articles fake or fact.

Chapter 2: Literature Review

Previous Fake News Models

A number of studies have focused on detection and classification of fake news using various machine learning algorithms on different kinds of datasets. The pre-processing and methods are shown in Table 1. A study by Ahmad (2020) explored textual properties on four fake news datasets including politics, entertainment, technology, and sports. The data was used to train individual machine learning algorithms and a combination of machine learning algorithms using different ensembling techniques. Ahmad's research group found that ensemble learners showed an overall better score in accuracy, recall, precision, and F1-score.

A study by Fauzi et al. (2019) created a fake news model using data extracted from Twitter to achieve a model accuracy of 78%. This included features such as, number of retweets, number of hashtags, URLs, provocations, feuds, anxieties, and unbalanced news. Fauzi's group utilized term frequency-inverse document frequency (TF-IDF) to quantify the data and support vector machine (SVM) algorithm to analyze the data.

Kulkarni (2021) and his research group used fake news data from several social media sources to compare the performance of different machine learning algorithms. The algorithms used were: random forest, logistic regression, decision tree, KNN, and gradient booster. Fauzi et al.(2019) prepared their data by using TF-IDF to quantify their data. By converting documents into a vector, the computer can understand the word as a numerical value. This will allow the machine learning algorithms to analyze the text and determine if the news is fake or fact.

Kulkarni’s group found that logistic regression, random forest, and KNN were the top three performers for identifying fake news.

A study by Aslam et al. (2021) utilized two deep neural network models to classify fake news using data from Politico.com. Data from Politico.com has been commonly used in related fake news studies, such as Khan et al. (2021), Shu et al. (2020), and Brasoveanu (2020). Aslam et al.’s first deep learning model was designed with 10 fully connected dense layers with an input of nine feature variables. Their second deep learning model was a neural network with nine layers with an input of 200 per size of the vector for each word. In comparison to other studies that used the same data from Politico.com, Aslam’s group achieved the highest accuracy of 90% using only the statement feature. The statement feature is the headline title of the news article, which is typically one sentence long.

Data Preprocessing

A study by Thota et al. (2018) applied various text preprocessing steps to convert data into a form ready for modeling. They applied data preprocessing methods to both headlines and articles. They started their data preprocessing stage with stop word removal, which helped them remove the most common words in a document that did not provide much context. Removing stop words helped them improve their model processing time since the analysis was performed on less data. They used the Natural Language Toolkit Library (NLTK) to remove the stop words. Thota also removed punctuation in their data as it did not provide much value. They implemented stemming in their preprocessing stage to remove prefixes and suffixes. Stemming helped reduce inflectional forms and in some cases derivationally related forms of a word to a common base form. To convert the raw text into numerical features, they experimented with two techniques: bag of words and TF-IDF. Using a finely tuned TF-IDF model, they were able to achieve an accuracy of 94.1% on their test data.

A study by Hakak et al. (2021) applied tokenization, stemming, and stop word removal methods to remove the noise from the data. They also used the NLTK for data cleaning. They first applied tokenization to split the raw text of data into a list of tokens. After tokenizing, they used stop word removal to remove the insignificant words to reduce the noise in the text classification. Hakak et al. (2021) then used a stemming method to reduce the words to its root. Stemming also helped reduce the frequency of derived words. After the data cleaning stage was complete, they used an ensemble approach for training and testing purposes. They used the following machine learning algorithms: decision tree, random forest, and extra tree. Precise feature selection and hyperparameter tuning were two important phases that contributed to achieving better performance. Their experimentation with the model yielded an accuracy of 44.15% on their first dataset (ISOT dataset) and an accuracy of 100% on their second dataset (LIAR dataset). Another method to improve accuracy is through the use of n-grams. A study by Furnkranz (1998) shows that utilization of n-grams increases the performance of the model.

A study by Jianqiang et al. (2017) applied six preprocessing methods to analyze twitter sentiments. Sentiment analysis is the process of automatically detecting whether a text segment contains emotional or opinionated content, and it can determine the text's polarity. Twitter sentiment classification aims to classify the sentiment polarity of a tweet as positive, negative, or neutral. They replaced negative mentions by transforming ''won't'', ''can't'', and ''n't'' into ‘‘will not’’, ‘‘cannot’’, and ‘‘not’’, respectively. They removed URL links in the corpus. Twitter short URLs were expanded to URLs and tokenized. They reverted words that contain repeated letters to their original English form. Words with repeated letters, e.g., ‘‘coooool’’, are common in tweets, and people tend to use this way to express their sentiments. Here, a sequence of more than three similar characters were replaced by three characters. In general, numbers are useless when measuring sentiment, so they were removed from tweets to refine the tweet content. Jianqiang and Xiaolin’s group also removed stop words. They expanded acronyms to their original words by using an acronym dictionary. Acronyms and slang are common in tweets but are ill-formed words. Experimental results indicated that replacing negation and expanding acronyms can improve the classification accuracy.

Algorithms

A study by Gravanis et al. (2019) created a model for fake news detection using content based features and machine learning (ML) algorithms. In this study, they compared four classifiers: Naive Bayes, SVM, Decision Trees, KNNs. In addition, they explored two ensemble methods (AdaBoost and Bagging). They evaluated the performance of their algorithms by assessing the accuracy of each one. Evaluation experiments concluded that ensemble methods (AdaBoost & Bagging) and SVM performed the best and were very close in ranking. Their chosen features combined with their ML algorithms obtained an accuracy up to 95% over multiple fake news datasets. AdaBoost performed the best, SVM performed second, and Bagging performed third. Each of these three algorithms had similar accuracies and did not have a statistically significant difference. Such results prove that the classification of articles according to their truthfulness is possible by selecting proper features and suitable ML algorithms. Their experimental results showed that the use of an enhanced linguistic feature set with word embeddings, along with support vector machine (SVM) and ensemble algorithms, are capable of classifying fake news with higher accuracy.

A study by Mahabub (2020) proposed an intelligent detection model to deal with news classification of both real and fake. An ensemble voting classifier was used for detection. In this study, eleven mostly well-known machine-learning algorithms like Naïve Bayes, KNN, SVM, Random Forest, Artificial Neural Network, Logistic Regression, Gradient Boosting, Ada Boosting, etc. were used for detection. After cross-validation, they used the best three machine learning algorithms in the Ensemble Voting Classifier. Their results demonstrated that Ensemble Voting classifiers demonstrated better sufficiency scores which stood out from the results procured by the individual classifiers The experimental outcomes affirm that their proposed framework can accomplish 94.5% outcomes as far as accuracy. The other parameters like ROC score, precision, recall and F1-score were also outstanding. The proposed recognition framework can effectively find the most important highlights of the news.

A study by Khanam et al. (2021) made an analysis of the research related to fake news detection and explored traditional machine learning models to choose the best method. Khanam's group created a fake news model using supervised machine learning algorithms using tools like Python's Scikit-learn and NLTK for textual analysis. They used the Scikit-learn library in Python to perform tokenization and feature extraction of text data by using packages, such as CountVectorizer and TiffVectorizer. When the data was cleaned, they tested the data by using a dataset from Kaggle (LIAR dataset). Several ML algorithms were utilized in their research: K-Nearest Neighbors (KNN), Linear Regression, XGBoost, Naive Bayes, Decision Tree, Random Forests and Support Vector Machine (SVM). Accuracies between all the algorithms were similar and between about 68-76%.

Voting Ensemble

Voting ensemble is a method that combines predictions from multiple machine learning models. It is a common technique to help improve model performance, since it applies the “two heads are better than one” ideology. Multiple algorithms may tackle a problem better than an individual algorithm. In our study, a voting ensemble allows each algorithm to output a prediction if the news is either fake or fact. Then, the majority vote of all the algorithms will determine if the news is either fake or fact. There are two approaches for a voting ensemble: hard voting and soft voting. Hard voting takes the sum of votes from each algorithm, and the label with the largest sum wins. Soft voting takes the probability from each algorithm, and the label with the highest probability wins. In our study and related studies, the hard voting approach was used.

There have been several studies that have used a voting ensemble to improve their model performance. Ahmad et al’s (2020) study compared three different ensemble classifiers to individual algorithms for their fake news dataset: Bagging, boosting, and voting ensemble classifiers. Each ensemble classifier performed better than the individual classifiers (logistic regression, linear SVM, multilayer perceptron, K-nearest neighbors, and random forest). This study measured each classifier based on accuracy, precision, recall, and F1-score.

Osamor & Okezie (2021) also used a voting ensemble to increase their model's performance. They developed a classification model using machine learning algorithms to aid in the diagnosis of tuberculosis. The model analyzed tuberculosis gene expression data that was trained using Naive Bayes (NB) and Support Vector Machine (SVM) algorithms. Osamor & Okezie used a voting ensemble with NB and SVM, which increased the performance of their model. Their voting ensemble classifier performed with an accuracy of 95%, which performed better than 92% and 87% from SVM and NB, respectively.

A study by Satapathy et al. (2017) used a voting ensemble method to analyze electroencephalogram (EEG) signals to detect an epileptic seizure. They trained neural networks and support vector machines (SVM) to classify EEG signals to label epilepsy patients. The neural networks trained were multi-layer perceptron, probabilistic neural network, radial basis function neural network, and recurrent neural network. Satapathy et al.'s group found that their individual neural networks performed poorly. However, when they used them together in a voting ensemble, they achieved an average model accuracy of 99.1% at detecting an epileptic seizure.

Table 1

Previous research on fake news detector models.

Researchers	Data Pre-Processing	Methods
Ahmad et. al, 2020	Unknown	Logistic regression (LR) Linear SVM (LSVM) Multilayer perceptron

		K-nearest neighbors (KNN)
		Random forest (RF)
		Voting classifier (RF, LR, KNN)
		Voting classifier (LR, LSVM, CART)
		Bagging classifier (Decision trees)
		Boosting classifier (Adaboost)
		Boosting classifier (XGBoost)
Fauzi et. al, 2019	Case folding	Support Vector Machine (SVM)
	Tokenizing	
	Stop word removal	
	Stemming	
Kulkarni, 2021	Tokenizing	Random forest
	Stemming	Logistic regression
		Decision tree
		K-nearest Neighbors (KNN)
		Gradient booster
Aslam, et. al, 2021	Tokenizing	Deep learning (10 dense layers)
	Lemmatization	
	Stop word removal	
	Word embedding technique	
Thota, 2018	Stop word removal	Dense neural network

	Stemming	
	Bag of words	
	TF-IDF	
Hakak et. al, 2021	Tokenization	Ensemble Method
	Stemming	Decision tree
	Stop word removal	Random Forest
		Extra tree
Jianqiang & Xiaolin, 2017	Replacing negative mentions	Logistic Regression
	Removing URL links	Naives Bayes
	Reverting words to original English form	Support Vector Machine (SVM)
	Removing numbers	Random Forest
	Stop word removal	
	Acronym expansion	
Gravanis et. al, 2019	Unknown	Naives Bayes
		Support Vector Machine (SVM)
		Decision tree
		K-nearest neighbors (KNN)
		Ensemble
		AdaBoost
		Bagging classifier (decision trees)
Mahabub, 2020	Unknown	Ensemble voting

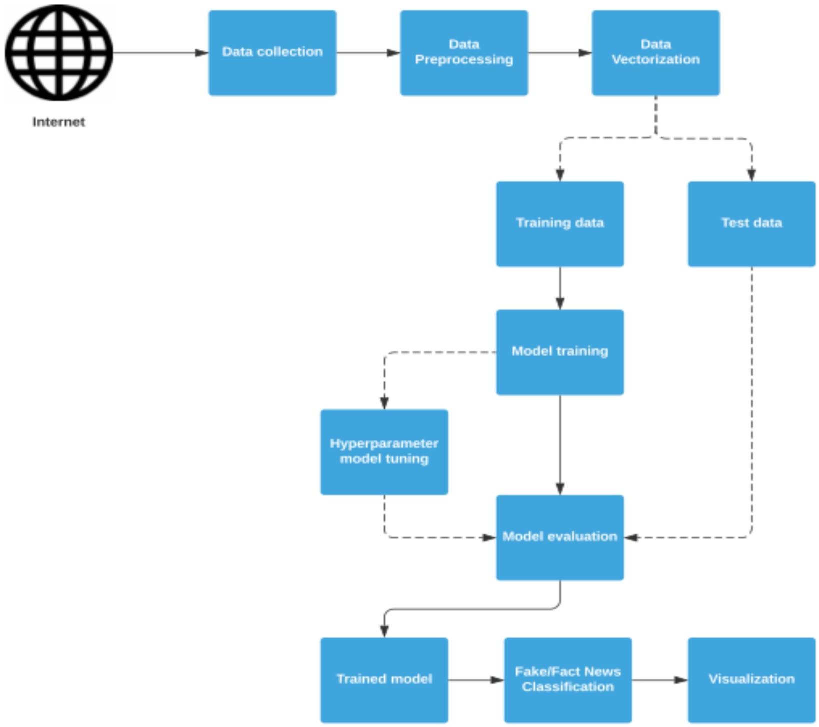
	Naives Bayes
	K-nearest neighbors (KNN)
	Support Vector Machine (SVM)
	Random forest
	Artificial neural network
	Logistic regression
	Gradient Booster
	AdaBoost

Chapter 3: Methodology

This project builds upon previous literature by using a combination of methods to explore the optimal model to predict fake news. A combination of public datasets was used to examine the effects of data balancing and feature selection. The following features were analyzed: *title*, *body*, and *label*. The *label* attribute defined whether the news article was fact or fake. *Title* referred to the article's title and the *body* referred to the content. The approach for this study included: data collection, data preprocessing steps, data vectorization, model training/tuning/evaluation, fake news classification, and data visualization. Figure 1 illustrates the workflow diagram used for this project.

Figure 1

Workflow diagram for our fake news detector model.



Data Description

The datasets used for this study are public and freely available online. The data came from two main sources: Kaggle and FakeNewsNet. Kaggle is a public data platform with a vast community for predictive modeling and analytics. FakeNewsNet is a data repository in Github with news content, social context, and spatiotemporal information for studying fake news on social media (Shu et al., 2020). Each dataset contains fake and fact articles from multiple domains. Fake news articles contain claims that were not factual, while fact news articles contained text that accurately described real-world events. Each dataset contains a *label* feature that identifies the news as “fact” or “fake.” Many of the articles were manually checked with independent fact-checking websites, such as Politifact and GossipCop, to determine if the article was fake or fact.

The first Kaggle dataset (Kaggle1) contained a total of 20,800 articles (UTK Machine Learning Club, 2018). The features in Kaggle1 include *id*, *title*, *author*, *text*, *label*. The accumulation of articles was built from multiple sources on the Internet that contained fake and fact articles from different categories, such as politics and world news. The second Kaggle dataset (Kaggle2) contained a total of 4,009 articles (Ruvika, 2017). The features in Kaggle2 include *URLs*, *Headline*, *Body*, *Label*. The accumulation of articles were built from multiple trusted and untrusted news sources. Some examples of trusted news sources include Reuters, BBC News, New York Times, and CNN. Some examples of untrusted news sources include Daily Buzz Live, Before It’s News, and Activist Post. Kaggle2 covers several news categories, such as politics, entertainment, world news, and sports.

FakeNewsNet is the result of a study by Shu et al. (2020) and his colleagues. The study responds to a lack of comprehensive and community-driven fake news datasets publicly available. FakeNewsNet contains datasets from two sources, Politifact and GossipCop. Both datasets have a combined total of 23,196 articles. The features in both datasets include *id*, *news_url*, *title*, *tweet_ids*. Politifact and GossipCop fact check multiple sources, such as NBC News, CBS News, ABC News, Fox News, and Forbes. Politifact focuses on political news, and GossipCop focuses on entertainment news.

For Model 1, the project started with four datasets extracted from Kaggle and FakeNewsNet. The four datasets include: Kaggle1, Kaggle2, Politifact, and GossipCop. When the four datasets were combined, there were a total of 47,447 records. Model 1's features are: *id*, *headline title,* and *label*. Model 1's independent variable was *headline title* and the dependent variable was *label*. The ratio between fake and fact data was 63.2% fact and 36.9% fake

For Model 2, the project started with two datasets extracted from Kaggle. The two datasets include: Kaggle 1 and Kaggle 2. When the two datasets were combined, there were a total of 24,251 articles. Model 2's features are: *id*, *headline title, body,* and *label*. Model 2's independent variables were *headline title* and *body*, and the dependent variable was *label*. The ratio between fake and fact data was 51.6% fact and 48.4% fake.

Data Preprocessing

This study performed data cleaning to ensure the data had the same features from the different datasets. Our project renamed the column names in all datasets to be the same. For example, the "headline" column was renamed to "title" and "Label" column was renamed to “label” in the Kaggle dataset. It was found that there was missing information and null values. For the Kaggle data, this study removed the 558 records containing nulls. In this case, nulls account for less than 2% of the dataset. This study removed the following features due to the large amount of missing data (+25% missing values): news_url, tweet_ids, URLs, and author.

Data preprocessing is a preliminary step that organizes, sorts, and merges the available information. Raw data could contain a lot of noise, which leads to faulty insights because the system will develop biases and deviations because of the faulty data. This study was able to reduce noise by converting all characters to lowercase, removing punctuation marks, removing stop words and typos, html tags, URLs, and emojis. Preprocessing was completed using NLTK (Natural Language Toolkit), which is open-sourced and widely used in the NLP library. NLTK is a powerful tool complete with different Python modules and libraries to carry out simple to complex natural language processing (NLP). NLTK has a large, structured text known as a corpus. A corpus contains machine-readable text files in a directory produced for NLP tasks. NLTK comes with in-built functions and algorithms such as nltk.tokenize method (for tokenizing text), nltk.stem.porter.PorterStemmer method (popular Porter stemming algorithm), and other such methods.

Lowercase

Lowercasing words helps maintain the consistency of words during NLP tasks and text mining. It is a common approach to lowercasing everything for the sake of simplicity. The

.lower() function from the Python core text library ensured that the whole process was simple and straightforward.

Punctuation

The punctuation from sentences increased the noise that brought ambiguity while training the model. The main reason to separate punctuation by spacing in a sentence was to get a vector representation of words. Thus, this study removed punctuation and replaced it with spaces.

Remove Special Characters

In real-life, human generated text data contain various words with special symbols, and emojis. This study removed this kind of noisy text data before feeding it to the machine learning model. Special characters such as – (hyphen); / (slash); and ? (question marks), do not add any value so these are removed. This project removed special characters because they do not carry useful information while training this model.

Tokenization

Tokenization is the process of splitting the text/string into the list of tokens which is similar to splitting a whole sentence into separate words. In a study by Hakak et al. (2021), tokenizing is considered the first step in natural language processing before the feature extraction process. Tokenizing is an important step because several deep learning architectures for NLP processes the raw text at the token level. Tokenizing splits a piece of text into individual words based on a certain delimiter. It can further be used to convert a string (text) into numeric data so that machine learning models can digest it. This study used the word tokenize module from the NLTK library in Python, which breaks the words into tokens and these words act as an input for the other preprocessing task**s.**

Stop Words Removal

The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead. Stop words are insignificant in a language that create noise when used as features in text classification. They can safely be ignored without sacrificing the intent of the sentence. Removing stop words decreases the dataset size and the time to train the model without a huge impact on the accuracy of the model. Stop word removal can potentially help improve performance because it isolates the significant tokens that remain. Therefore, the classification accuracy is expected to improve. Hakak et al. (2021), removed common words including, but not limited to, *a*, *about*, *an*, *are*, *as*, *at*, *be*, *by*, *from*, *how*, *in*, *is*, and *of.* We used the stop words module from NLTK library in Python to remove the stop words from the text data.

Stemming

There are several variants of words that do not add valuable information and lead to redundancy, ultimately bringing ambiguity when training machine learning models for predictions. Take “He likes to read” and “He likes reading,” for example. Both have the same meaning, so the stemming function will remove the suffix and convert “reading” to “read.”

Hakak et al. (2021) said that stemming is a process of reducing the words to its root, also known as lemma. The main purpose of stemming is to reduce the frequency of derived words. For instance, the words such as reading, read and reader will be reduced to its lemma which is the word “read”. The PorterStemmer tool was used, which is the most common stemming algorithm from the NLTK library in Python**.**

Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is a technique that quantifies words by creating a weight to each word. TF-IDF is used to weigh terms for NLTP tasks since it assigns a value to them according to their importance across all documents. Term frequency is how frequent a word is found in a document. Inverse document frequency adds a weight to each word, where frequently occurring words like “is” or “the,” are weighed less than words like “Biden”. When we multiply the term frequency and the inverse document frequency together, we create a TF-IDF vector. The formula shown below (1) shows how the term frequency is multiplied by the inverse document frequency. TF-IDF is calculated as:

$$Weight_{ij} = tf_{ij} \times log(\frac{N}{df_i}) \tag{1}$$

N is the number of documents, the weight of the term *i* is in document *j*. tf_{ij} is how frequent term *i* is in document *j*. df_i is the number of documents with term *i*.

People may read a news article, word for word, and then understand the article’s message. In contrast, TF-IDF translates each word to a numerical weight to understand the pattern and its relationship to our dependent variable (fake/fact). This vectorization process helps our model focus on the relevance of each word, instead of focusing on “opinion.” By converting the words into vectors, our machine algorithms can understand the relevance of each word to help it classify news articles as fake or fact. This study used the TfidfVectorizer module from Sklearn library in Python to transform text to feature vectors that are used as input to estimator.

Train-Validate-Test Split

After the data was preprocessed and vectorized, it was split into training and test sets (80% for training, 20% for testing). The 80% of data used for training was split again into a subset: a training and validation set (70% for training, 30% for validation). This set was used so the model could be trained and the validation set could be used to assess how generalizable the model was at evaluating fake news. The test set was the last set to use to assess the model’s performance after the model was trained and tuned.

The training data was then inputted into four different machine learning algorithm methods: logistic regression, support vector machine (SVM), k-nearest neighbors (KNN), and voting ensemble. The independent and dependent variables were assigned for each machine learning method to explore their relationships.

Logistic Regression

Logistic regression is a statistical model based on the logistic function. Independent variables were combined to predict the dependent variable. Logistic regression uses a sigmoid function to transform the output to a probability. The goal was to have a favorable probability, while minimizing the cost function. Logistic regression was chosen for this project because it was ideal for binary classification. The project’s dependent variable was binary and categorical, since it determined if the news article was either fake or fact. This made logistic regression a good model for the project to see how well it performed at classifying fake news. The model took in words from news articles and outputted the classification as either fake or fact.

This project assessed the following parameters for logistic regression: C, penalty, solver, and maximum iterations. Different combinations of hyperparameters for logistic regression were assessed to achieve an optimal score. C was the inverse of regularization strength, where smaller values showed stronger regularization. Penalty specified the norm of the penalty, such as L1 regularization and L2 regularization. Solver indicated different types of parameter weights that minimize the cost function in logistic regression. Examples of solver options included stochastic average gradient descent, library for large linear classification, and newton methods. Maximum iterations were the number of iterations for the selected solver to converge.

Support Vector Machine (SVM)

SVM is a statistical model that is also ideal for binary classification problems. SVM creates a hyperplane in-between the two classes to identify points on either side of the hyperplane. Between the points and the hyperplane lies a margin. The objective was to maximize the margin so the model can predict the target classes for new data. The maximum margin area is important so the hyperplane can separate the two classes distinctly.

Since the project's dependent variable was either "fake" or "fact", SVM classified new data by putting it on the "fake" side or the "fact" side of the hyperplane. SVM was a good model to test in this project because it can quickly provide a high accuracy with less computation power.

This project assessed the following parameters for SVM: C, kernel, gamma. Different combinations of hyperparameters for SVM were assessed to achieve an optimal score. C is the regularization parameter, where the strength of the regularization is inversely proportional to C. The kernel parameter specified the kernel type. There are different kinds of kernels that correspond to variations of SVM, such as 'linear', 'poly', and 'rbf'. Gamma is the kernel coefficient, which can either be set as 'rbf', 'poly', and 'sigmoid.'

K-Nearest Neighbors (KNN)

K-nearest neighbors (KNN) is a non-parametric classification method that is used to analyze binary classification problems. Each data point is classified by the most common neighbors (k). The assigned value of k is the number of nearest neighbors that classifies each data point. KNN relies on the distance between each point for classification to calculate the nearest neighbor. So, for this project, KNN would determine if each data point would be classified as either fake or fact based on the proximity of its neighbors.

This project assessed the following parameters for KNN: n-neighbors, weights, and metric. Different combinations of hyperparameters for KNN will be assessed to achieve an optimal score. N-neighbors were the number of neighbors set for KNN. Weights were the weight function used in predicting each data point, such as 'uniform' and 'distance'. Metric was the distance metric type to set for KNN, such as 'minkowski' or 'euclidean'.

Voting Ensemble

Voting ensemble is a type of ensemble method where multiple algorithms create a voting system to improve predictive performance. For this project, the algorithms included in the voting ensemble were: logistic regression, SVM, and KNN. Each algorithm would vote if a news article is either fake or fact, and the majority would prevail. The project parameters for voting ensemble will be the default parameters in Python's Sklearn for hard voting. Hard voting predicts class labels for majority rule voting. Since a voting ensemble method utilizes multiple algorithms, it will also use the same hyperparameters that were tuned for each algorithm.

Parameter and Hyperparameter Tuning

Parameter tuning was performed for each machine learning method to find the optimal hyperparameters to maximize our metrics. Different combinations of hyperparameters were tested to see how our metrics changed. GridSearchCV is a function in the Sklearn library in Python that makes parameter tuning more efficient. It fine-tuned the parameters of the model by automatically trying different combinations of hyperparameters the user wanted it to try. Then, it showed the best combination of hyperparameters that produced the highest accuracy.

Performance Metrics

The performance of the model was evaluated by the following key metrics: accuracy, recall, precision, F1-score, and AUC-ROC curves. Most of these metrics were based on the confusion matrix, which was a performance measurement for machine learning classification. A confusion matrix consists of four parameters: true positive (TP), false positive (FP), false negative (FN), and true negative (TN).

Accuracy

Accuracy represents the total percentage of correctly predicted observations. For this model, accuracy tells us how many fake news articles the model predicted correctly. Accuracy is calculated by:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2}$$

Recall

Recall represents the ratio of total number of positive classifications out of the total number of fake news articles. The closer the value is to 1, the better the recall. For this model, recall tells us how many fake news articles the model predicted correctly in the actual results. Recall is calculated by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

Precision

Precision represents the ratio of true positive predictions out of the total number of positive predictions. The closer the value is to 1, the better the precision. For this model, precision tells us how many fake news articles the model predicted correctly from the predicted results. Precision is calculated by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

F1-score

F1-score represents the harmonic mean of both recall and precision. It calculates the average of recall and precision that weights both metrics in a balanced way. The closer the value is to 1, the better the F1-score. For this model, the F1-score tells us how balanced our dataset is. A high F1-score shows that there was low bias in determining whether a news article was fake or fact.

$$\mathrm{F1-score} = 2\frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{5}$$

AUC-ROC Curve

The Receiver Operator Characteristic (ROC) curve is a model performance metric that tells how well the model can distinguish between a binary output. The ROC curve is a visualization that graphs the false positive rate (specificity) on the x-axis and the true positive rate (sensitivity) on the y-axis. The area under the curve (AUC) quantifies the curve with a number between 0 to 1 to show how well the model can distinguish between a binary output. The closer the AUC value is to 1, the better the score. The AUC-ROC curve is an ideal metric for this project because it evaluates how well the model performs at identifying fake and fact news.

Feature Weights Using Bi-grams

To further understand the data, this project ranked the top 10 features that each model (Model 1 and 2) was interpreting as fake or fact. It was important to know what words the model is weighing more as fake or fact data. As the models were trained it picked up patterns of words and their relationship to the dependent variable (fake/fact news). To improve model processing time and scalability, the models used a natural language processing (NLP) technique called bi-grams.

Bi-grams are a two-word sequence from a sentence (Moawad et. al, 2018). For example, the sentence “I like machine learning,” would be transformed using bi-grams into “I like” “machine learning.” This was beneficial for our model because it simplified sentences by “bucketing” words together, so the model analyzed words in pairs, instead of individually. The model could store and understand more data because it can scale the large amount of text into smaller chunks.

For Model 1 and 2, logistic regression was used to explore the top 10 features. These features were ranked by how much the model weighed the terms to classify fake and fact news. For logistic regression, the weight is the regression coefficient, β_1. The logistic regression coefficient (β_1) associated with a predictor (X) is the expected change in log odds of having the outcome per unit change in X. Logistic regression can be modeled as:

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \tag{6}$$

Y_i is the dependent variable, β_0 is the population Y intercept, β_1 is the population slope coefficient, X_i is the independent variable, and ε_i is the random error term.

To interpret the regression coefficients, the odds ratio is calculated by:

$$\text{Odds Ratio} = e^{\beta 1} \tag{7}$$

The regression coefficient was calculated for each bi-gram in Model 1 and 2 using the sklearn package, linear_model.LogisticRegression. A positive coefficient indicates that it is associated with fake news, and a negative coefficient indicates that it is associated with fact news. The top 10 positive coefficients were ranked as the top 10 bi-grams that were associated with fake news. The top 10 negative coefficients were ranked as the top 10 bi-grams that were associated with fact news.

Chapter 4: Results & Analysis

Model 1 Bi-gram Results

Figure 2 shows the top 10 bi-grams related to fake news from Model 1. “world war”, “despite report”, and “brad pitt” top the list as most weighted bi-grams associated with fake news. This represents what terms the model is weighing more/less to help classify fake news. For example, the feature, “world war”, has a weight of 3.0. Using the odds ratio formula (7), the odds ratio for “world war” is 20.1. This can be interpreted as: The odds of the term “world war” being associated with fake news is 20.1 times more likely (or 1,909% higher) than any other term in the dataset.

Figure 2 reveals that Model 1’s fake news data is largely a mix of political and entertainment news. Since Model 1’s fake news are mainly political and entertainment news related, it gives insight that the model will likely perform better with political and entertainment news. “brad pitt” are “celebr react” are related to entertainment news and are ranked 3rd and 5th. “hillari clinton”, “hillari email”, and “us elect” are related to political news and are ranked 7th, 9th, and 10th, respectively.

Figure 3 shows the top 10 bi-grams related to fact news from Model 1. “york time”, “new york”, and “red carpet” top the list as most weighted bi-grams associated with fact news. For example, the feature, “york time”, has a weight of -10.2. Using the odds ratio formula (7), the odds ratio for “york time” is 3.7×10^{-5}. This can be interpreted as: The odds of the term “york time” being associated with fake news is 100% times less likely than any other term in the dataset.

Figure 2

Top 10 fake features and their corresponding weights in Model 1.

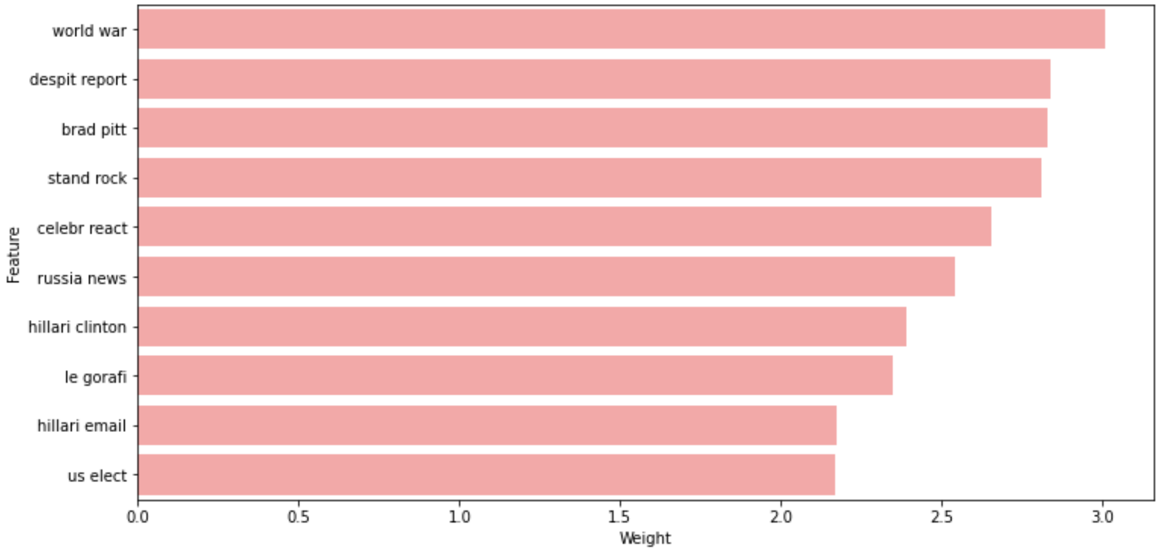


Figure 3

Top 10 fact features and their corresponding weights in Model 1.

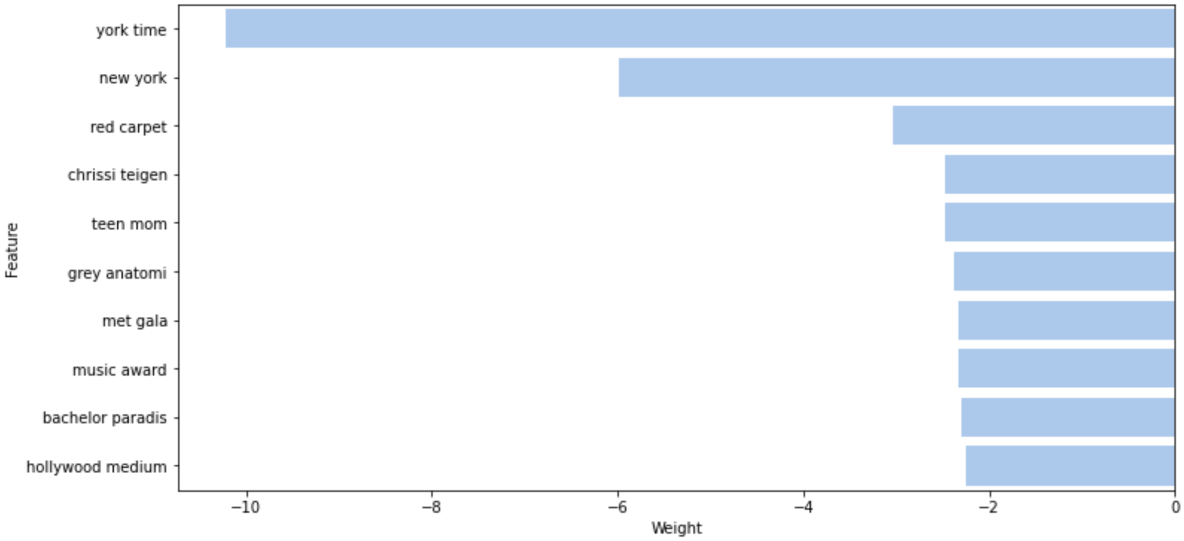


Figure 3 reveals that Model 1's fact news data is largely entertainment news. Since Model 1's fact news is mainly entertainment news related, it gives insight that the model will likely perform better with entertainment news. "red carpet", "chrissi teigen", "teen mom", "grey anatomi", "music award", "bachelor paradis", and "hollywood medium" are related to entertainment news and are ranked 3rd, 4th, 5th, 6th, 8th, 9th, and 10th respectively.

Model 2 Bi-gram Results

Figure 4 shows the top 10 bi-grams related to fake news from Model 2. "hilari clinton", "year old", and "file photo" top the list as most weighted bi-grams associated with fake news. This represents what terms the model is weighing more/less to help classify fake news. For example, the feature, "hilari clinton", has a weight of 5.2. Using the odds ratio formula (7), the odds ratio for "hilari clinton" is 181.3. This can be interpreted as: The odds of the term "hilari clinton" being associated with fake news is 181.3 times more likely (or 18,027% higher) than any other term in the dataset.

Figure 4 reveals that Model 2's fake news data is largely political news. Since Model 2's fake news is mainly political news related, it gives insight that the model will likely perform better with political news. "hillari clinton" and "presid elect", and "us elect" are related to political news and ranked 1st and 5th, respectively. "york time" and "new york" are closely related to the New York Times. Since they are the top 2 ranked fact features in Model 2, this means that articles related to the New York Times are most likely to be fact.

Figure 5 shows the top 10 bi-grams related to fact news from Model 2. "york time", "new york", and "sourc http" top the list as most weighted bi-grams associated with fact news. This represents what terms the model is weighing more/less to help classify fact news. For example, the feature, “york time”, has a weight of -12.3. Using the odds ratio formula (7), the odds ratio for “york time” is 4.5 x 10^{-6}. This can be interpreted as: The odds of the term “york time” being associated with fake news is 100% times less likely than any other term in the dataset.

Figure 5 reveals that Model 2’s fact news data is mainly Donald Trump related. “presid donald”, “mr trump”, and “presid trump” are related to Donald Trump and are ranked 5th, 6th, 8th, respectively. “york time” and “new york” are closely related to the New York Times. Since they are the top 2 ranked fact features in Model 2, this means that articles related to the New York Times are most likely to be fact.

Figure 4

Top 10 fake features and their corresponding weights in Model 2.

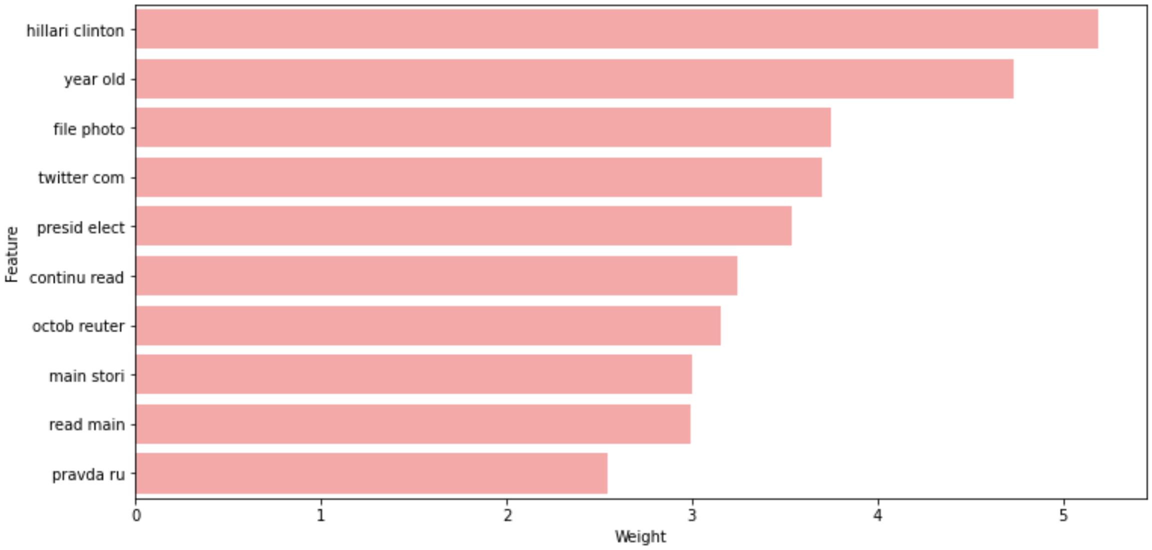
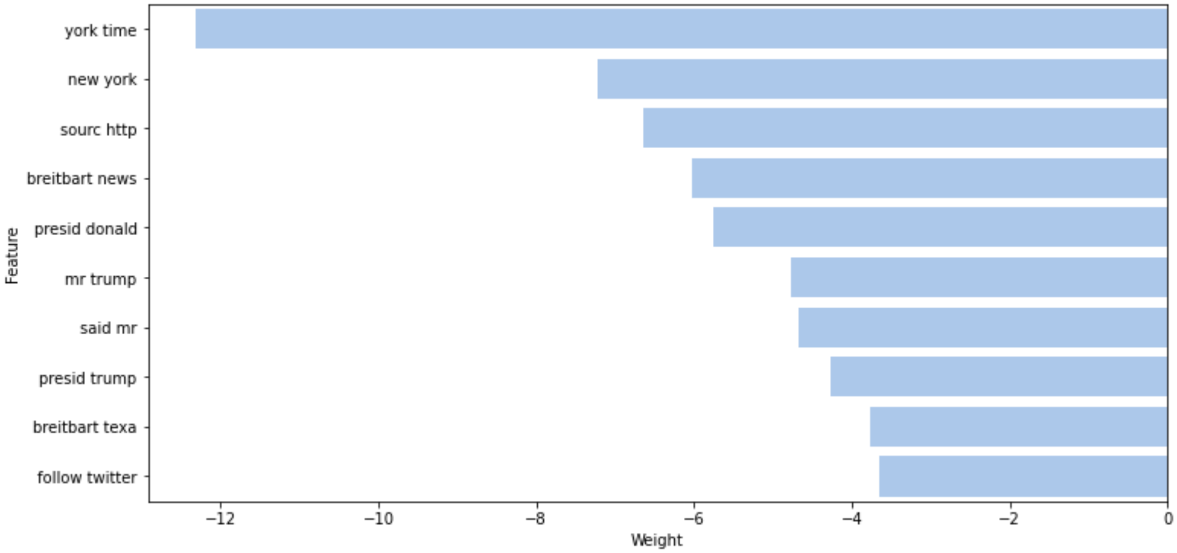


Figure 5

Top 10 fact features and their corresponding weights in Model 2.

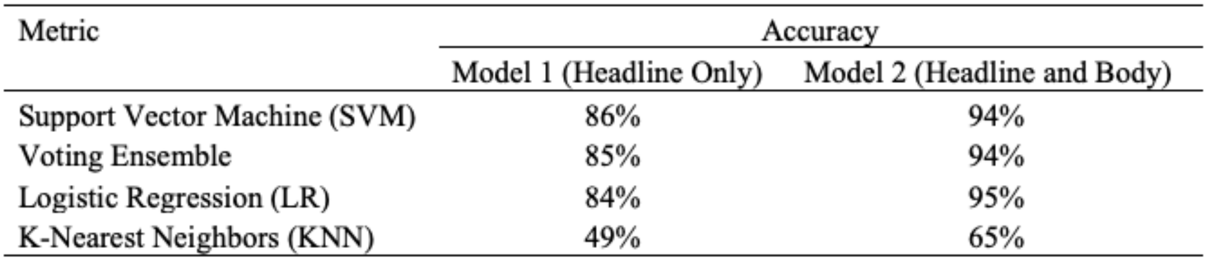


Accuracy Results

Table 2 displays the results for accuracy of our support vector machine, voting ensemble, logistic regression, and k-nearest neighbor algorithms for both chosen models. For Model 1, support vector machine achieved the highest accuracy with 86%. Voting ensemble achieved 85% accuracy for Model 1, while logistic regression achieved 84%. For Model 1, k-nearest neighbors performed the poorest with 49% accuracy. Model 2 performed better than Model 1, with the only difference being that Model 2 included the body of the article as an input, whereas Model 1 included only the headline. For Model 1, logistic regression ranked the third best performer, while logistic regression ranked first for Model 2 at 95% accuracy. Support vector machine and voting ensemble both achieved 94% accuracy for Model 2. Similar to Model 1, k-nearest neighbors was the worst performer for Model 2 at 65% accuracy.

Table 2

Accuracy percentages for each algorithm between Model 1 and Model 2.



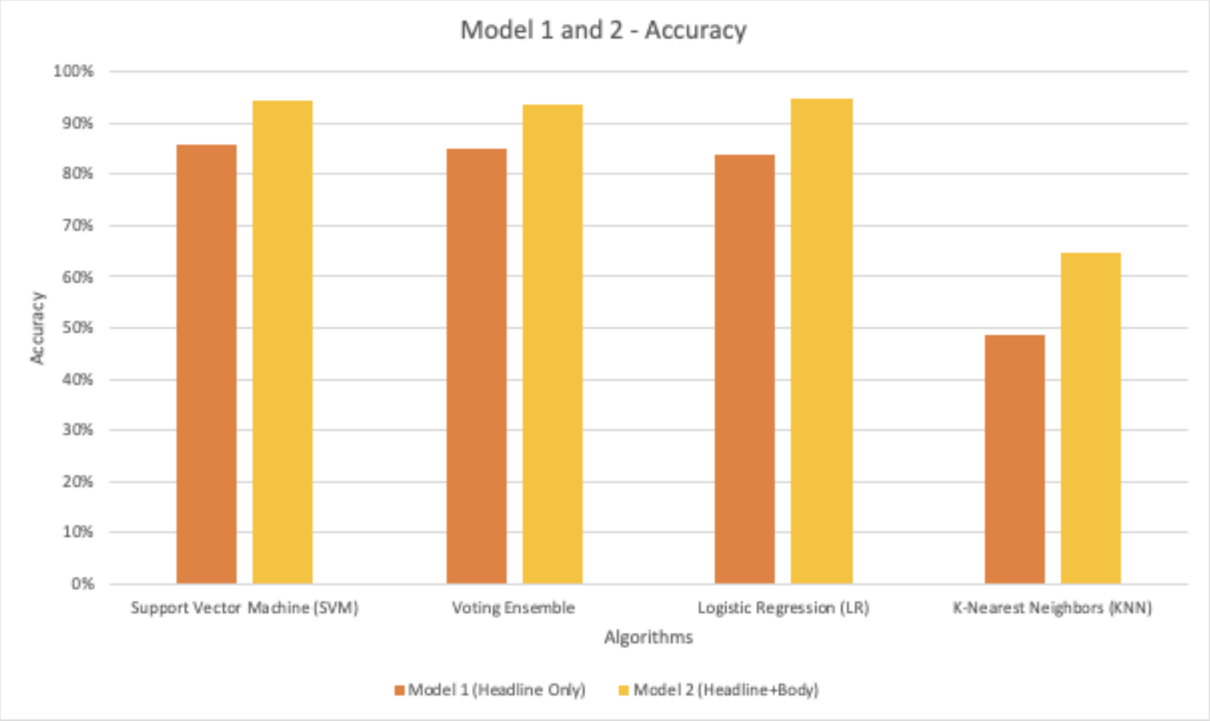
Metric	Accuracy	
	Model 1 (Headline Only)	Model 2 (Headline and Body)
Support Vector Machine (SVM)	86%	94%
Voting Ensemble	85%	94%
Logistic Regression (LR)	84%	95%
K-Nearest Neighbors (KNN)	49%	65%

The average accuracy for Model 1 was 76%, and the average accuracy for Model 2 was 87%. If the study removed the worst performer for both models, k-nearest neighbors, it was observed that the average accuracy for Model 1's algorithms to be 87% and the average for Model 2 to be 94%. Observing accuracy only, it was evident Model 2 performed better with the respective highest performing algorithm, with a 9% difference in accuracy. Figure 6 also depicts how the algorithms performed for both models. Again, this study showed that Model 2 clearly performed better for accuracy with each algorithm and that k-nearest neighbors performed the worst for both models.

These accuracy results demonstrate that SVM excels well when there is a clear margin of separation between the classes (fake/fact news). In the SVM model, the algorithm creates a hyperplane to learn and understand new data. This hyperplane distinctly classifies the data points, which also helps it understand data with high dimensionality. Our data had high dimensionality with over 10,000 unique words. This proves that SVM can handle high dimensional data and can generalize well on new data it hasn't seen. Due to KNN's low performance, it reduced the performance of the voting ensemble. If KNN performed better, we would have likely seen the voting ensemble algorithm perform the best.

Figure 6

Accuracy comparison between Model 1 and Model 2 for each algorithm.



Precision, Recall, F1-score Results

Table 3 displays the precision, recall, and F1-score for each algorithm and both models. For Model 1, looking at average precision, support vector machine performed the best at 85%. There was only a 1% difference in the average precision of the voting ensemble and logistic regression, with 84% and 83% precision, respectively. K-nearest neighbors had the lowest average precision for Model 1 at 63%. Model 2's average precision for support vector machine, voting ensemble, and logistic regression is 94%. Similar to Model 1, Model 2's precision using the k-nearest neighbors algorithm was the lowest at 73%. Figure 7 compares the precision, recall, and F1-score. It was clear that the precision for Model 2 was the highest.

Table 3

Precision, recall, and F1-score percentages for each algorithm between Model 1 and Model 2.

	Model 1 (Headline Only)			Model 2 (Headline and Body)		
Metric	Precision	Recall	F1-score	Precision	Recall	F1-score
Support Vector Machine (SVM)	85%	84%	84%	94%	94%	94%
Voting Ensemble	84%	84%	84%	94%	94%	94%
Logistic Regression (LR)	83%	82%	82%	94%	94%	94%
K-Nearest Neighbors (KNN)	63%	58%	47%	73%	66%	62%

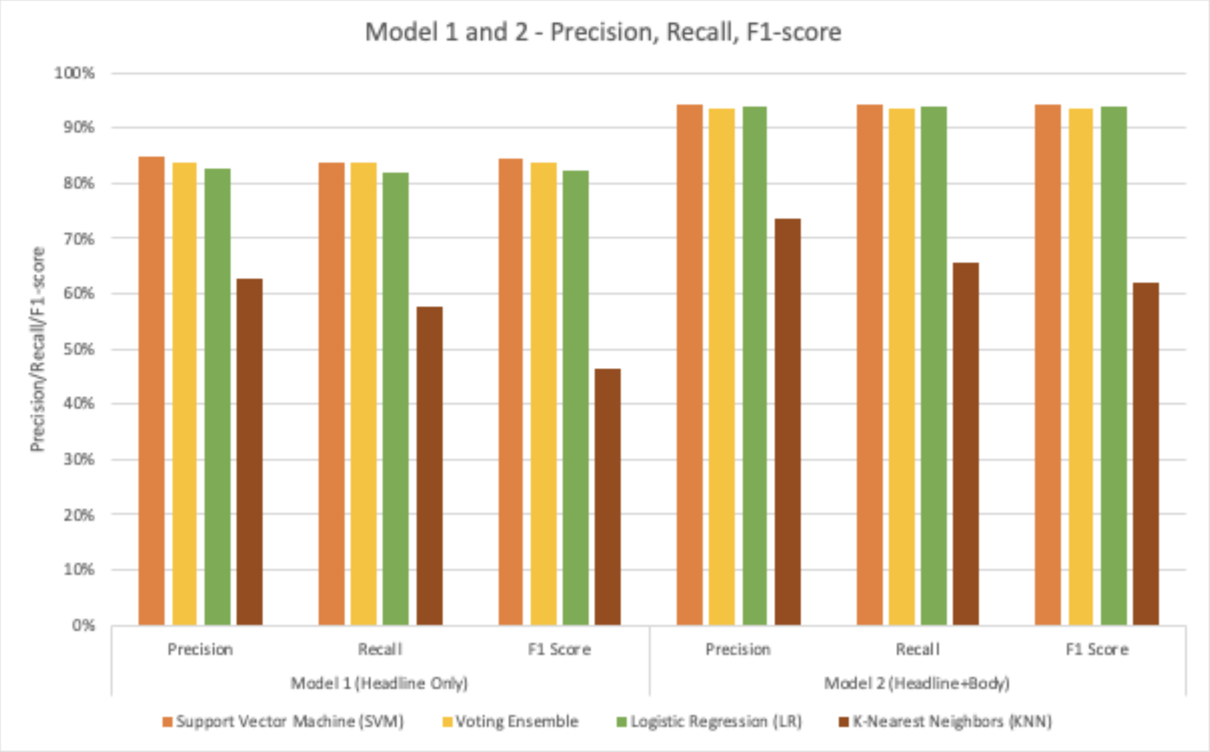
Referring to recall, the metrics showed that for Model 1, support vector machine and voting ensemble performed the highest at 84%. Logistic regression for Model 1 had an average recall of 82%, and k-nearest neighbors scored the lowest at 58%. Model 2's recall was higher than Model 1's by 10% on the best performing algorithm. The average recall was 94% for support vector machine, voting ensemble, and logistic regression. K-nearest neighbors had the lowest average recall at 66%. Figure 6 also shows that Model 2 performed better than Model 1 for recall.

In Model 1, the F1-score for support vector machine and voting ensemble algorithms were at 84%. Logistic regression for Model 1 and K-nearest neighbors was 82% and 47%, respectively. Again, referring to Figure 3, it was clear Model 2 performed better in regard to the F1-score. Support vector machine, voting ensemble, and logistic regression all had an F1-score of 94%. While Model 2's k-nearest neighbors F1-score was 15% higher for Model 2 than for

Model 1, 62% was still the lowest performance. The high scores for precision, recall, and F1-score showed that all the models, except KNN, had minimal instances where it falsely classified fake news and falsely classified fact news. For each model, SVM, voting ensemble, and logistic regression had similar percentages. This showed that there was low bias towards either fake or fact news in these models, and the lack of variance between precision and recall showed that the classes were balanced.

Figure 7

Precision, recall, and F1-score comparison between Model 1 and Model 2 for each algorithm.

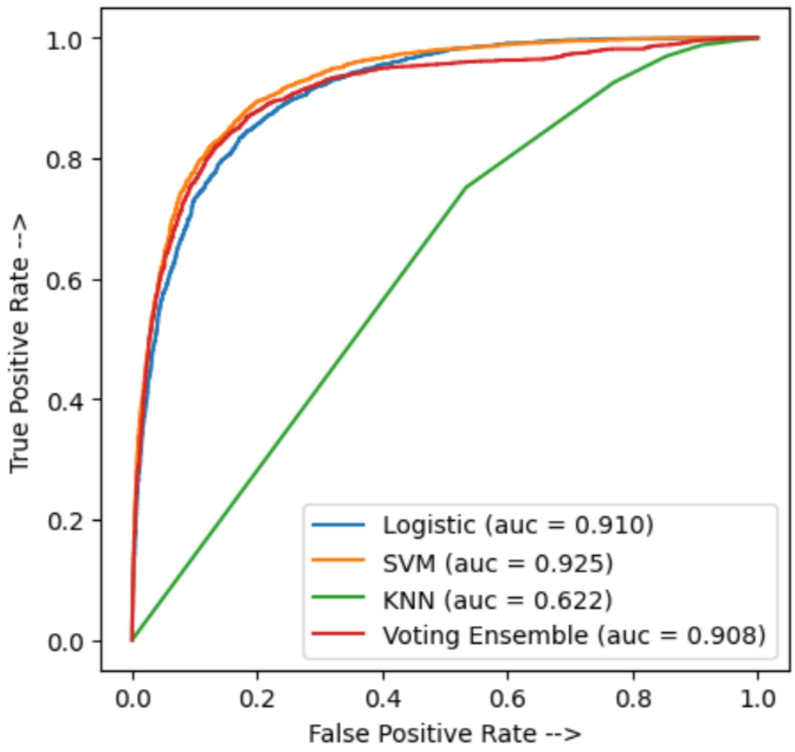


AUC-ROC Curve Results

Figure 8 shows the ROC Curve for Model 1. The k-nearest neighbors algorithm had the least area under the curve at 0.622. There was a slight difference between the other algorithms. Support vector machine scored the highest at 0.925. Logistic regression and the voting ensemble closely follow at 0.910 and 0.908, respectively.

Figure 8

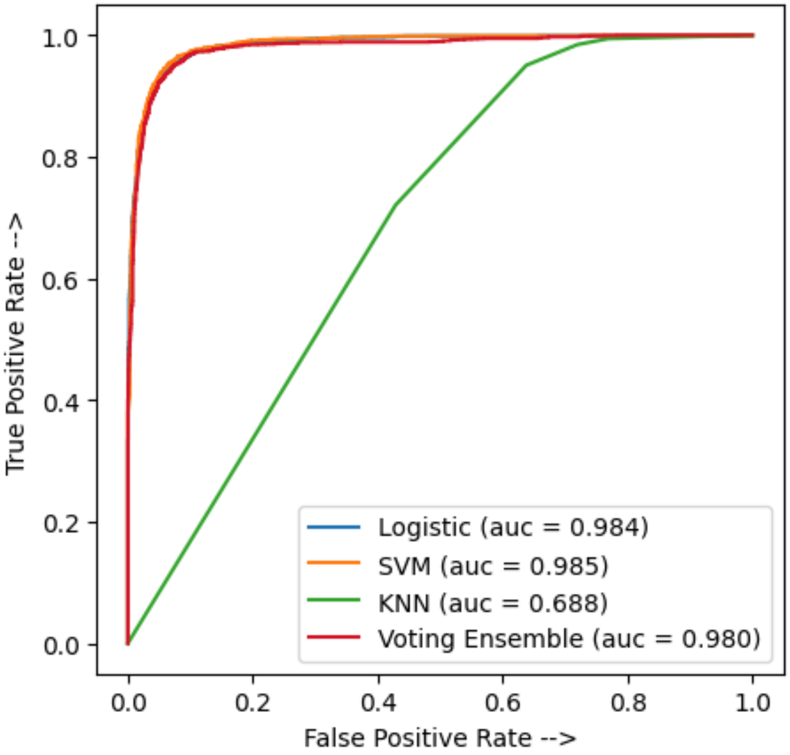
AUC-ROC curves for each algorithm in Model 1 (Headline Only).



Model 2's ROC Curve, shown in Figure 9, also showed k-nearest neighbors performing the worst, with the area under the curve being 0.688. The other three algorithms performed similarly well, with a variance of 0.001. Support vector machine's area was 0.985, logistic regression's area was 0.984, and the voting ensemble's area was 0.980. Models AUC-ROC results showed that SVM, voting ensemble, and logistic regression were high performing models at identifying fake news. For the best performing model, SVM, there was a 98% chance that Model 2 would correctly distinguish a fake article based on the news headline and body.

Figure 9

AUC-ROC curves for each algorithm in Model 2 (Headline and Body).



It was not a surprise to see that Model 2 performed better than Model 1. The reason made sense that the model with more textual input would be better at predicting the outcome. It was also understandable that the voting ensemble performed lower than the support vector machine and logistic regression for Model 1 because it included k-nearest neighbors, which performed poorly on both models.

Chapter 5: Conclusions and Recommendations

Today's technologies has made connecting, communicating, and spreading information easier over the years. However, it has also created an ideal environment for misinformation and is threatening a sense of truth. When COVID-19 vaccines were available, information was spreading on how they worked, what the side effects were, if they were safe, or even if they included a microchip. The entire world was talking about it, but people had different answers to the same questions.

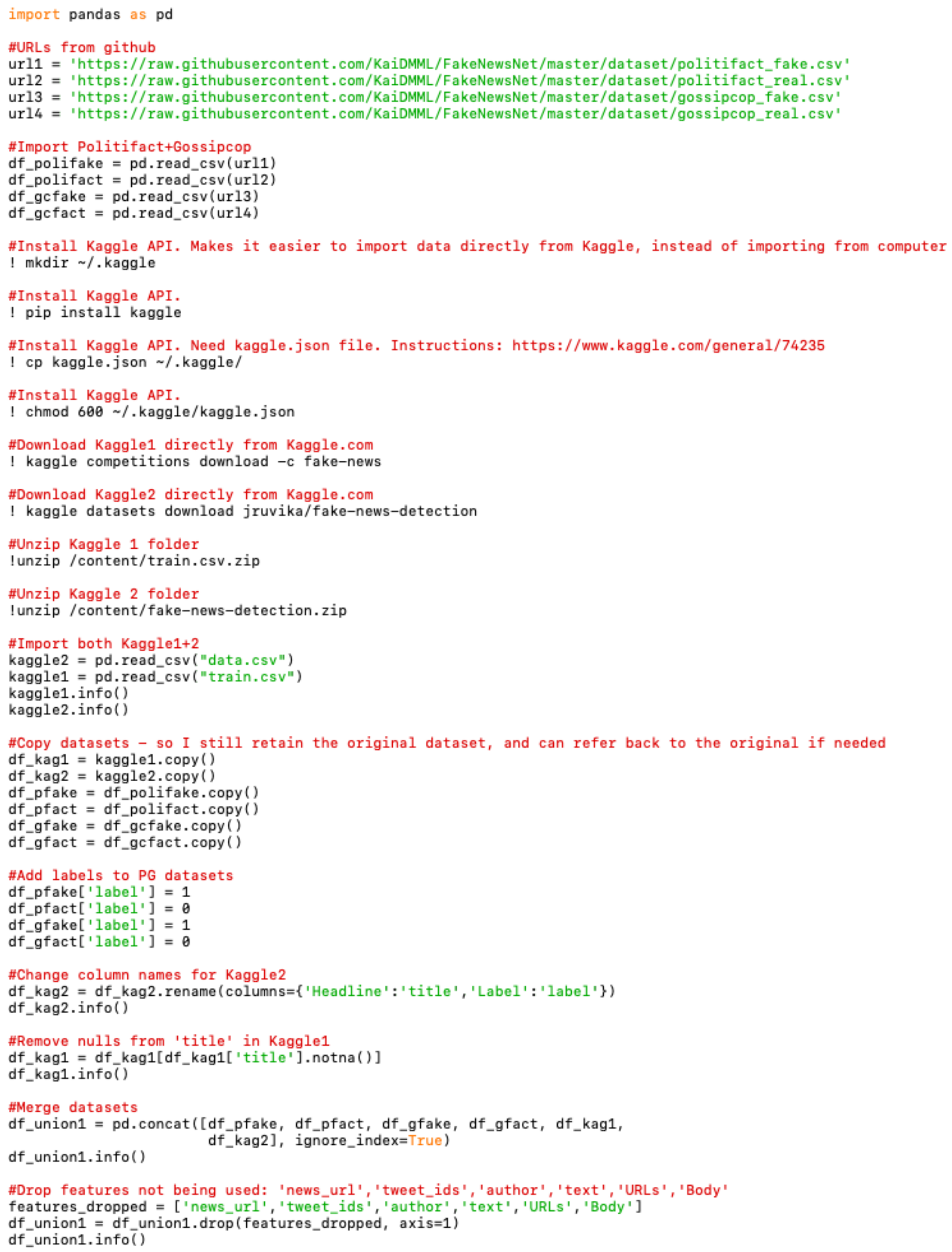
In this study, it was investigated how one can use machine learning methods to create high performing models to classify fake news articles. We compared two models: the first model had a news headline as the only feature (Model 1) and the second model had a news headline and the body of the article as features (Model 2). SVM was the best performing algorithm that classified fake news in Model 1 and Model 2 with an accuracy of 86% and 94%, respectively. Model 1 shows us that SVM can accurately determine if a news article is fake by only analyzing the headline — which in the data, is only one sentence. Model 2 shows that if we incorporate one more feature (body of the article), it could boost the model's accuracy from 86% to 94%.

One of the ways we plan on expanding this project in the future is to diversify the data. Based on our bigram figure (Figure 1), the models are picking up a lot of fake and fact political and celebrity gossip data. Since the models were trained using this data, models performed well when this type of data was presented to it. However, fake news that was not political or celebrity gossip related, may not be able to be classified correctly. So, a way to expand this project is to introduce other news categories, such as health, education, business, finance, and technology. Fake news is a broad term and our future work could dive into other avenues of news.

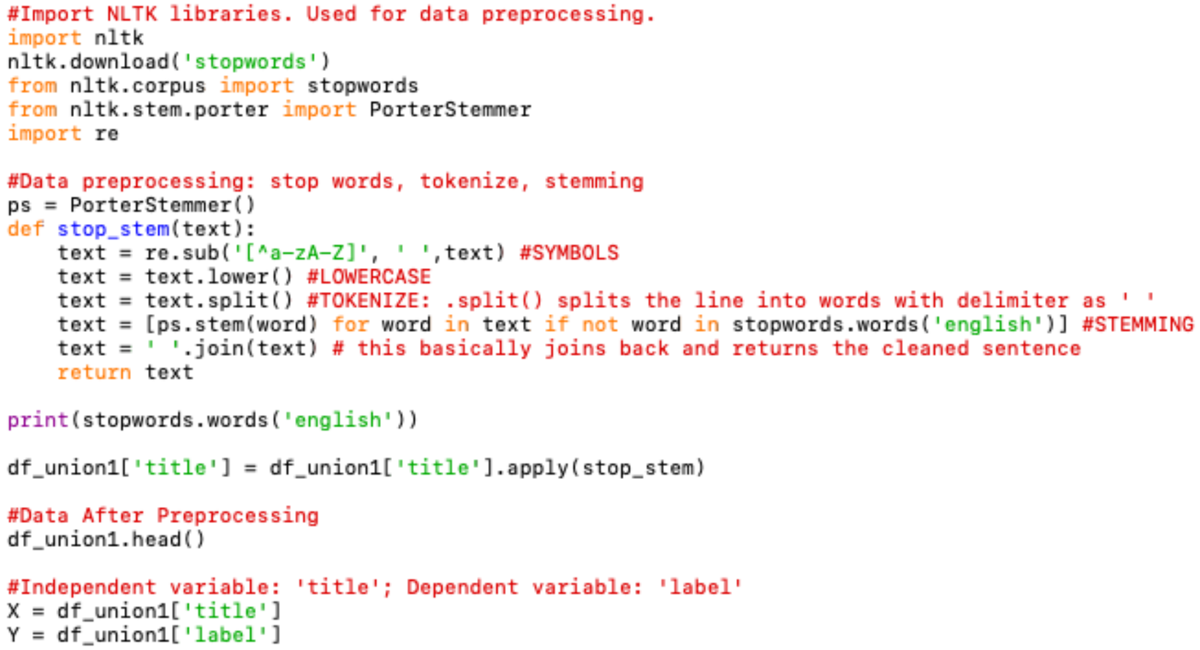
Another way to expand this study, is that different algorithms could be used to see if metrics change. In this study, it used logistic regression, SVM, k-nearest neighbors, and voting ensemble. However, there are so many kinds of algorithms that it would be interesting to see how other algorithms perform. Algorithms, such as Naive Bayes and random forest, also perform well with a binary classification problem like this study. Deep learning could also be another approach we can take, such as using multilayer perceptrons or a convolutional neural network. These other approaches might yield better results at identifying fake news and could be another way to expand this project.

The results from this project are important because it demonstrated how it can gather publicly available data to create high performing models to identify fake news. The project started this study to use machine learning methods to combat fake news, and wanted to show that fancy tools, or expensive equipment was not required ─ just a computer with publicly available software found online. We cannot escape accumulating information, so we might as well improve how we absorb the information. We hope that this project demonstrates and spreads awareness of how it can use current technology to find solutions to current problems.

Appendix A1: Python Code - Model 1 - Data Preparation/Data Preprocessing

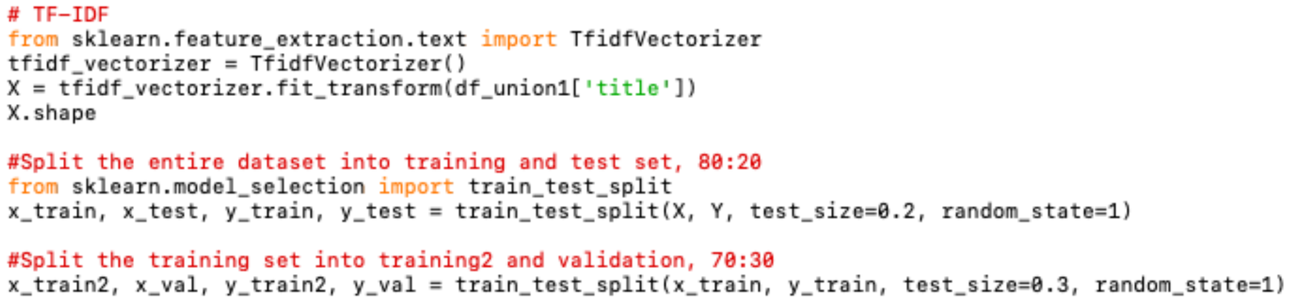


```
import pandas as pd

#URLs from github
url1 = 'https://raw.githubusercontent.com/KaiDMML/FakeNewsNet/master/dataset/politifact_fake.csv'
url2 = 'https://raw.githubusercontent.com/KaiDMML/FakeNewsNet/master/dataset/politifact_real.csv'
url3 = 'https://raw.githubusercontent.com/KaiDMML/FakeNewsNet/master/dataset/gossipcop_fake.csv'
url4 = 'https://raw.githubusercontent.com/KaiDMML/FakeNewsNet/master/dataset/gossipcop_real.csv'

#Import Politifact+Gossipcop
df_polifake = pd.read_csv(url1)
df_polifact = pd.read_csv(url2)
df_gcfake = pd.read_csv(url3)
df_gcfact = pd.read_csv(url4)

#Install Kaggle API. Makes it easier to import data directly from Kaggle, instead of importing from computer
! mkdir ~/.kaggle

#Install Kaggle API.
! pip install kaggle

#Install Kaggle API. Need kaggle.json file. Instructions: https://www.kaggle.com/general/74235
! cp kaggle.json ~/.kaggle/

#Install Kaggle API.
! chmod 600 ~/.kaggle/kaggle.json

#Download Kaggle1 directly from Kaggle.com
! kaggle competitions download -c fake-news

#Download Kaggle2 directly from Kaggle.com
! kaggle datasets download jruvika/fake-news-detection

#Unzip Kaggle 1 folder
!unzip /content/train.csv.zip

#Unzip Kaggle 2 folder
!unzip /content/fake-news-detection.zip

#Import both Kaggle1+2
kaggle2 = pd.read_csv("data.csv")
kaggle1 = pd.read_csv("train.csv")
kaggle1.info()
kaggle2.info()

#Copy datasets - so I still retain the original dataset, and can refer back to the original if needed
df_kag1 = kaggle1.copy()
df_kag2 = kaggle2.copy()
df_pfake = df_polifake.copy()
df_pfact = df_polifact.copy()
df_gfake = df_gcfake.copy()
df_gfact = df_gcfact.copy()

#Add labels to PG datasets
df_pfake['label'] = 1
df_pfact['label'] = 0
df_gfake['label'] = 1
df_gfact['label'] = 0

#Change column names for Kaggle2
df_kag2 = df_kag2.rename(columns={'Headline':'title','Label':'label'})
df_kag2.info()

#Remove nulls from 'title' in Kaggle1
df_kag1 = df_kag1[df_kag1['title'].notna()]
df_kag1.info()

#Merge datasets
df_union1 = pd.concat([df_pfake, df_pfact, df_gfake, df_gfact, df_kag1,
                       df_kag2], ignore_index=True)
df_union1.info()

#Drop features not being used: 'news_url','tweet_ids','author','text','URLs','Body'
features_dropped = ['news_url','tweet_ids','author','text','URLs','Body']
df_union1 = df_union1.drop(features_dropped, axis=1)
df_union1.info()
```



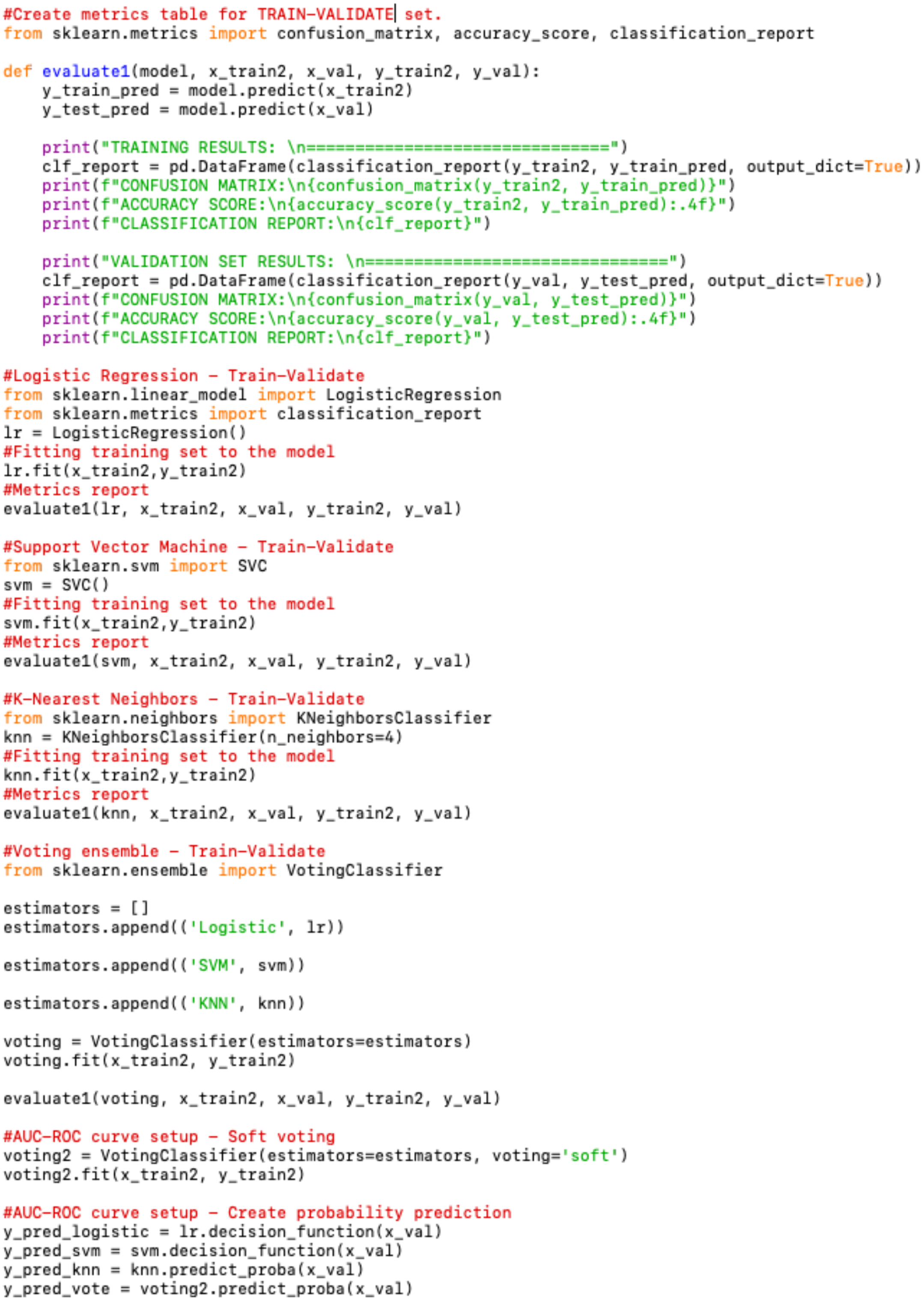
```
#Import NLTK libraries. Used for data preprocessing.
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import re

#Data preprocessing: stop words, tokenize, stemming
ps = PorterStemmer()
def stop_stem(text):
    text = re.sub('[^a-zA-Z]', ' ',text) #SYMBOLS
    text = text.lower() #LOWERCASE
    text = text.split() #TOKENIZE: .split() splits the line into words with delimiter as ' '
    text = [ps.stem(word) for word in text if not word in stopwords.words('english')] #STEMMING
    text = ' '.join(text) # this basically joins back and returns the cleaned sentence
    return text

print(stopwords.words('english'))

df_union1['title'] = df_union1['title'].apply(stop_stem)

#Data After Preprocessing
df_union1.head()

#Independent variable: 'title'; Dependent variable: 'label'
X = df_union1['title']
Y = df_union1['label']
```

Appendix A2: Python Code - Model 1 - Data Vectorization/Train-Validate-Test Split

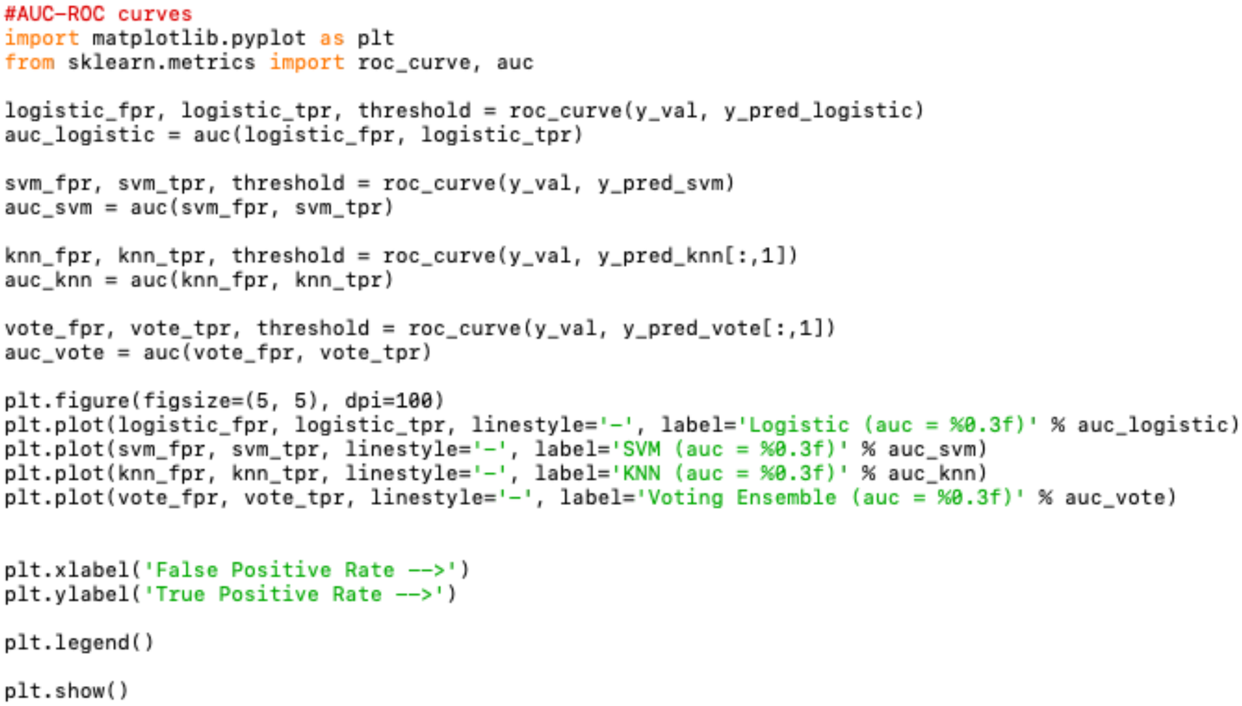


```
# TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(df_union1['title'])
X.shape

#Split the entire dataset into training and test set, 80:20
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)

#Split the training set into training2 and validation, 70:30
x_train2, x_val, y_train2, y_val = train_test_split(x_train, y_train, test_size=0.3, random_state=1)
```

Appendix A3: Python Code - Model 1 - Model Training

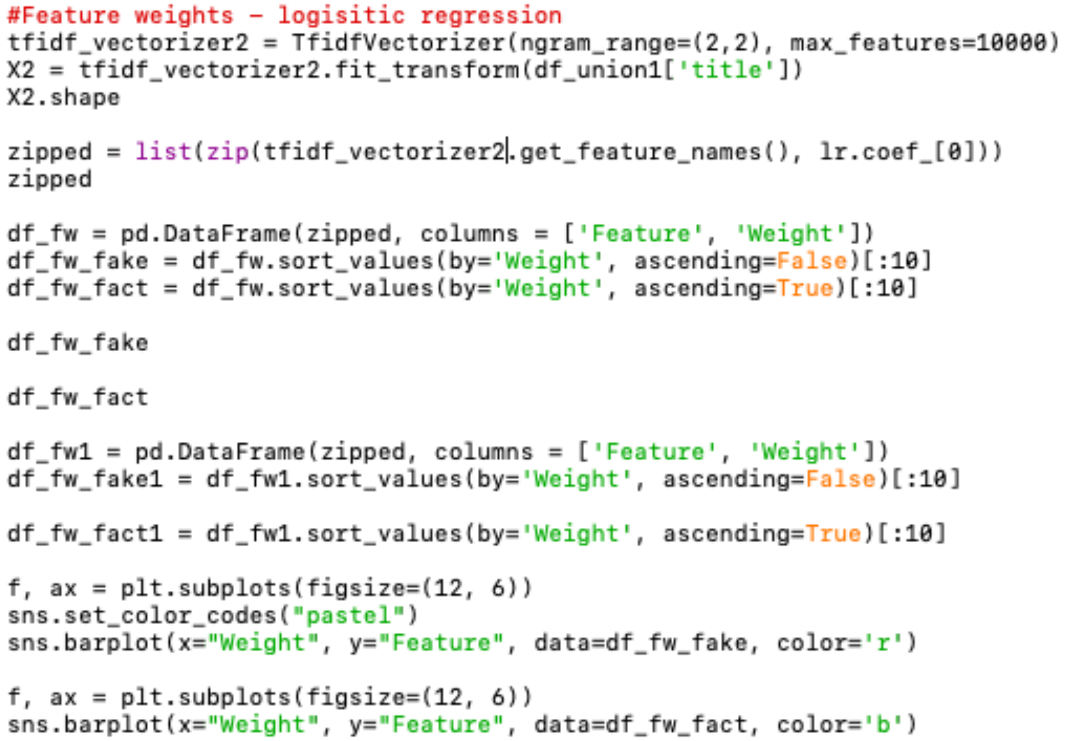


```
#Create metrics table for TRAIN-VALIDATE set.
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

def evaluate1(model, x_train2, x_val, y_train2, y_val):
    y_train_pred = model.predict(x_train2)
    y_test_pred = model.predict(x_val)

    print("TRAINING RESULTS: \n================================")
    clf_report = pd.DataFrame(classification_report(y_train2, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train2, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train2, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("VALIDATION SET RESULTS: \n================================")
    clf_report = pd.DataFrame(classification_report(y_val, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_val, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_val, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

#Logistic Regression - Train-Validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
lr = LogisticRegression()
#Fitting training set to the model
lr.fit(x_train2,y_train2)
#Metrics report
evaluate1(lr, x_train2, x_val, y_train2, y_val)

#Support Vector Machine - Train-Validate
from sklearn.svm import SVC
svm = SVC()
#Fitting training set to the model
svm.fit(x_train2,y_train2)
#Metrics report
evaluate1(svm, x_train2, x_val, y_train2, y_val)

#K-Nearest Neighbors - Train-Validate
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
#Fitting training set to the model
knn.fit(x_train2,y_train2)
#Metrics report
evaluate1(knn, x_train2, x_val, y_train2, y_val)

#Voting ensemble - Train-Validate
from sklearn.ensemble import VotingClassifier

estimators = []
estimators.append(('Logistic', lr))

estimators.append(('SVM', svm))

estimators.append(('KNN', knn))

voting = VotingClassifier(estimators=estimators)
voting.fit(x_train2, y_train2)

evaluate1(voting, x_train2, x_val, y_train2, y_val)

#AUC-ROC curve setup - Soft voting
voting2 = VotingClassifier(estimators=estimators, voting='soft')
voting2.fit(x_train2, y_train2)

#AUC-ROC curve setup - Create probability prediction
y_pred_logistic = lr.decision_function(x_val)
y_pred_svm = svm.decision_function(x_val)
y_pred_knn = knn.predict_proba(x_val)
y_pred_vote = voting2.predict_proba(x_val)
```



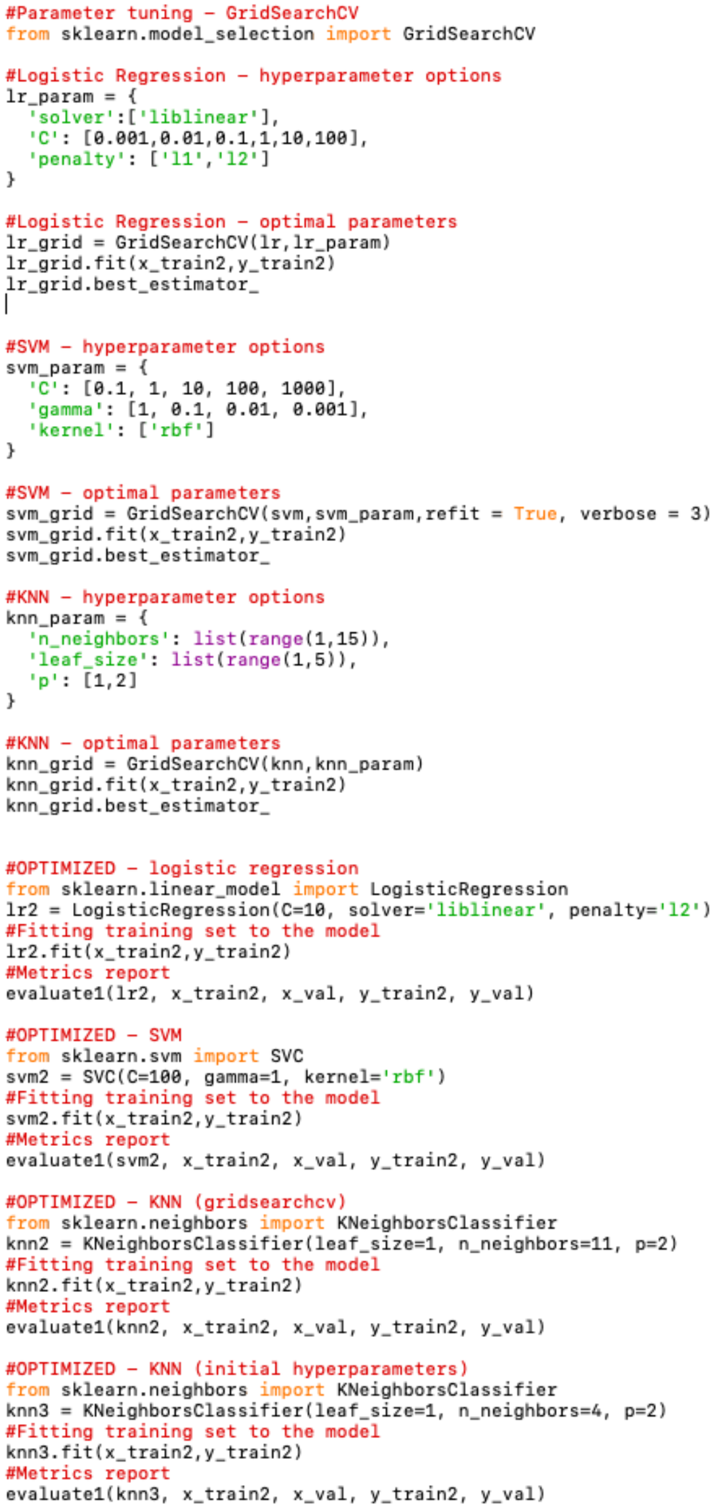
```
#AUC-ROC curves
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

logistic_fpr, logistic_tpr, threshold = roc_curve(y_val, y_pred_logistic)
auc_logistic = auc(logistic_fpr, logistic_tpr)

svm_fpr, svm_tpr, threshold = roc_curve(y_val, y_pred_svm)
auc_svm = auc(svm_fpr, svm_tpr)

knn_fpr, knn_tpr, threshold = roc_curve(y_val, y_pred_knn[:,1])
auc_knn = auc(knn_fpr, knn_tpr)

vote_fpr, vote_tpr, threshold = roc_curve(y_val, y_pred_vote[:,1])
auc_vote = auc(vote_fpr, vote_tpr)

plt.figure(figsize=(5, 5), dpi=100)
plt.plot(logistic_fpr, logistic_tpr, linestyle='-', label='Logistic (auc = %0.3f)' % auc_logistic)
plt.plot(svm_fpr, svm_tpr, linestyle='-', label='SVM (auc = %0.3f)' % auc_svm)
plt.plot(knn_fpr, knn_tpr, linestyle='-', label='KNN (auc = %0.3f)' % auc_knn)
plt.plot(vote_fpr, vote_tpr, linestyle='-', label='Voting Ensemble (auc = %0.3f)' % auc_vote)


plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')

plt.legend()

plt.show()
```

Appendix A4: Python Code - Model 1 - Feature Weights

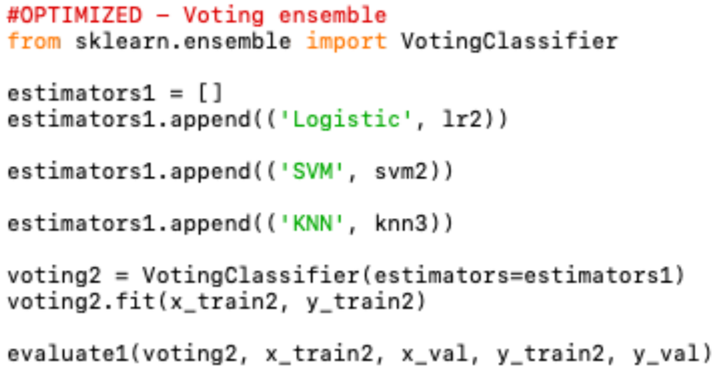


```
#Feature weights - logisitic regression
tfidf_vectorizer2 = TfidfVectorizer(ngram_range=(2,2), max_features=10000)
X2 = tfidf_vectorizer2.fit_transform(df_union1['title'])
X2.shape

zipped = list(zip(tfidf_vectorizer2.get_feature_names(), lr.coef_[0]))
zipped

df_fw = pd.DataFrame(zipped, columns = ['Feature', 'Weight'])
df_fw_fake = df_fw.sort_values(by='Weight', ascending=False)[:10]
df_fw_fact = df_fw.sort_values(by='Weight', ascending=True)[:10]

df_fw_fake

df_fw_fact

df_fw1 = pd.DataFrame(zipped, columns = ['Feature', 'Weight'])
df_fw_fake1 = df_fw1.sort_values(by='Weight', ascending=False)[:10]

df_fw_fact1 = df_fw1.sort_values(by='Weight', ascending=True)[:10]

f, ax = plt.subplots(figsize=(12, 6))
sns.set_color_codes("pastel")
sns.barplot(x="Weight", y="Feature", data=df_fw_fake, color='r')

f, ax = plt.subplots(figsize=(12, 6))
sns.barplot(x="Weight", y="Feature", data=df_fw_fact, color='b')
```

Appendix A5: Python Code - Model 1 - Model Hypertuning

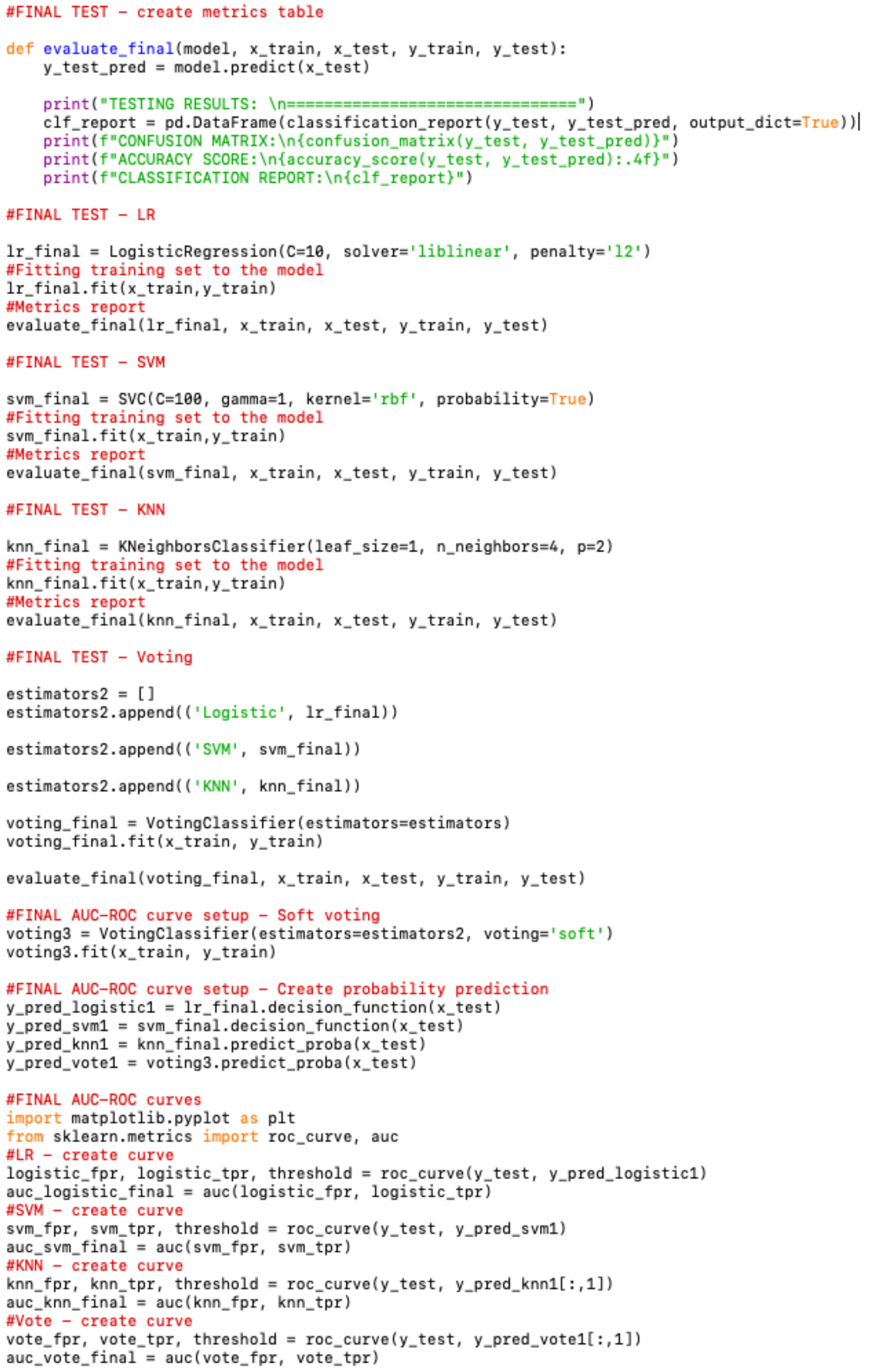


```
#Parameter tuning - GridSearchCV
from sklearn.model_selection import GridSearchCV

#Logistic Regression - hyperparameter options
lr_param = {
  'solver':['liblinear'],
  'C': [0.001,0.01,0.1,1,10,100],
  'penalty': ['l1','l2']
}

#Logistic Regression - optimal parameters
lr_grid = GridSearchCV(lr,lr_param)
lr_grid.fit(x_train2,y_train2)
lr_grid.best_estimator_


#SVM - hyperparameter options
svm_param = {
  'C': [0.1, 1, 10, 100, 1000],
  'gamma': [1, 0.1, 0.01, 0.001],
  'kernel': ['rbf']
}

#SVM - optimal parameters
svm_grid = GridSearchCV(svm,svm_param,refit = True, verbose = 3)
svm_grid.fit(x_train2,y_train2)
svm_grid.best_estimator_

#KNN - hyperparameter options
knn_param = {
  'n_neighbors': list(range(1,15)),
  'leaf_size': list(range(1,5)),
  'p': [1,2]
}

#KNN - optimal parameters
knn_grid = GridSearchCV(knn,knn_param)
knn_grid.fit(x_train2,y_train2)
knn_grid.best_estimator_


#OPTIMIZED - logistic regression
from sklearn.linear_model import LogisticRegression
lr2 = LogisticRegression(C=10, solver='liblinear', penalty='l2')
#Fitting training set to the model
lr2.fit(x_train2,y_train2)
#Metrics report
evaluate1(lr2, x_train2, x_val, y_train2, y_val)

#OPTIMIZED - SVM
from sklearn.svm import SVC
svm2 = SVC(C=100, gamma=1, kernel='rbf')
#Fitting training set to the model
svm2.fit(x_train2,y_train2)
#Metrics report
evaluate1(svm2, x_train2, x_val, y_train2, y_val)

#OPTIMIZED - KNN (gridsearchcv)
from sklearn.neighbors import KNeighborsClassifier
knn2 = KNeighborsClassifier(leaf_size=1, n_neighbors=11, p=2)
#Fitting training set to the model
knn2.fit(x_train2,y_train2)
#Metrics report
evaluate1(knn2, x_train2, x_val, y_train2, y_val)

#OPTIMIZED - KNN (initial hyperparameters)
from sklearn.neighbors import KNeighborsClassifier
knn3 = KNeighborsClassifier(leaf_size=1, n_neighbors=4, p=2)
#Fitting training set to the model
knn3.fit(x_train2,y_train2)
#Metrics report
evaluate1(knn3, x_train2, x_val, y_train2, y_val)
```



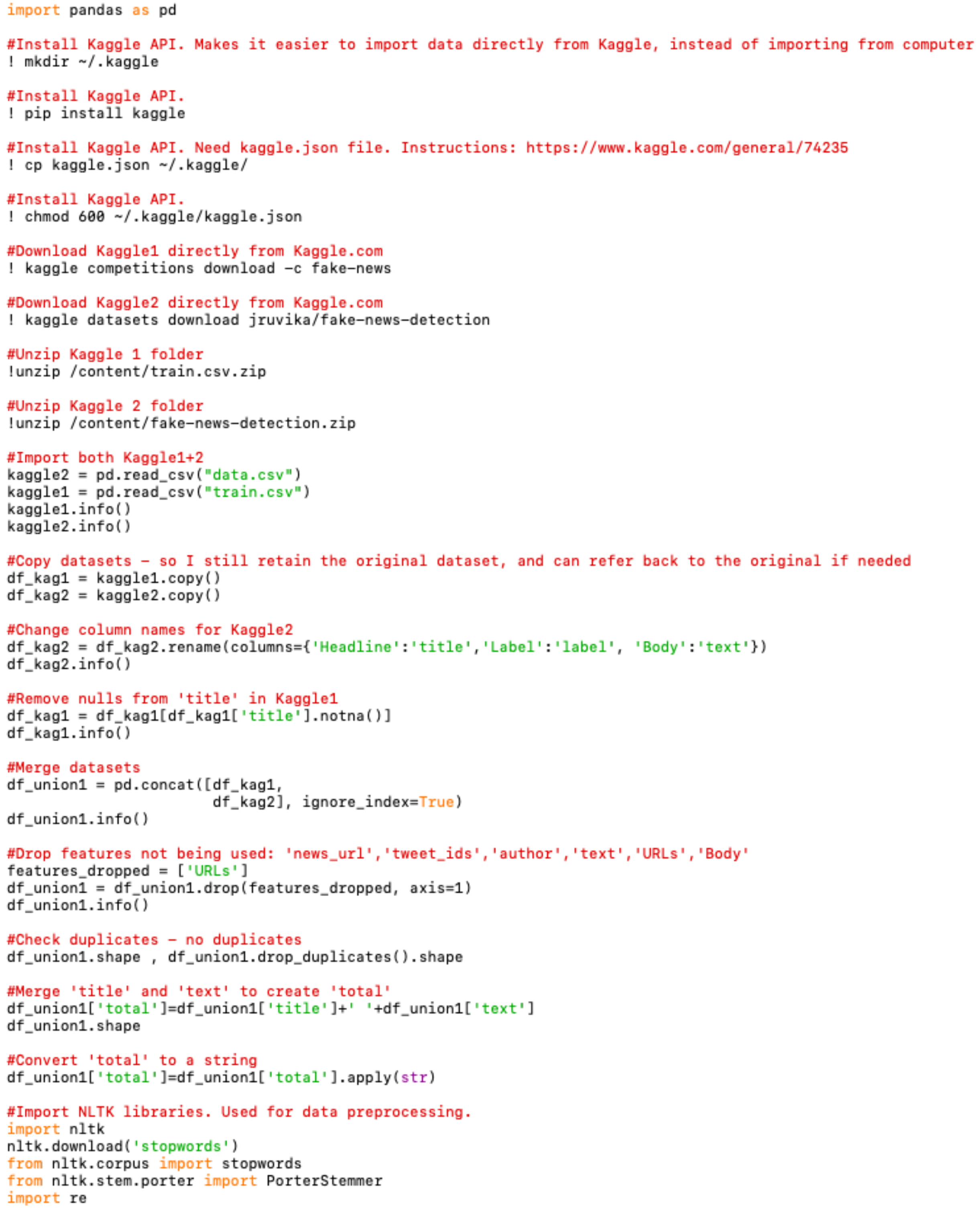
```
#OPTIMIZED - Voting ensemble
from sklearn.ensemble import VotingClassifier

estimators1 = []
estimators1.append(('Logistic', lr2))

estimators1.append(('SVM', svm2))

estimators1.append(('KNN', knn3))

voting2 = VotingClassifier(estimators=estimators1)
voting2.fit(x_train2, y_train2)

evaluate1(voting2, x_train2, x_val, y_train2, y_val)
```

Appendix A6: Python Code - Model 1 - Model Testing

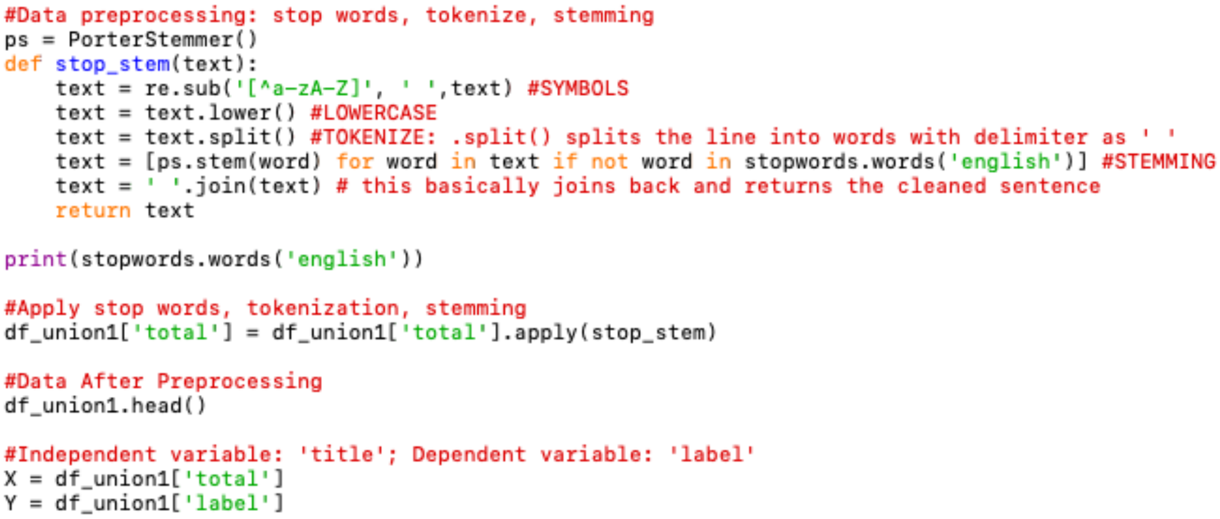


```
#FINAL TEST - create metrics table

def evaluate_final(model, x_train, x_test, y_train, y_test):
    y_test_pred = model.predict(x_test)

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

#FINAL TEST - LR

lr_final = LogisticRegression(C=10, solver='liblinear', penalty='l2')
#Fitting training set to the model
lr_final.fit(x_train,y_train)
#Metrics report
evaluate_final(lr_final, x_train, x_test, y_train, y_test)

#FINAL TEST - SVM

svm_final = SVC(C=100, gamma=1, kernel='rbf', probability=True)
#Fitting training set to the model
svm_final.fit(x_train,y_train)
#Metrics report
evaluate_final(svm_final, x_train, x_test, y_train, y_test)

#FINAL TEST - KNN

knn_final = KNeighborsClassifier(leaf_size=1, n_neighbors=4, p=2)
#Fitting training set to the model
knn_final.fit(x_train,y_train)
#Metrics report
evaluate_final(knn_final, x_train, x_test, y_train, y_test)

#FINAL TEST - Voting

estimators2 = []
estimators2.append(('Logistic', lr_final))

estimators2.append(('SVM', svm_final))

estimators2.append(('KNN', knn_final))

voting_final = VotingClassifier(estimators=estimators)
voting_final.fit(x_train, y_train)

evaluate_final(voting_final, x_train, x_test, y_train, y_test)

#FINAL AUC-ROC curve setup - Soft voting
voting3 = VotingClassifier(estimators=estimators2, voting='soft')
voting3.fit(x_train, y_train)

#FINAL AUC-ROC curve setup - Create probability prediction
y_pred_logistic1 = lr_final.decision_function(x_test)
y_pred_svm1 = svm_final.decision_function(x_test)
y_pred_knn1 = knn_final.predict_proba(x_test)
y_pred_vote1 = voting3.predict_proba(x_test)

#FINAL AUC-ROC curves
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
#LR - create curve
logistic_fpr, logistic_tpr, threshold = roc_curve(y_test, y_pred_logistic1)
auc_logistic_final = auc(logistic_fpr, logistic_tpr)
#SVM - create curve
svm_fpr, svm_tpr, threshold = roc_curve(y_test, y_pred_svm1)
auc_svm_final = auc(svm_fpr, svm_tpr)
#KNN - create curve
knn_fpr, knn_tpr, threshold = roc_curve(y_test, y_pred_knn1[:,1])
auc_knn_final = auc(knn_fpr, knn_tpr)
#Vote - create curve
vote_fpr, vote_tpr, threshold = roc_curve(y_test, y_pred_vote1[:,1])
auc_vote_final = auc(vote_fpr, vote_tpr)
```

Appendix B1: Python Code - Model 2 - Data Preparation/Data Preprocessing

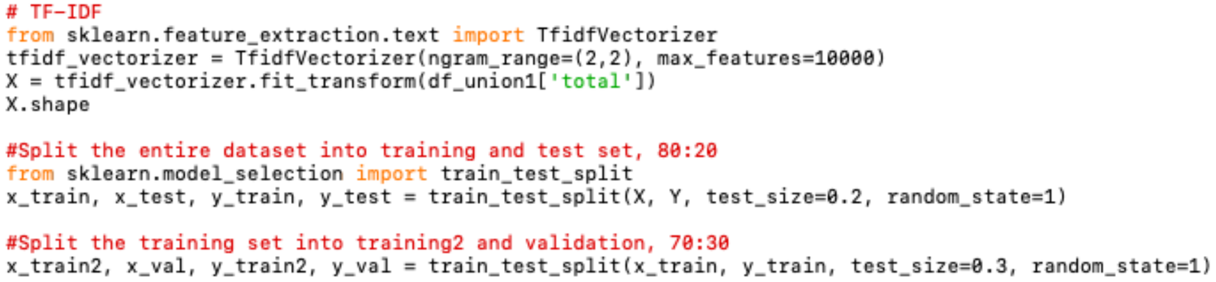


```
import pandas as pd

#Install Kaggle API. Makes it easier to import data directly from Kaggle, instead of importing from computer
! mkdir ~/.kaggle

#Install Kaggle API.
! pip install kaggle

#Install Kaggle API. Need kaggle.json file. Instructions: https://www.kaggle.com/general/74235
! cp kaggle.json ~/.kaggle/

#Install Kaggle API.
! chmod 600 ~/.kaggle/kaggle.json

#Download Kaggle1 directly from Kaggle.com
! kaggle competitions download -c fake-news

#Download Kaggle2 directly from Kaggle.com
! kaggle datasets download jruvika/fake-news-detection

#Unzip Kaggle 1 folder
!unzip /content/train.csv.zip

#Unzip Kaggle 2 folder
!unzip /content/fake-news-detection.zip

#Import both Kaggle1+2
kaggle2 = pd.read_csv("data.csv")
kaggle1 = pd.read_csv("train.csv")
kaggle1.info()
kaggle2.info()

#Copy datasets - so I still retain the original dataset, and can refer back to the original if needed
df_kag1 = kaggle1.copy()
df_kag2 = kaggle2.copy()

#Change column names for Kaggle2
df_kag2 = df_kag2.rename(columns={'Headline':'title','Label':'label', 'Body':'text'})
df_kag2.info()

#Remove nulls from 'title' in Kaggle1
df_kag1 = df_kag1[df_kag1['title'].notna()]
df_kag1.info()

#Merge datasets
df_union1 = pd.concat([df_kag1,
                       df_kag2], ignore_index=True)
df_union1.info()

#Drop features not being used: 'news_url','tweet_ids','author','text','URLs','Body'
features_dropped = ['URLs']
df_union1 = df_union1.drop(features_dropped, axis=1)
df_union1.info()

#Check duplicates - no duplicates
df_union1.shape , df_union1.drop_duplicates().shape

#Merge 'title' and 'text' to create 'total'
df_union1['total']=df_union1['title']+' '+df_union1['text']
df_union1.shape

#Convert 'total' to a string
df_union1['total']=df_union1['total'].apply(str)

#Import NLTK libraries. Used for data preprocessing.
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import re
```



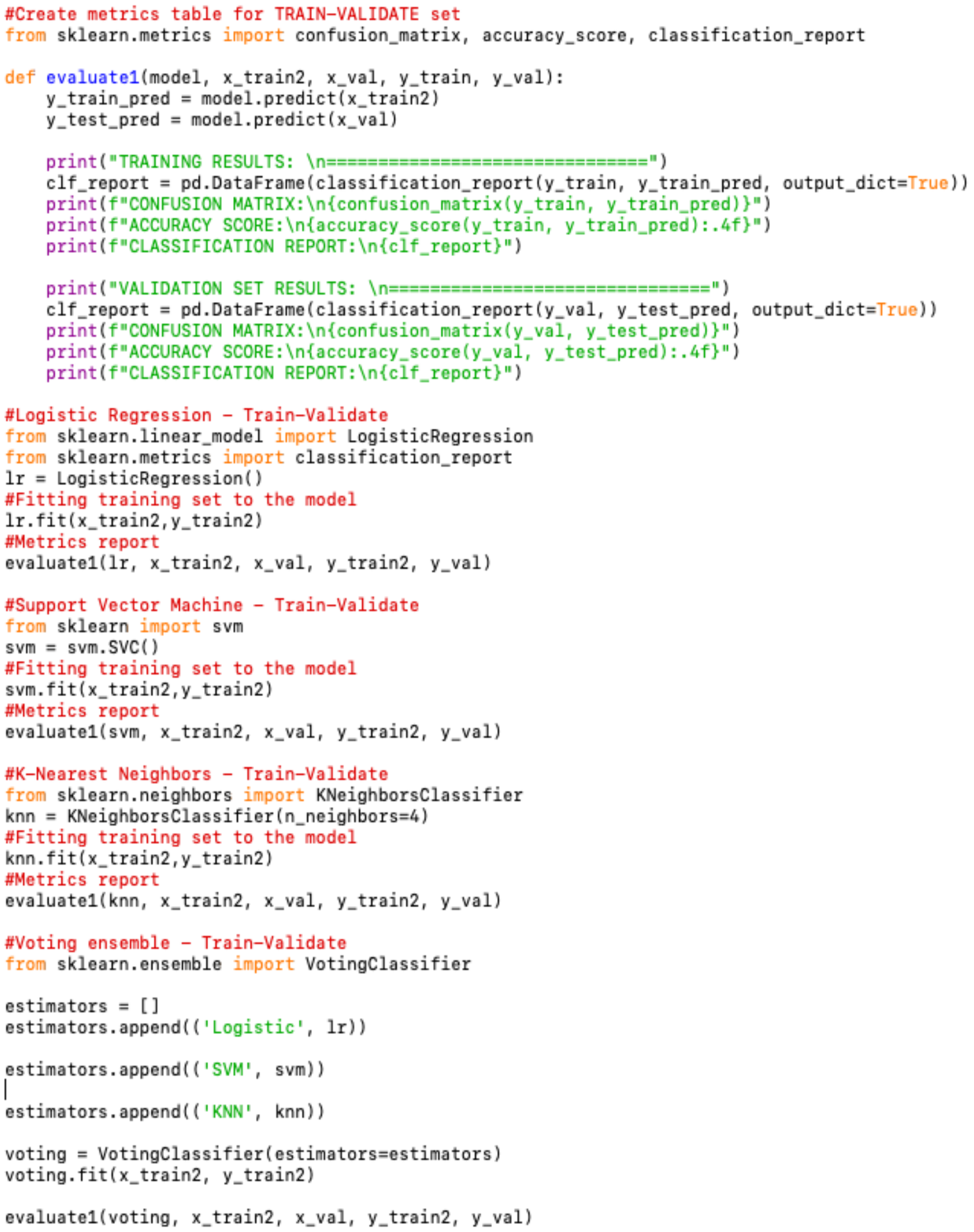
```
#Data preprocessing: stop words, tokenize, stemming
ps = PorterStemmer()
def stop_stem(text):
    text = re.sub('[^a-zA-Z]', ' ',text) #SYMBOLS
    text = text.lower() #LOWERCASE
    text = text.split() #TOKENIZE: .split() splits the line into words with delimiter as ' '
    text = [ps.stem(word) for word in text if not word in stopwords.words('english')] #STEMMING
    text = ' '.join(text) # this basically joins back and returns the cleaned sentence
    return text

print(stopwords.words('english'))

#Apply stop words, tokenization, stemming
df_union1['total'] = df_union1['total'].apply(stop_stem)

#Data After Preprocessing
df_union1.head()

#Independent variable: 'title'; Dependent variable: 'label'
X = df_union1['total']
Y = df_union1['label']
```

Appendix B2: Python Code - Model 2 - Data Vectorization/Train-Validate-Test Split



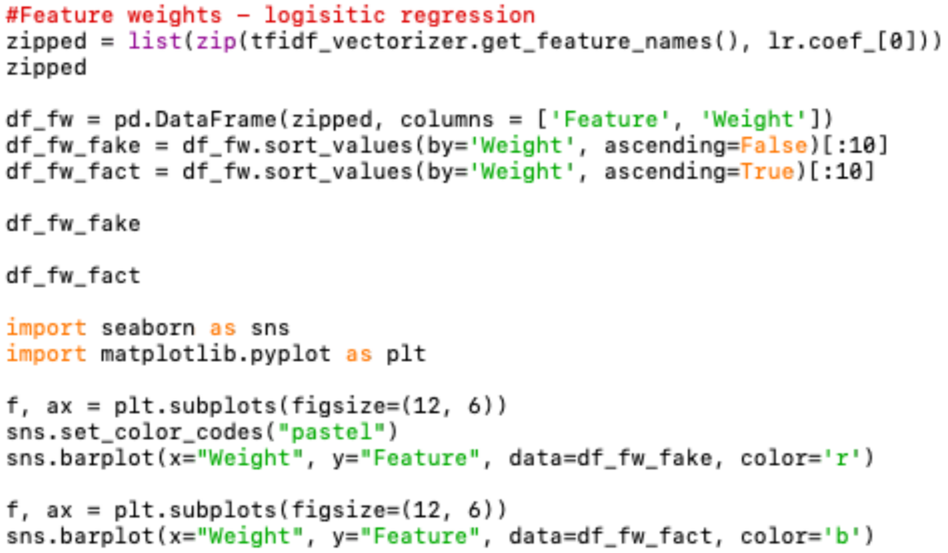
```
# TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2), max_features=10000)
X = tfidf_vectorizer.fit_transform(df_union1['total'])
X.shape

#Split the entire dataset into training and test set, 80:20
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)

#Split the training set into training2 and validation, 70:30
x_train2, x_val, y_train2, y_val = train_test_split(x_train, y_train, test_size=0.3, random_state=1)
```

Appendix B3: Python Code - Model 2 - Model Training



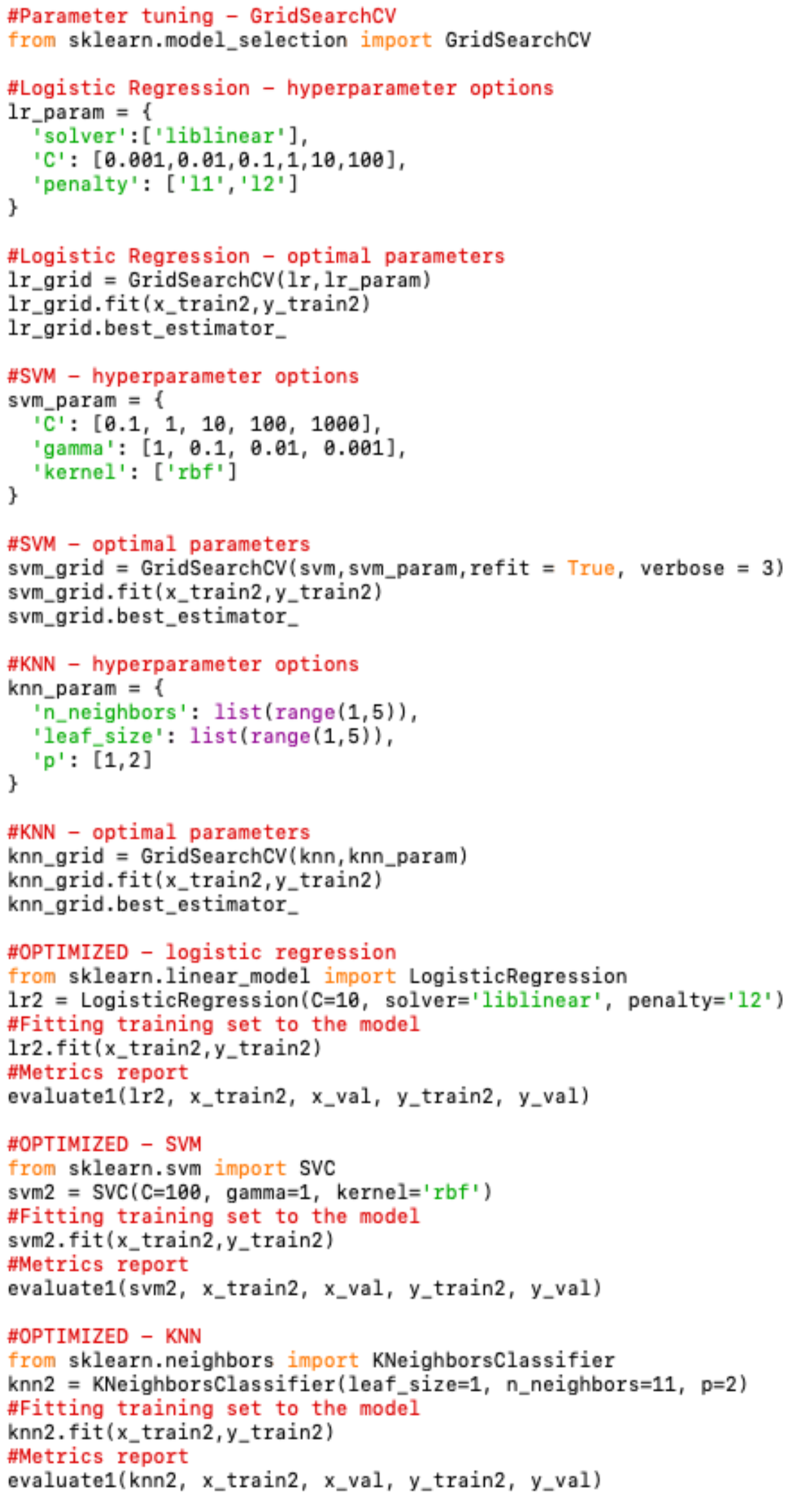
```
#Create metrics table for TRAIN-VALIDATE set
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

def evaluate1(model, x_train2, x_val, y_train, y_val):
    y_train_pred = model.predict(x_train2)
    y_test_pred = model.predict(x_val)

    print("TRAINING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("VALIDATION SET RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_val, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_val, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_val, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

#Logistic Regression - Train-Validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
lr = LogisticRegression()
#Fitting training set to the model
lr.fit(x_train2,y_train2)
#Metrics report
evaluate1(lr, x_train2, x_val, y_train2, y_val)

#Support Vector Machine - Train-Validate
from sklearn import svm
svm = svm.SVC()
#Fitting training set to the model
svm.fit(x_train2,y_train2)
#Metrics report
evaluate1(svm, x_train2, x_val, y_train2, y_val)

#K-Nearest Neighbors - Train-Validate
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
#Fitting training set to the model
knn.fit(x_train2,y_train2)
#Metrics report
evaluate1(knn, x_train2, x_val, y_train2, y_val)

#Voting ensemble - Train-Validate
from sklearn.ensemble import VotingClassifier

estimators = []
estimators.append(('Logistic', lr))

estimators.append(('SVM', svm))
|
estimators.append(('KNN', knn))

voting = VotingClassifier(estimators=estimators)
voting.fit(x_train2, y_train2)

evaluate1(voting, x_train2, x_val, y_train2, y_val)
```

Appendix B4: Python Code - Model 2 - Feature Weights

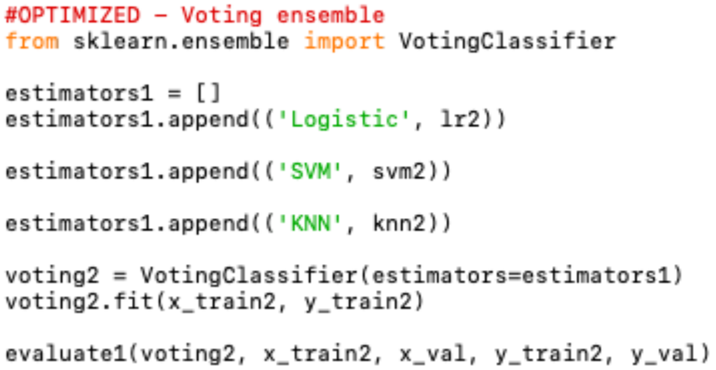


```
#Feature weights - logisitic regression
zipped = list(zip(tfidf_vectorizer.get_feature_names(), lr.coef_[0]))
zipped

df_fw = pd.DataFrame(zipped, columns = ['Feature', 'Weight'])
df_fw_fake = df_fw.sort_values(by='Weight', ascending=False)[:10]
df_fw_fact = df_fw.sort_values(by='Weight', ascending=True)[:10]

df_fw_fake

df_fw_fact

import seaborn as sns
import matplotlib.pyplot as plt

f, ax = plt.subplots(figsize=(12, 6))
sns.set_color_codes("pastel")
sns.barplot(x="Weight", y="Feature", data=df_fw_fake, color='r')

f, ax = plt.subplots(figsize=(12, 6))
sns.barplot(x="Weight", y="Feature", data=df_fw_fact, color='b')
```

Appendix B5: Python Code - Model 2 - Model Hypertuning

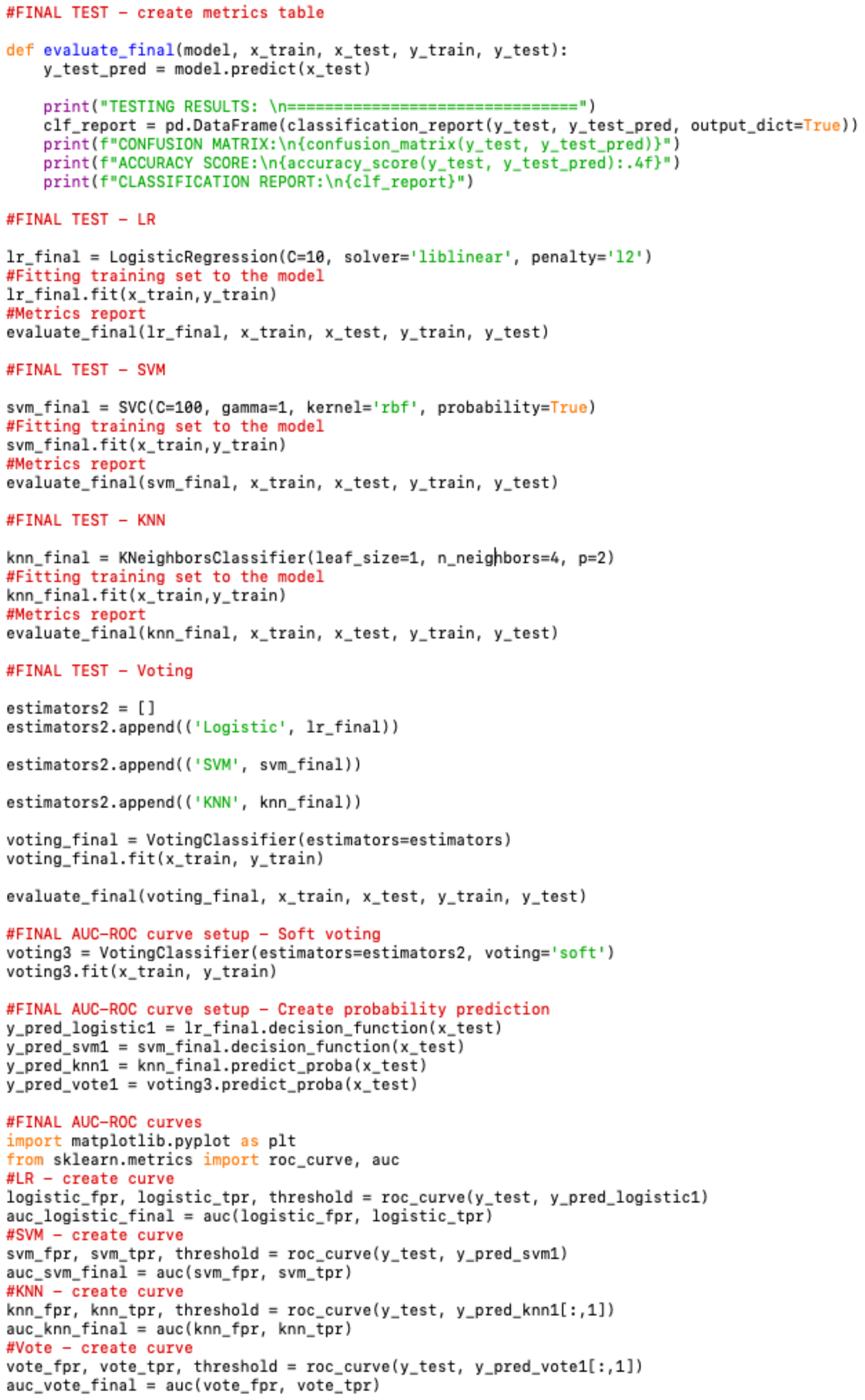


```
#Parameter tuning - GridSearchCV
from sklearn.model_selection import GridSearchCV

#Logistic Regression - hyperparameter options
lr_param = {
  'solver':['liblinear'],
  'C': [0.001,0.01,0.1,1,10,100],
  'penalty': ['l1','l2']
}

#Logistic Regression - optimal parameters
lr_grid = GridSearchCV(lr,lr_param)
lr_grid.fit(x_train2,y_train2)
lr_grid.best_estimator_

#SVM - hyperparameter options
svm_param = {
  'C': [0.1, 1, 10, 100, 1000],
  'gamma': [1, 0.1, 0.01, 0.001],
  'kernel': ['rbf']
}

#SVM - optimal parameters
svm_grid = GridSearchCV(svm,svm_param,refit = True, verbose = 3)
svm_grid.fit(x_train2,y_train2)
svm_grid.best_estimator_

#KNN - hyperparameter options
knn_param = {
  'n_neighbors': list(range(1,5)),
  'leaf_size': list(range(1,5)),
  'p': [1,2]
}

#KNN - optimal parameters
knn_grid = GridSearchCV(knn,knn_param)
knn_grid.fit(x_train2,y_train2)
knn_grid.best_estimator_

#OPTIMIZED - logistic regression
from sklearn.linear_model import LogisticRegression
lr2 = LogisticRegression(C=10, solver='liblinear', penalty='l2')
#Fitting training set to the model
lr2.fit(x_train2,y_train2)
#Metrics report
evaluate1(lr2, x_train2, x_val, y_train2, y_val)

#OPTIMIZED - SVM
from sklearn.svm import SVC
svm2 = SVC(C=100, gamma=1, kernel='rbf')
#Fitting training set to the model
svm2.fit(x_train2,y_train2)
#Metrics report
evaluate1(svm2, x_train2, x_val, y_train2, y_val)

#OPTIMIZED - KNN
from sklearn.neighbors import KNeighborsClassifier
knn2 = KNeighborsClassifier(leaf_size=1, n_neighbors=11, p=2)
#Fitting training set to the model
knn2.fit(x_train2,y_train2)
#Metrics report
evaluate1(knn2, x_train2, x_val, y_train2, y_val)
```

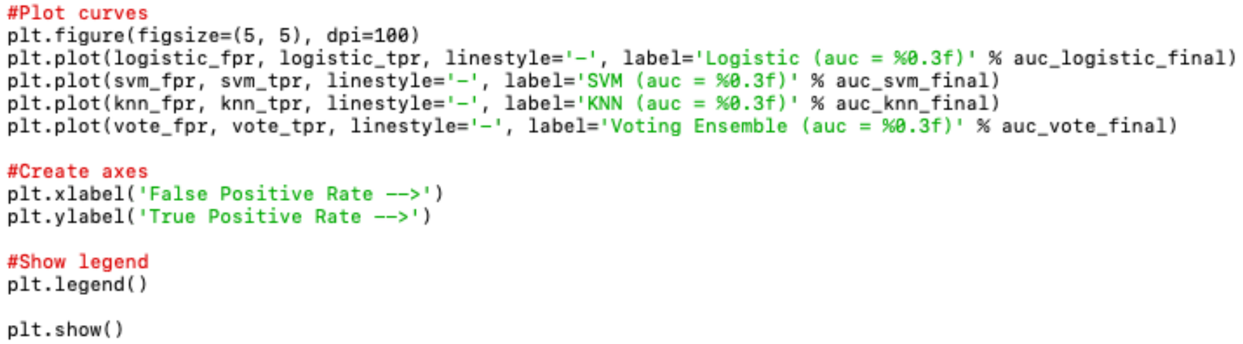


```
#OPTIMIZED - Voting ensemble
from sklearn.ensemble import VotingClassifier

estimators1 = []
estimators1.append(('Logistic', lr2))

estimators1.append(('SVM', svm2))

estimators1.append(('KNN', knn2))

voting2 = VotingClassifier(estimators=estimators1)
voting2.fit(x_train2, y_train2)

evaluate1(voting2, x_train2, x_val, y_train2, y_val)
```

Appendix B6: Python Code - Model 2 - Model Testing



```python
#FINAL TEST - create metrics table

def evaluate_final(model, x_train, x_test, y_train, y_test):
    y_test_pred = model.predict(x_test)

    print("TESTING RESULTS: \n================================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

#FINAL TEST - LR

lr_final = LogisticRegression(C=10, solver='liblinear', penalty='l2')
#Fitting training set to the model
lr_final.fit(x_train,y_train)
#Metrics report
evaluate_final(lr_final, x_train, x_test, y_train, y_test)

#FINAL TEST - SVM

svm_final = SVC(C=100, gamma=1, kernel='rbf', probability=True)
#Fitting training set to the model
svm_final.fit(x_train,y_train)
#Metrics report
evaluate_final(svm_final, x_train, x_test, y_train, y_test)

#FINAL TEST - KNN

knn_final = KNeighborsClassifier(leaf_size=1, n_neighbors=4, p=2)
#Fitting training set to the model
knn_final.fit(x_train,y_train)
#Metrics report
evaluate_final(knn_final, x_train, x_test, y_train, y_test)

#FINAL TEST - Voting

estimators2 = []
estimators2.append(('Logistic', lr_final))

estimators2.append(('SVM', svm_final))

estimators2.append(('KNN', knn_final))

voting_final = VotingClassifier(estimators=estimators)
voting_final.fit(x_train, y_train)

evaluate_final(voting_final, x_train, x_test, y_train, y_test)

#FINAL AUC-ROC curve setup - Soft voting
voting3 = VotingClassifier(estimators=estimators2, voting='soft')
voting3.fit(x_train, y_train)

#FINAL AUC-ROC curve setup - Create probability prediction
y_pred_logistic1 = lr_final.decision_function(x_test)
y_pred_svm1 = svm_final.decision_function(x_test)
y_pred_knn1 = knn_final.predict_proba(x_test)
y_pred_vote1 = voting3.predict_proba(x_test)

#FINAL AUC-ROC curves
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
#LR - create curve
logistic_fpr, logistic_tpr, threshold = roc_curve(y_test, y_pred_logistic1)
auc_logistic_final = auc(logistic_fpr, logistic_tpr)
#SVM - create curve
svm_fpr, svm_tpr, threshold = roc_curve(y_test, y_pred_svm1)
auc_svm_final = auc(svm_fpr, svm_tpr)
#KNN - create curve
knn_fpr, knn_tpr, threshold = roc_curve(y_test, y_pred_knn1[:,1])
auc_knn_final = auc(knn_fpr, knn_tpr)
#Vote - create curve
vote_fpr, vote_tpr, threshold = roc_curve(y_test, y_pred_vote1[:,1])
auc_vote_final = auc(vote_fpr, vote_tpr)
```



```
#Plot curves
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(logistic_fpr, logistic_tpr, linestyle='-', label='Logistic (auc = %0.3f)' % auc_logistic_final)
plt.plot(svm_fpr, svm_tpr, linestyle='-', label='SVM (auc = %0.3f)' % auc_svm_final)
plt.plot(knn_fpr, knn_tpr, linestyle='-', label='KNN (auc = %0.3f)' % auc_knn_final)
plt.plot(vote_fpr, vote_tpr, linestyle='-', label='Voting Ensemble (auc = %0.3f)' % auc_vote_final)

#Create axes
plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')

#Show legend
plt.legend()

plt.show()
```

References

Ahlers, D. (2006). News consumption and the new electronic media. *Harvard International Journal of Press/Politics*, *11*(1), 29–52. https://doi.org/10.1177/1081180x05284317

Ahmad, I., Yousaf, M., Yousaf, S., & Ahmad, M. O. (2020). Fake news detection using machine learning ensemble methods. *Complexity*, *2020*, 1–11. https://doi.org/10.1155/2020/8885861

Aslam, N., Ullah Khan, I., Alotaibi, F. S., Aldaej, L. A., & Aldubaikil, A. K. (2021). Fake detect: A deep learning ensemble model for fake news detection. *Complexity*, *2021*, 1–8. https://doi.org/10.1155/2021/5557784

Bleakley, P. (2021). Panic, pizza and mainstreaming the alt-right: A social media analysis of Pizzagate and the rise of the QAnon conspiracy. *Current Sociology*, 001139212110348. https://doi.org/10.1177/00113921211034896

Bloomberg, A. (2017, August 3). Facebook Has a New Plan to Curb 'Fake News'. Fortune. https://fortune.com/2017/08/03/facebook-fake-news-algorithm/

Brasoveanu, A., & Andonie, R. (2020). Integrating machine learning techniques in semantic fake news detection. *Neural Processing Letters*. https://doi.org/10.1007/s11063-020-10365-x

Dean, W. (2021). Journalism Essentials. *American Press Institute*. https://www.americanpressinstitute.org/journalism-essentials/what-is-journalism/purpose-journalism/

Fauzi, A., Setiawan, E. B., & Baizal, Z. K. (2019). Hoax news detection on Twitter using term frequency inverse document frequency and support vector machine Method. *Journal of*

Physics: Conference Series, *1192*, 012025. https://doi.org/10.1088/1742-6596/1192/1/012025

Furnkranz, J. (1998). A Study Using n-gram Features for Text Categorization. Austrian Research Institute for Artificial Intelligence. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.133

Gravanis, G., Vakali, A., Diamantaras, K., & Karadais, P. (2019). Behind the cues: A benchmarking study for fake news detection. *Expert Systems with Applications*, *128*, 201–213. https://doi.org/10.1016/j.eswa.2019.03.036

Hakak, S., Alazab, M., Khan, S., Gadekallu, T. R., Maddikunta, P. K., & Khan, W. Z. (2021). An ensemble machine learning approach through effective feature extraction to classify fake news. *Future Generation Computer Systems*, *117*, 47–58. https://doi.org/10.1016/j.future.2020.11.022

Hua, J., & Shaw, R. (2020). Corona virus (covid-19) “infodemic” and emerging issues through a data lens: The case of china. *International Journal of Environmental Research and Public Health*, *17*(7), 2309. https://doi.org/10.3390/ijerph17072309

Jianqiang, Z., & Xiaolin, G. (2017). Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, *5*, 2870–2879. https://doi.org/10.1109/access.2017.2672677

Khan, J. Y., Khondaker, M. T., Afroz, S., Uddin, G., & Iqbal, A. (2021). A benchmark study of machine learning models for online fake news detection. *Machine Learning with Applications*, *4*, 100032. https://doi.org/10.1016/j.mlwa.2021.100032

Khanam, Z., Alwasel, B. N., Sirafi, H., & Rashid, M. (2021). Fake news detection using machine learning approaches. *IOP Conference Series: Materials Science and Engineering*, *1099*(1), 012040. https://doi.org/10.1088/1757-899x/1099/1/012040

Kulkarni, P. (2021). Fake News Detection using Machine Learning. *ITM Web of Conferences*.

Liu, Y. (2019). Early detection of fake news on social media. Dissertation. 1436. https://digitalcommons.njit.edu/dissertations/1436

Mahabub, A. (2020). A robust technique of fake news detection using ENSEMBLE Voting classifier and comparison with other classifiers. *SN Applied Sciences*, *2*(4). https://doi.org/10.1007/s42452-020-2326-y

McGonagle, T. (2017). "Fake news": False fears or real concerns? Netherlands Quarterly of Human Rights, 35 (4) (2017), pp. 203-209, https://journals.sagepub.com/doi/10.1177/0924051917738685

Mitchell, A., Jurkowitz, M., Oliphant, J., & Shearer, E. (2020). Americans Who Mainly Get Their News on Social Media Are Less Engaged, Less Knowledgeable. Pew Research Center. https://www.journalism.org/2020/07/30/americans-who-mainly-get-their-news-on-social-media-are-less-engaged-less-knowledgeable/

Moawad, I., Alromima, W., & Elgohary, R. (2018). Bi-Gram Term Collocations-based Query Expansion Approach for Improving Arabic Information Retrieval. *Arabian Journal for Science and Engineering*, *43*(12), 7705–7718. https://doi.org/10.1007/s13369-018-3145-y

Naeem, S. B., Bhatti, R., & Khan, A. (2021). An exploration of how fake news is taking over social media and putting public health at risk. Health Information and Libraries Journal, 38(2), 143-149. https://doi.org/10.1111/hir.12320

Osamor, V., & Okezie, A.. (2021). Enhancing the weighted voting ensemble algorithm for tuberculosis predictive diagnosis. *Scientific Reports*, *11*(1). https://doi.org/10.1038/s41598-021-94347-6

Patterson, S. (2016, November 17). How Facebook Users Can Protect Against Fake News. *Network World (Online)*. https://www.proquest.com/docview/1841250667?accountid=25320

Ruvika, J. (2017, December 7). *Fake News Detection*. Kaggle. https://www.kaggle.com/jruvika/fake-news-detection

Satapathy, S. K., Jagadev, A. K., & Dehuri, S. (2017). Weighted majority voting based ensemble of classifiers using different machine learning techniques for classification of EEG signal to detect epileptic seizure. *Informatica (Ljubljana)*, *41*(1), 99–.

Shu, K., Mahudeswaran, D., Wang, S., Lee, D., & Liu, H. (2020). FakeNewsNet: A Data Repository with News Content, Social Context, and Spatiotemporal Information for Studying Fake News on Social Media. *Big Data*, *8*(3), 171–188. https://doi.org/10.1089/big.2020.0062

Thota, A. (2021). Fake News Detection: A Deep Learning Approach. SMU Data Science Review, 9(8), 1046–1050. https://doi.org/10.30534/ijeter/2021/01982021

UTK Machine Learning Club. (2018, February 22). *Fake News*. Kaggle. https://www.kaggle.com/c/fake-news