Lappeenranta-Lahti University of Technology
School of Engineering Science
Degree programme in Software Engineering

# Effects of emotive language on performance of fake news detection model

Teodora Kostoska

Examiner: Post-doctoral researcher Annika Wolff

# TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma
Teodora Kostoska

**Tunnepitoisen kielenkäytön vaikutus valheellisten uutisten havaitsevan koneoppimismallin suorituskykyyn**

Kandidaatintyö 2020

47 sivua, 11 kuva, 1 taulukko, 1 liite
Työn tarkastajat: Tutkijatohtori Annika Wolff

Hakusanat: koneoppiminen, luokittelu, tunneanalyysi, luonnollisen kielen käsittely, valeuutiset
Keywords: machine learning, classification, sentiment analysis, natural language processing, fake news

Valeuutisten kasvanut määrä on aiheuttanut tarvetta löytää erilaisia menetelmiä valeuutisten havaitsemiseen. Yksi tapa havaita valeuutisia on koneoppimismenetelmien avulla. Vaikka erilaisia valeuutisten havaitsemiseen käytettyjä koneoppimismalleja on tutkittu paljon, ei ole vielä paljon tutkimusta siitä, miten tunnetieto vaikuttaa koneoppimismallien luokittelutarkkuuteen. Tunnetiedolla tarkoitetaan tietoaineiston kunkin uutisartikkelin yleistä sävyä, eli onko artikkeli positiivinen, neutraali vai negatiivinen. Tämän työn tavoitteena on selvittää, millainen vaikutus tunnetiedolla on kahteen suosituimpaan valeuutisten havaitsemiseen käytetyn koneoppimismallin suorituskykyyn. Nämä mallit ovat Naïve Bayes ja Support Vector Machine. Tunneanalyysi tehtiin käyttäen TextBlobia ja Vaderia, jotka ovat Pythonista löytyviä valmiiksi valmennettuja ja testattuja tunneanalyysityökaluja. Tulokset näyttivät, että tunnetiedolla ei ollut merkittävä vaikutus valeuutisia havaitsevien koneoppimismallien luokittelutarkkuuteen. Suurimmassa osassa tuloksista tunnetietojen lisääminen hiukan laski mallien tarkkuutta.

# ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Degree Programme in Software Engineering
Teodora Kostoska

**Effects of emotive language on performance of fake news detection model**

Bachelor's Thesis 2020

47 pages, 11 figures, 1 table, 1 appendices
Examiner: Post-doctoral researcher Annika Wolff

Keywords: machine learning, sentiment analysis, natural language processing, classification, fake news

The increased amount of fake news has created a demand for different fake news detection methods. One way to detect fake news is with machine learning models. While there is a lot of research on different fake news detection models, there is not that much research done on the effects of sentiment information on the classification accuracy of the models. Sentiment information means the overall tone of each of the news articles in the dataset, whether it is positive, neutral, or negative. The goal of this thesis is to find out how sentiment information affects the performance of two of the most popular fake news detection machine learning models. These models are Naïve Bayes and Support Vector Machine. The sentiment analysis was done with TextBlob and Vader, which are already tested and trained sentiment analysis tools available in Python. The results showed that sentiment information did not have any significant effect on the classification accuracy of the fake news detection models. In most of the cases the addition of the sentiment information slightly decreased the accuracy of the models.

# TABLE OF CONTENTS

## Abbreviations and symbols

| | |
|---|---|
| NLP | Natural Language Processing |
| SVM | Support Vector Machine |

# 1 INTRODUCTION

The way people consume their news has changed drastically over the years. In addition to traditional ways of consuming news i.e., print media, nowadays news articles can also be found online, via search engine or on online platforms of popular newspapers, as well as on social media. On social media news content can be shared from user to user without any fact checking. It is beginning to be hard to recognize reliable news articles from unreliable ones. The Oxford English Dictionary defines fake news followingly: "False information that is broadcast or published as news for fraudulent or politically motivated purposes". While fake news has been around always, there was a spike in fake news articles during the 2016 US presidential election. This spike of fake news created an increased need for ways to detect fake news from real ones.

The goal of fake news detection is to find out the probability an article is intentionally deceiving (Conroy et al., 2015). Fake news detection can be automated with a machine learning model. Poddar et al. (2019) have done a comparison of different machine learning models for detection of fake news. In the paper the Naïve Bayes and the Support Vector Machine (SVM) models produced a good accuracy when classifying fake news, with SVM model producing the best precision at 92.8 %. In a paper written by Kesarwani et al. (2020), which presented different supervised machine learning models for fake news detection, the Naïve Bayes model produced an accuracy of 94 %. Hussain et al. (2020) created a SVM model for the classification of news articles in the Bangla language, which produced a classification accuracy of 96.64 %. In this thesis the fake news detection will be done by implementing a Naïve Bayes and Support Vector Machine model.

As presented in the previous section, there is a lot of research done on the different models and methods to classify news articles into fake and real news. However, there is not much research done on the effects emotive language can have on the precision of the models. In a paper done by Guo et al. (2019a) a novel Emotion-based Fake News (EFN) framework for detecting fake news on microblogs was presented. The method analysed the emotions of both the publisher and the reader. By implementing a neural network that exploited the gained sentiment information Guo et al. (2019a) were able to increase the accuracy of the

fake news detection model by almost 12 %. In another study done by Guo et al. (2019b) a different framework called Dual Emotion-based fAke News (DEAN) was presented, which was based on the EFN framework. The resulting model was tested on two different datasets, with results increasing by almost 12 % on the first dataset and 13.8 % on the second dataset. Anoop et al. (2020) had a different approach to adding sentiment to a fake news detection model. The approach included adding emotion labels into the documents after words that can be considered emotive. This approach produced an accuracy of 79 % with the Naïve Bayes classification model and 90 % with the SVM model.

The goal of this thesis is to find out whether adding sentiment information, would have any effects on the prediction accuracy of the chosen machine learning models. The sentiment analysis will be done using the off-the-shelf sentiment analysis tools Vader and TextBlob.

The main research question for this thesis can be composed as:
*How does adding sentiment information affect the performance of the most popular models used for fake news detection?*

Additionally, the following question will be answered:
*Which sentiment analysis tool resulted in better performance of the most popular models used for fake news detection?*

The hypothesis is, that if the sentiment information is added to the model input, the precision of the machine learning model will improve. This is because fake news articles often try to elicit a reaction from the reader by appealing to their emotions, which might mean that the article will contain more negative or positive vocabulary.

Section 2 of this thesis contains information on related work. The third section contain information on the methods used in this thesis. The fourth section introduces the structure of the program and contains information on the environment and tools used in the implementation of the program. The fifth section presents the results. The sixth section contains the discussion on the results, as well as suggestions for future research. Last is the conclusion, which contains the summary of everything that has been addressed in this thesis.

# 2 LITERATURE REVIEW

This section contains the literature review for both the sentiment analysis and the fake news classification model. The performance of machine learning models has been researched plenty when it comes to fake news detection. In this thesis the concentration will be on the SVM and Naïve Bayes models.

The Naïve Bayes model has been used for text data classification and has produced good results, although other models have become more favourable (Lewis, 1998). Granik and Masyura (2017) used the Naïve Bayes model to detect fake news in Facebook news posts. While in their paper the model was used for fake news detection on social media i.e., Facebook, it still gives a good estimate of the accuracy of the Naïve Bayes model for fake news detection. Granic and Masyura (2017) got an accuracy of 75.59 % for true articles, 71.73 % for fake news articles and 75.4 % overall accuracy.

The SVM model has also been popular in automated fake news detection. An important aspect of the SVM model is the kernel, which is a mathematical function that takes the input data and transforms it into the correct form. The kernel is decided based on what types of data needs to be classified. In the case of fake news detection, the input data is a vector that represents the bag of words. The most common kernels for fake news detection were RBF (radial basis function) and linear kernel. A study by Tijare (2019), where a Support Vector machine with RBF kernel was used for fake news detection produced an accuracy of 87.37 %. While this is a very good accuracy, the model can be improved by using the linear kernel. A research by Hussain et al. (2020) done on the detection of fake news in articles in Bangla language used a linear kernel for the SVM model and produced an accuracy of 96.64 %.

Another aspect that can affect the performance of a machine learning model is the feature extraction method used. Feature extraction means that the textual data will be transformed into numeric form, which the machine learning model can process. In this thesis the two feature extraction models that are being considered are the Count Vectorization and the TF-IDF vectorizer. The TF-IDF method of creating the bag of words has given good results. In a study done by Kesarwani et al. (2020) the combination of feature extraction with TF-IDF

and the Naïve Bayes model for fake news detection produced an accuracy of 94.14 %, but in the same study the TF-IDF model did not work well with the Support Vector Machine, only giving an accuracy of 53 %. This could be due to the dataset and the feature extraction model, as in the same study the same model was tested with a different dataset with a feature selection model, and the SVM model returned an accuracy of 76.19 %. Feature selection can be applied if there is a reason to suspect that the dataset contains a lot of noise i.e., irrelevant features. The assumption for this thesis is that there will be no need to perform feature selection and feature extraction should be enough.

In a study by Poddar et al. (2019) the accuracy of five different fake news classification models combined with TF-IDF or Count Vectorization was calculated with different input sample sizes. The study shows that for the Naïve Bayes model and for larger sample sizes the accuracy is only slightly better with Count Vectorizer at an accuracy of 86.3 % compared to the TF-IDF model with accuracy of 85.4 %. A study done by Agudelo et al. (2018) for fake news detection produced a classification accuracy of 89.3 %, when combining the Naïve Bayes method with a Count Vectorizer model to extract features and gaining a lower accuracy, when combining the Naïve Bayes classifier with a TF-IDF. The difference in accuracy can be explained by the difference of dataset sizes.

For the Support Vector Machine model the study by Poddar et al. (2019) found that the model is dependent of input sample size and that the accuracy of the model is better when text vectorization is done with the TF-IDF model with an accuracy of 92.8 %, with the accuracy of the SVM model dropping to 89.1 % when the vectorization is done with Count Vectorization. A paper done by Vijayaraghavan et al. (2020) showed that the SVM model with the linear kernel produced an accuracy of 94.58 %, when combined with TF-IDF and an accuracy of 93.06 % when combined with Count Vectorization.

There is also some research done on different methods of sentiment analysis combined with fake news detection models. In a paper by Guo et al. (2019a) an Emotion-based Fake News (EFN) detection model was proposed. The goal of the model was to detect fake news on microblogs. The model calculated information on the emotion of both the publisher and the users i.e., was the text written by the publisher emotionally charged, and what kind of emotions did the text evoke in the readers and used the gained information to detect fake

news. The EFN model was built using neural networks. The EFN model increased the accuracy of the decision tree model by almost 12 %.

In a different paper by Guo et al. (2019b) a Dual Emotion-based fAke News (DEAN) detection framework was introduced, with the same aim of detecting fake news on microblogs. The second framework was based on the EFN model. This framework was also built to detect the emotions of both the publisher and the users. The proposed model by Guo et al. (2019b) increased the accuracy of the decision tree model by nearly 12 % on the first dataset, which was the same dataset used in the EFN model, and by 13.8 % on the second dataset.

Anoop et al. (2020) created a model that detected fake medical news by implementing emotion intensity lexicons to enrich emotion information within documents. The emotionalization of the documents means that emotion labels are added after words that evoke some emotion e.g., danger-> fear, so it will add the word fear into the document after the word danger. The model created by Anoop et al. (2020) was used with both the Naïve Bayes classification model, as well as the SVM model, along with a few other models. In the paper for document dimensionality of 100 the Naïve Bayes model produced an accuracy of 79 % and the SVM model an accuracy of 90 %. The same method was repeated with document dimensionality of 300, which produced accuracies of 83 % for the Naïve Bayes model and 89 % for the SVM model.

In an article by Pease (2018), the fake news detection was done by applying a sentiment value to each article using the sentiment analysis toolkit Vader after which the feature vector with the sentiment value added to it was put through a Naïve Bayes classification model. The result was that the precision of the Naïve Bayes model improved, when the sentiment value calculated by Vader was added to the feature vector. In this thesis the sentiment analysis will be done by implementing two off-the-shelf sentiment analysis tools available in Python: Vader and TextBlob.

# 3 METHODOLOGY

This section contains the descriptions of the methods that will be implemented in this thesis for both the sentiment analysis and the fake news classification model. Additionally, the section contains information on feature extraction, natural language processing and model evaluation.

## 3.1 Natural language processing

Before the dataset can be used in the sentiment analysis and in the fake news classifier, the text body data in the dataset needs to be pre-processed. The pre-processing will be done using Python's Natural Language Toolkit (NLTK). First the text data will be turned to lowercase, and any punctuation will be removed. Next all stop-words will be removed, this will be done by implementing the vector provided by NLTK, that contains the most common English stop-words i.e., the, and, or, ours. Next the text data needs to be tokenized, which means that the text is broken down into words. Additionally, the text will be lemmatized, which means that all words will be returned to their root form i.e. words->word, better->good. The outcome of the pre-processing is a corpus of documents, where each document is represented by a sequence of words.

## 3.2 Feature extraction

The corpus needs to be converted into a numeric form, for the machine learning model to be able to process the data (Poddar et al., 2019). The bag of words will be used to convert variable-length text into a fixed-length vector of numbers. The bag of words represents a vocabulary of known words and the measure of presence of those words in the text data. The bag of words can be created by using Count Vectorization or TF-IDF (Poddar et al, 2019).
In Count Vectorization the idea is to create vectors with dimensionality of the vocabulary in the corpus of documents, the size of a dimension increases every time the vocabulary word is encountered in the document. The output is an encoded vector that contains the vocabulary of the corpus with the frequency of each word in each document of the corpus (Poddar et al, 2019).

TF-IDF (Term Frequency-Inverse Document Frequency) consists of two values. TF is the frequency a word occurs in a document and IDF is the frequency a word occurs in a corpus of documents (Bali et al., 2019). The TF-IDF model evaluates how important a word is in the document and in the corpus (Kesarwani et al., 2020). The idea is that the rarer the word is in the document, the higher its TF-IDF value will be. The function for the TF-IDF calculation is:

$$TF - IDF = TF * IDF.$$

The TF part of the function calculates the count of a term in a document, divided by the total number of terms in the document. It returns a value between 1 and 0, where 0 means there are no instances of a word and 1 means that every word in the document is the same word. This can be represented as the following equation:

$$TF = \frac{t}{d},$$

where t = frequency of term in document and d = total number of terms in document.

The IDF value stands for the inverse of the frequency of a term in the corpus of documents. The calculation will return a low value for terms that appear frequently in documents. This can be represented as the following equation:

$$IDF = \log\left(\frac{N}{tf + 1}\right),$$

where N = corpus size and tf = frequency of term in documents.

The TF-IDF calculation returns a bag of words or a matrix of features, where each word in the corpus is given an importance value, based on the frequency the word appears in documents. The machine learning model can then use the vector to learn to classify the data, based on the importance of each word in the corpus.

The TF-IDF model of feature extraction was chosen for this thesis. The reasoning behind this choice is that the TF-IDF model produced good results when combined with both the Naïve Bayes and the SVM model in related work. It also produced better overall accuracy for the SVM model in similar work and produced only slightly worse results for the Naïve Bayes model when compared to the Count Vectorization model.

## 3.3 Sentiment analysis

As presented in the literature review section sentiment information can influence the outcome of the classification models. The models proposed by Guo et al. (2019) were both built with neural networks. Neural networks are based on the functionality of the brain and try to imitate the interconnections between braincells. Neural networks consist of layers, where each layer is responsible for different tasks, for example one task could be extracting sentiment information from text. In this thesis neural networks will not be implemented, as sentiment analysis will happen separately from the classification of the articles. The sentiment analysis will be done by using already trained and tested tools for sentiment analysis available in the Python Natural Language ToolKit (NLTK) and the TextBlob package.

From the NLTK the Vader sentiment analysis tool will be used. Vader is originally made for sentiment analysis of tweets, but it has been used for sentiment analysis of larger text data, and it can quite reliably predict the sentiment label of the text. Vader uses a bag of words model combined with heuristics i.e., if the word very is before some other word the intensity of the sentiment increases. The Vader model gives the correct polarity also for contradictions, punctuation, slang, and negations, which some other readily available sentiment analysis models cannot do. The output is calculated by a ratio of the proportion of text that fall into each category (positive, neutral, negative) (“Using Pre-trained VADER Models for NLTK Sentiment Analysis,” n.d.). Vader gives an output that contains a score for positivity, negativity, neutrality, and a compound polarity score, which is the overall sentiment of the article. The compound score is between -1 and 1, where -1 means strong negative, 1 means strong positive and 0 means neutral sentiment. Most articles should fall somewhere between these polarity scores.

The second proposed way to do the sentiment analysis is by using Python's TextBlob package. Both Vader and TextBlob use a sentiment lexicon, which contains words paired with their sentiment label. In addition to a polarity score, TextBlob also calculates the subjectivity score. Like in Vader the polarity score is between -1 and 1, while the subjectivity score is in the range of 1 and 0, where 1 means very subjective, and 0 means objective. The downside to TextBlob is that it does not contain the heuristics that Vader contains.

Both sentiment analysis methods will be tested with the classification models proposed in the next section. The goal is to find out whether the addition of the sentiment value to the input vector will improve the overall accuracy of the fake news classification models. Additional information on which sentiment analysis tool works better with the proposed fake news classification models should be gained.

### 3.4 Classification models

Machine learning methods can be roughly classified into supervised and unsupervised methods. The idea behind an unsupervised method is that the model is taught using unlabelled and unclassified data and it must independently find similar features between different data instances of the dataset e.g. clustering (Kotsiantis, 2007).While the model might not come to the desired solution, it can still find some conclusion and discover new unknown classes (Kotsiantis, 2007). The method used for classification of data is the supervised method. The idea behind the method is that the model is trained using correctly labelled and classified data instances. The method should then use the input data to learn similarities between data instances that have the same label. The goal is that the model will learn to correctly classify future data. In the case of fake news, the model will be trained using a dataset, with each instance being classified as fake or real news. The model should then in the future be able to classify new data instances correctly into fake and real news.

The two types of classification are binomial and multi-class. These refer to the desired output of the model. If the classification is multi-class, it means that the data should be classified

into more than 2 classes. The fake news classification is binomial, as there are two possibilities for the desired output: fake and real.

There are many possible machine learning models, which enable fake news classification. The goal of this section is to explain the theory behind the chosen classification methods.

One commonly used and one of the oldest models used for classification is the Naïve Bayes model (Lewis, 1998). The model uses the Bayes theorem, which is a conditional prediction formula. The Bayes theorem:

$$P(A|B) = (P(A) * P(B|A))/P(B)$$

The equation calculates the probability a value belongs to class A if B is true. The probability is calculated by multiplying the probability of A with the probability of B, when A is true, and dividing the result with the probability of B. In the Naïve Bayes method, the assumption is that a feature x is independent of any other features (Lewis, 1998).

From the Bayes theorem, it is possible to derive a formula for calculating the probability some news is fake. Granik and Mesyura (2017), who used a spam detection model as the basis of the fake news classifier to classify Facebook news posts, define the formula followingly:

$$P(F|W) = P(W|F) * P(F)/(P(W|F) * P(F) + P(W|T) * P(T))$$

In the equation F = fake, W = word and T = true. The sum of the probability of every significant word of a text document appearing in a fake article, divided by the sum of the probabilities the document is fake or true, when it contains the certain words gives the value of the total probability the document is fake (Granik and Mesyura, 2017). This value can then be used in the classifier model to classify documents into fake and true news.

Another model, which has been developed more recently than the Naïve Bayes model is the Support Vector Machine (SVM). The SVM model is a binary classifier, which means that

the data instances are classified into two classes. This model should work well for the classification of fake news, as there are only two possible classes the news can belong to. The SVM model tries to maximize the margin between the two classes and find a hyperplane that divides the dataset into two classes ("Support Vector Machines," n.d.). In a two-dimensional plane the hyperplane would be the line that divides the dataset into two classes, with each class being on a opposite side of the hyperplane (Poddar et al, 2019). The idea is that the farther the data point is from the hyperplane, the more likely it belongs to a certain class ("Support Vector Machines," n.d.). The kernel chosen for this thesis is the linear kernel, as it produced better overall accuracy when used for fake news detection.

The reasoning behind the chosen methods is that they have often been used in the classification of news articles into fake and real. This gives more information on how the models usually perform and gives an idea on how the model used in this thesis should perform. The goal is to try and get similar accuracies as in previous related work, and then see whether the addition of sentiment information would have any impact on the accuracy of the models.

## 3.5 Model evaluation

The accuracy of the fake news classification model will be evaluated with classification accuracy and a confusion matrix. Classification accuracy stands for the percentage of correctly made classifications from all the classifications made. The distribution of the fake news and true news in the dataset needs to be equal for the classification accuracy to work.

For more information, the confusion matrix is recommended. A confusion matrix was used for the evaluation of the classification models in both the paper done by Kesarwani et al. (2020) and the paper written by Agudelo et al. (2020). The confusion matrix is a more reliable way of determining accuracy, as it gives as an output a matrix that contains the number of true positives, true negatives, false positives, and false negatives. A visualization of the output of the confusion matrix can be seen in table 1.

**Table 1** Output of confusion matrix.

| | | **Predicted Label** | |
|---|---|---|---|
| | | Negative | Positive |
| **Actual label** | Negative | True negative | False positive |
| | Positive | False negative | True positive |

The percentual accuracy is then obtained by dividing the sum of true positives and true negatives with the total samples. To see whether accuracy improves when sentiment analysis is added to the dataset, the classification models need to be trained and tested with data that does not contain the sentiment analysis results, as well as with data that contains the sentiment analysis results.

# 4 PROPOSED APPROACH

This section contains information on the environment and tools used, as well as information on the dataset and program structure.

## 4.1 Environment and tools

The program was done with Python 3.8.5. The most important libraries used in the program were: NLTK and scikit-learn. The NLTK library was used for all the text pre-processing done on the article bodies in the dataset, as well as for the implementation of the Vader sentiment analysis tool. The scikit-learn library was used for the implementation of the dataset splitting into testing and training data, the feature extraction (TF-IDF), the model training and testing, the calculation of the model precisions, and the calculation of the confusion matrix. Additionally, the pandas library was used to import the dataset, seaborn for visualization of the dataset, mlxtend for plotting the confusion matrices, and TextBlob for implementation of the TextBlob sentiment analysis tool. There were also some more libraries used to make singular adjustments to the text bodies, plots, dataset arrays, and feature vectors.

## 4.2 Dataset

The dataset that will be used in this thesis will be the fake and real news dataset by Clement Bisaillon. The dataset can be found on Kaggle and it is free to use. The link to the dataset can be found in the bibliography section. The dataset consists of already labelled data of fake and real news articles. The reasoning behind the choice of using an already existing dataset was due to time limitations, as there simply was not enough time to create a new dataset from scratch. Additionally, this dataset has been used in similar work and using a pre-validated dataset gives input on how it should perform, whether the results can be replicated and would there be any effects on the results when sentiment information is added to the model input.

Figure 1 represents the division between true and fake articles in the dataset. The dataset contains 44 898 articles, out of which 23 481 are labelled fake and 21 417 are labelled true.

As the chosen dataset has an almost even distribution of fake news and true news, the classification accuracy can be used. Each data instance in the dataset contains information on title of article, article text body, subject, and date. From these only the text body and the label of the article will be used in the classification models. Additionally, the sentiment value of each article will be added to the input vector.

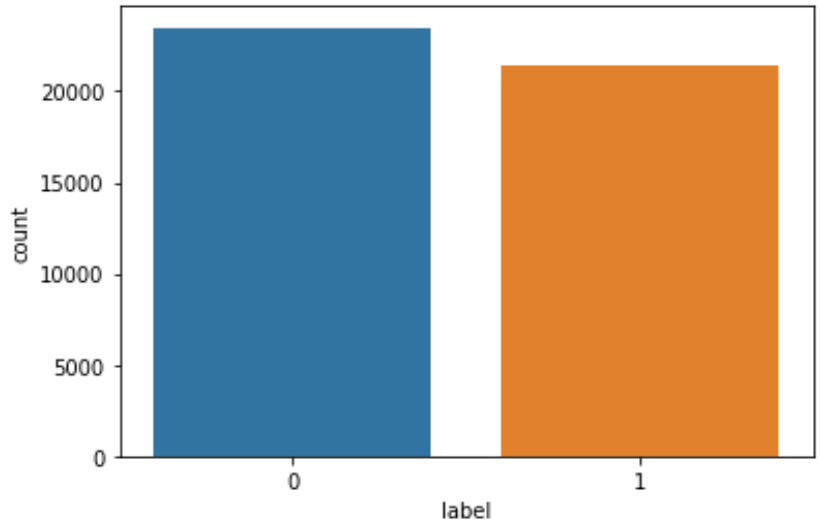


**Figure 1** Dataset division between fake and real articles. Source: George, 2020.

Upon inspection the dataset seems to be balanced enough to produce reliable machine learning models, without any additional restrictions. If the dataset contained significantly more articles from one of the categories, then the resulting machine learning model could be faulty, as it would give good accuracy by only classifying articles into the majority category. The dataset subjects vary from political news to US and middle east news, which is visualized in figure 2. Most of the data belongs to subject labels: political news, world news, news, and politics.

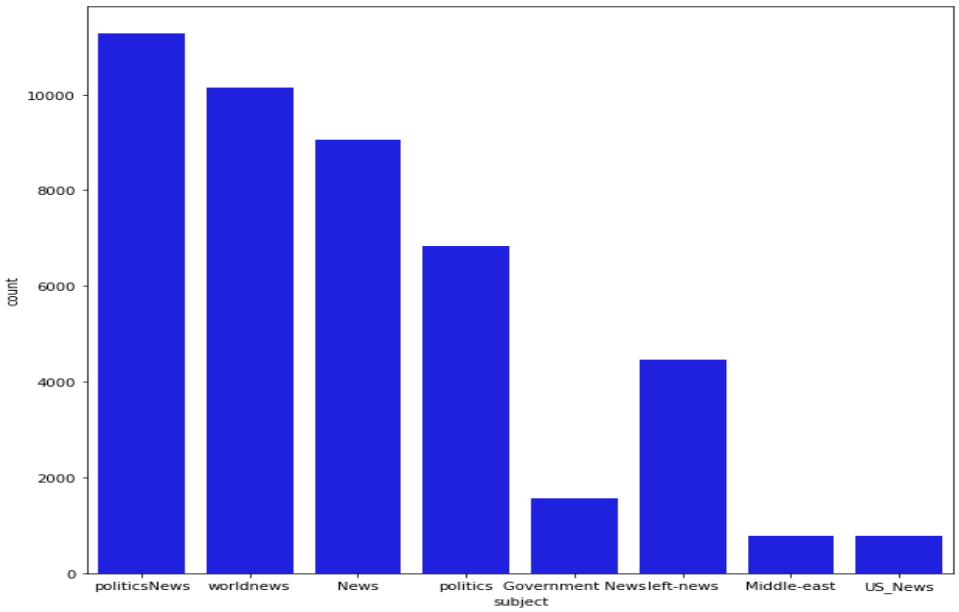


**Figure 2** Distribution of articles based on subject. Source: George, 2020.

The divide between true and fake news shows that most of the true news articles belong to subjects politics news and world news, while most of fake news articles belong to subjects news and politics (Figure 3). As all the articles from the true news classification belong to only two subjects, this means that the dataset is quite homogenous, which might cause some issues in the classification models, especially with overfitting. Overfitting means that the model will produce very good results on the training data and the testing data, but might not generalize well with unknown data i.e., will produce worse results with different dataset. Since the focus of this thesis was to find out how adding sentiment information would affect the accuracy of the classification model, the generalization ability of the model is not as important, thus it was decided that the dataset will be used despite the possibility of overfitting.

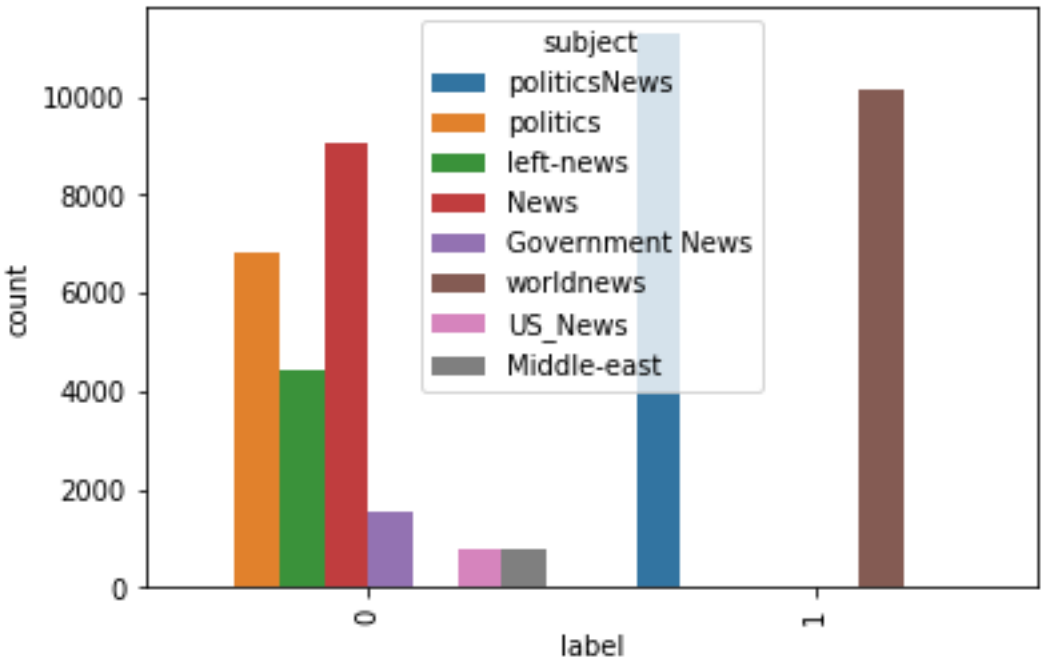


**Figure 3** Distribution between subjects based on label. Source: George, 2020.

### 4.3 Program

The program consists of a few basic components: sentiment analysis, text data pre-processing, data splitting into training and testing data, feature extraction, model training and testing, and evaluation of model performance. It is important to note that to be able to determine whether sentiment information has any effect on the performance of the classification models control results are necessary. These are gained by training and testing the model with the same dataset, but without the sentiment information. The code for the program is in attachment 1.

The program begins with the importation of the dataset, which is then shuffled, checked for any empty variables, and visualized. After this, all the unnecessary variables (title, subject, date) are removed from the dataset. Then the text pre-processing can begin on the article body in the dataset. The pre-processing is done in the following steps: all text is made lowercase, punctuation is removed, stop-words are removed, the text is tokenized, and all words are lemmatized. Figure 4 visualizes the steps.

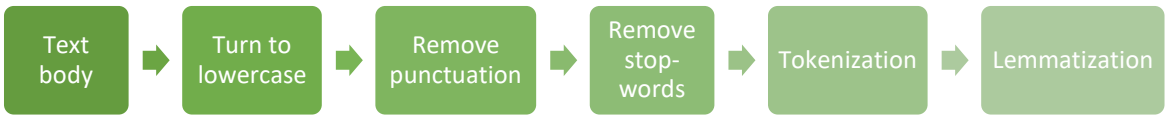


**Figure 4** Text pre-processing.

The outcome of the text pre-processing is a corpus of documents, which will need to be split into training and testing data. The split will be 80 % training and 20 % testing data, and it will be done by using scikit-learn's train-test-split function. The reasoning behind choosing this method to split the data instead of k-fold cross validation, which would better prevent the possibility of overfitting, is because the dataset is quite large, and it would require a large amount of processing power to implement it on the chosen dataset.

The corpus of documents, that was split into training and testing data, next needs to be converted into numeric form. This step is the feature extraction process, which will be done by using the TF-IDF vectorizer in scikit-learn. The training data will be put through the vectorizer first and it will be used to determine which features the text data in the corpus contains, this is called fitting of the vectorizer. The training data is then only transformed into features based on the fitted vectorizer. The output of the vectorizer is a matrix, which contains each feature or word in the corpus with an importance value for each document in the corpus.

The chosen classification models for this thesis were the Naïve Bayes and the Support Vector Machine models. The models will be implemented by using the scikit-learn library's MultinomialNB and SVC tools. The outcome matrix of the training data from the TF-IDF vectorization will be used to train each model. In the SVM model the kernel will be specified as linear. The outcoming trained models will then be used to predict the labels of the testing data. This outcome will then be used to calculate the accuracy of the model, as well as the confusion matrix for each model. The accuracy is gained by calculating how many of the predicted labels were correct, from all the predictions.

After the control results are gained, the process will be repeated two more times, with sentiment analysis being done by TextBlob first and Vader second. The sentiment analysis will be done before the pre-processing of the text data, as unprocessed data can contain more information on sentiment. The sentiment analysis results will be added into a new column in the dataset. The sentiment analysis results will be transformed into discrete values, as the Multinomial Naïve Bayes model cannot take negative values. The limits will be set at under -0.05 for negative, over 0.05 for positive, and everything between those numbers as neutral. The text data will then be pre-processed as shown in figure 4, after which the data will be split into testing and training data. The TF-IDF vectorization will be performed only on the corpus of documents i.e., the pre-processed text data, and not on the sentiment analysis results. The sentiment analysis results will then be added as an additional feature to the outcoming matrix from the TF-IDF vectorizer. Next the feature vectors from the training and testing data, along with the sentiment analysis results, will be used to train, and test the Naïve Bayes and SVM models. Lastly, the models will be evaluated by calculating the accuracy of the models on the testing data and the confusion matrix will be generated. These steps are represented in figure 5.

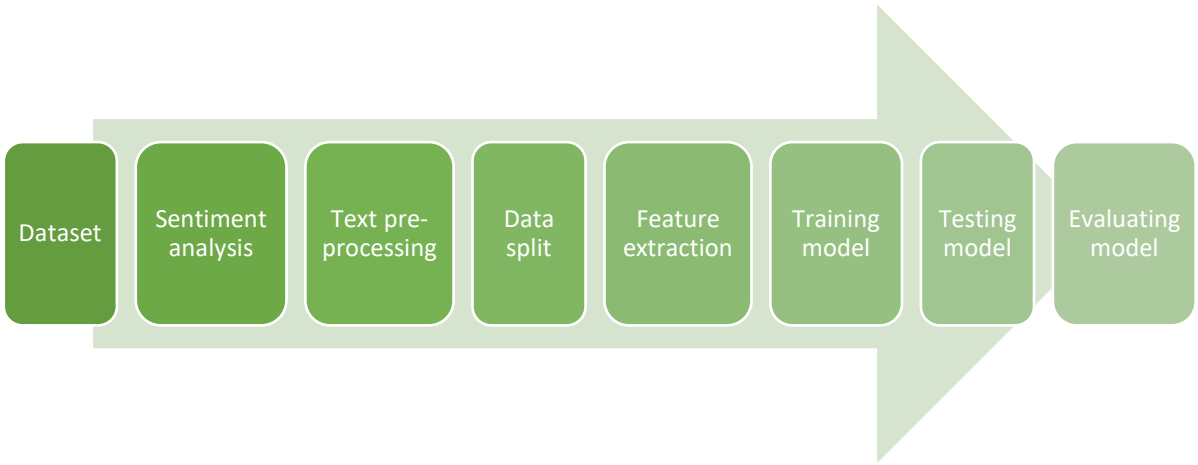


**Figure 5** Program structure.

# 5 RESULTS

This section contains information on the results of the program presented in section 3. The program was ran three times: without sentiment information, with sentiment information from TextBlob, and with sentiment information from Vader. As the program contained two different classification models the total outcome was six different accuracies and confusion matrices based on the testing data. The results are divided based on classification models.

## 5.1 Naïve Bayes results

The models' accuracy was calculated for both the training and the testing data to check how well the model would generalize on unknown data. The confusion matrix was created only from the predictions on the testing data. The confusion matrix shows how many articles were correctly classified into positive and negative, but also shows how many of the articles were labelled falsely positive or negative.

The control model, where the model was trained and tested without any sentiment information, produced an accuracy of 97.37 % on the training data and an accuracy of 96.92 % on the testing data. The results were very good from the beginning. The confusion matrix (figure 6) shows that this model produces more false positive results (5 %) than false negative (1 %). It means that the model falsely labels articles as true when they are fake more often than fake when the articles are true. Overall, 8703 articles were correctly classified, and 277 articles falsely classified.

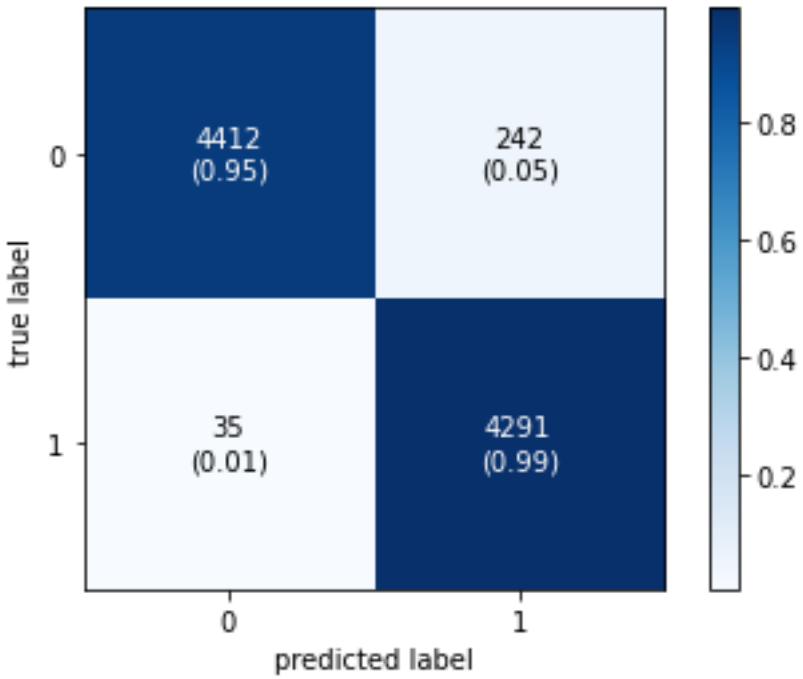


**Figure 6** Confusion matrix on testing data when model was trained and tested without sentiment information.

The model was then trained and tested with the same dataset, processed in the exact same way as in the control model, but with sentiment information gained from TextBlob added to the feature vector. The outcome of this model was an accuracy of 97.45 % on the training data and 96.86 % on the testing data. This accuracy was also very good but compared to the control accuracy it was slightly worse (0.06 %). The confusion matrix (figure 7) shows that the model classified the articles in a similar way, with the model classifying false articles as true more often (5 %) than true articles as false (1 %). The model correctly classified 8698 articles and falsely classified 283 articles, which is slightly worse than the control model.

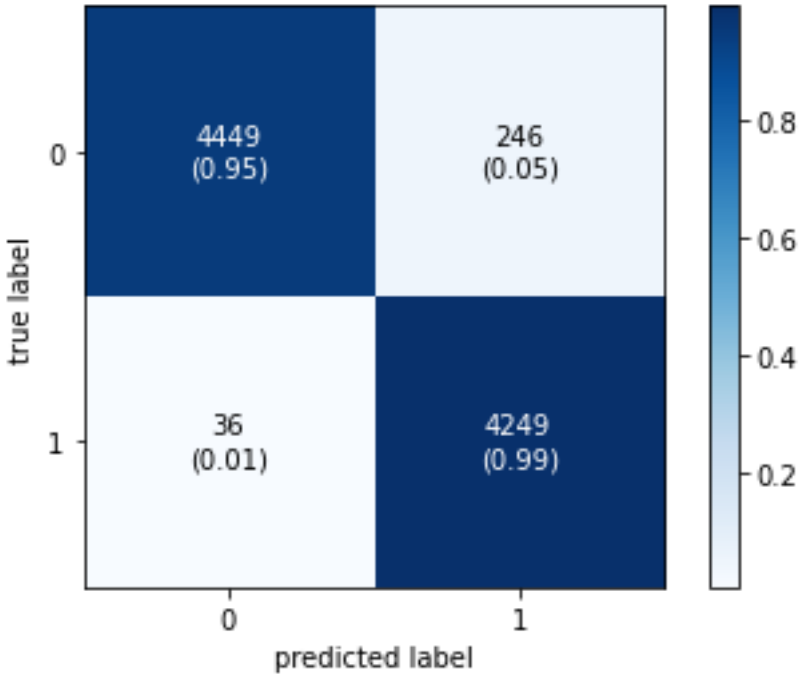


**Figure 7** Naive Bayes model results when trained and tested with data that contained sentiment information from TextBlob.

Lastly the model was trained and tested with the feature vector containing information on the sentiment value of each article gained by the Vader sentiment analysis tool. The accuracy on the training data was 95.86 % and on the training data 95.03 %. The biggest drop in accuracy was with the Vader sentiment analysis when compared with the control results as the difference between accuracy on the testing data was 1.89 %. The confusion matrix (figure 8) shows similar distribution between falsely predicted true (1 %) and false articles (9 %). The model correctly predicted 8534 articles and falsely predicted 446 articles.

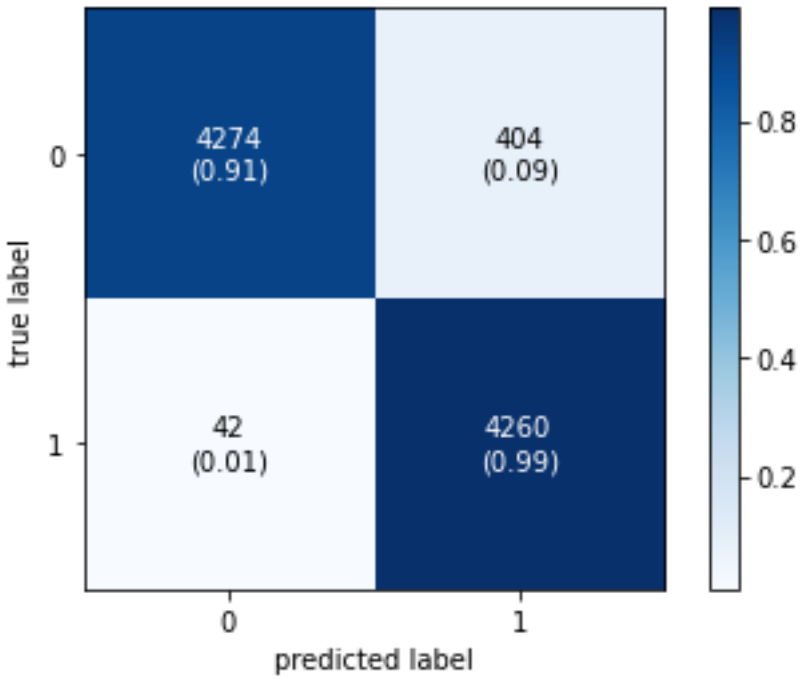


**Figure 8** Confusion matrix on Naive Bayes with sentiment information from Vader tool.

Sentiment information on the text data in the dataset does not seem to improve the performance of the Naïve Bayes model based on the results. There is a slight decline in accuracy when the TextBlob sentiment analysis results are added to the feature vector and a larger decline in accuracy when the Vader results are added to the feature vector.

## 5.2 Support Vector Machine results

The same steps were repeated for the SVM model. The accuracy was also calculated for both the training data and the testing data. And a confusion matrix was created for the testing results from the control model, model when TextBlob sentiment information added, and model when Vader sentiment information added.

The control model produced an accuracy of 99.95 % on the training data and 99.68 % on the testing data. These results were too good. The possible reason behind these results is the overfitting on the dataset. This means that the feature vector contains some feature or features that are a very high indicator of whether the article is real or fake. In the dataset all the true articles were from two subjects, which might mean that they contain similar

vocabulary, which the SVM model has been fitted to find. The resulting model will probably not generalize too well when it is used to classify a different dataset.

To fix this problem there are a few solutions: training and testing all the models on a new dataset, trying to find what the highly discriminating feature or features are and removing them, or implementing k-fold cross-validation to train and test the model. It was decided not to try and fix this problem as in the case of this thesis generalization is not as important, as the focus is to find whether adding sentiment information will improve the outcoming results of the model. Due to the goal, it does not matter if the control model accuracy is very high from the beginning, as an improvement or a deterioration to the accuracy can still be noticed. Additionally, time constraint was a factor, as it would take a lot of time to create a similarly sized dataset from scratch or to try and find the discriminating feature or features in the feature vector. And doing k-fold cross-validation needs a lot of processing power for the chosen dataset, which would decrease the efficiency of the program.

The confusion matrix from the results on the testing data of the control SVM model show that the model is slightly more likely to classify a fake article as true, than a true article as fake. This is consistent with the results of the Naïve Bayes model. The confusion matrix is presented in figure 9. The confusion matrix shows that the model correctly predicted 8951 articles and falsely predicted 29 articles.

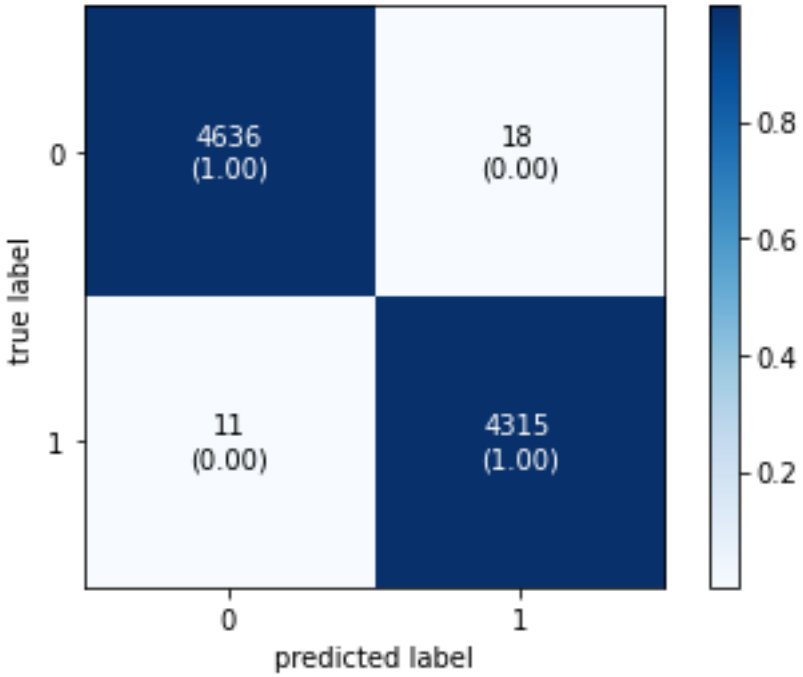


**Figure 9** Results of SVM model without sentiment information.

Next the same steps were repeated but the sentiment analysis results of TextBlob were gained from the raw text data and added to the feature vector gained from the TF-IDF vectorizer. The vector was then used to train and test the SVM model. The results were similarly high as in the control model. The model produced an accuracy of 99.96 % on the training data and 99.70 % on the testing data. This shows a very slight improvement at 0.02%. The confusion matrix (figure 10) shows a similar distribution between false positive and false negative results as all the other models. It appears that the model correctly predicted 8953 articles, which is 2 articles more than in the control model.

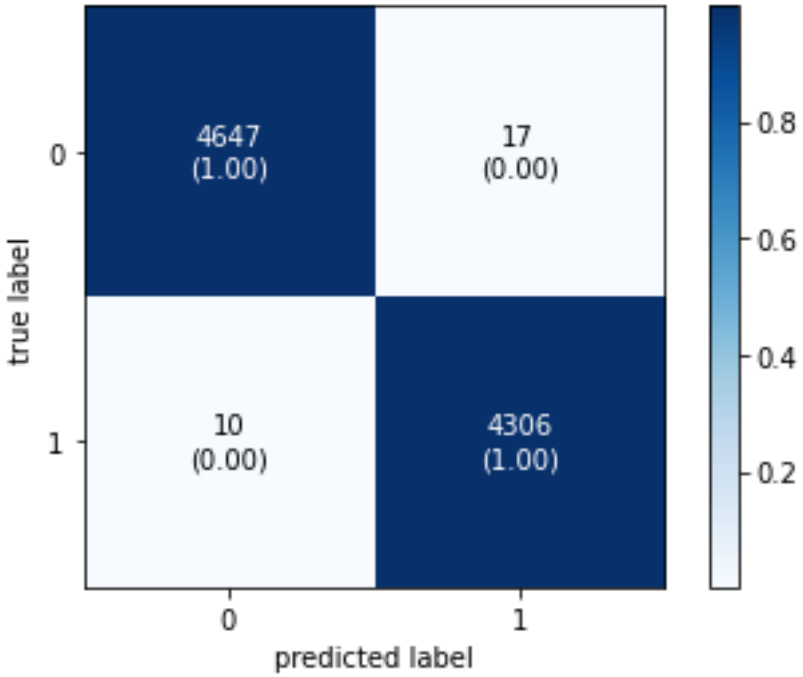


**Figure 10** Confusion matrix of SVM model predictions on testing data when TextBlob sentiment information is added to the feature vector.

Lastly, the SVM model was trained and tested with a feature vector that contained sentiment information gained by the Vader toolkit. The model's accuracy on the training data was 99.95 % and on the testing data 99.66 %. The accuracy of the model decreased by 0.02 %. The confusion matrix (figure 11) shows that the distribution of falsely predicted true and false articles is like all the other models. It appears that the model correctly predicted 8953 articles and falsely predicted 31 articles, which is 2 more than the control model.

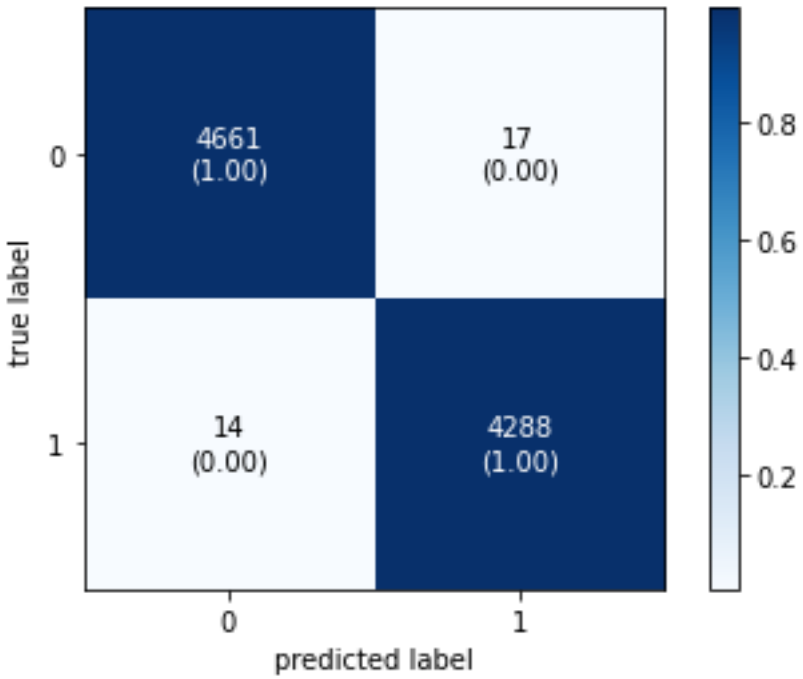


**Figure 11** Confusion matrix of SVM model when trained and tested with sentiment information gained by Vader.

It appears that the SVM model does not give significantly better predictions when sentiment information is added to the feature vector. There was a very slight improvement on accuracy with sentiment information by TextBlob, but the accuracy slightly decreased with sentiment information by Vader.

# 6 DISCUSSION

The results indicate that there is no improvement of the fake news detection models when sentiment information is added to the input. Especially for the Naïve Bayes model the addition of sentiment information appears to decrease the accuracy of the model. For the SVM model there was a very slight increase in accuracy when sentiment information was gained with the TextBlob toolkit, but a decrease in accuracy when sentiment information was gained with the Vader toolkit.

These results could be because of some incompatibility of the readily available sentiment analysis tools, TF-IDF vectorizer, and the Naïve Bayes and SVM models. As the TF-IDF vectorizer combined with the machine learning models produced good results in related work, the most probable issue is with the TextBlob and Vader sentiment analysis tools when combined with the machine learning models. The related work on the effects of sentiment information on machine learning models' accuracy indicated that sentiment information increases the accuracy of the models. It is important to note, that in the related work sentiment analysis was not done by using readily available sentiment analysis tools, which supports the hypothesis behind the possible reasons for the gained results.

Additionally, it is important to remember that the off-the-shelf sentiment analysis tools Vader and TextBlob are trained to perform sentiment analysis using tweets. Tweets can contain more sentiment information as there are limitations on the number of characters used and they usually try to gain the readers' attention, which means that there can be more sentiment information in a smaller amount of text, while articles are usually longer and try to contain more neutral language and thus this might have influenced the results of the sentiment analysis tools. Trying some different sentiment analysis methods could result in better results. The sentiment analysis can also be done from scratch, which might make the analysis more applicable for the desired purposes.

Another possible way to improve the results would be by trying to change the limits for when polarity is positive or negative. In this thesis the limit was set at under -0.05 for negative and over 0.05 for positive. Additionally, it is worth trying to find out whether adding the

subjectivity score from the TextBlob sentiment analysis tool to the input vector would have any impact on the accuracy of the model.

Another possible way to improve the results would be trying out different machine learning models. The decision tree model could be a good model to try, as it is possible to see what the features are on which the model is making its classifications based on. This would enable the possibility of removing very discriminating features, resulting in a more generalizable model. The SVM model appeared to be picking up on a very discriminating feature or features between the true and fake news data, which resulted in very good predictions on the dataset, but probably bad generalization ability. If the decision tree model were used, these highly discriminating features could have been removed from the dataset, thus leading to a better SVM model.

Another issue was the dataset used to train and test the model. The dataset was quite homogenous, as most of the true articles belonged to two subjects, which could indicate that the vocabulary inside the articles might also be similar. It appeared that the SVM model learned to classify the data inside the dataset used in this thesis too well, resulting in very good results on the testing data. While the homogeneity of the dataset should not have had an impact in the ability to distinguish how sentiment information affects the performance of the models, it might be good to conduct the same research again with a different dataset that has a larger variance between articles in the true and fake classes. This might enable larger differences between accuracy results of the machine learning models with sentiment and without sentiment, which might give a better way to estimate how much of an impact sentiment information had on the accuracy.

There needs to be some more research done on the effects of sentiment information on the accuracy of fake news detection models. Based on the results in this thesis adding sentiment information to the most popular models used for fake news detection does not have much of an impact on the accuracy, it might even slightly negatively impact the results. If sentiment information needs to be added to the method of detecting fake news, it appears that the TextBlob sentiment analysis tool is more compatible with both the Naïve Bayes and the Support Vector machine models. The hypothesis of sentiment information increasing the

accuracy was not correct for the models used in this thesis, when combined with the TextBlob and Vader sentiment analysis tools.

# 7 CONCLUSION

As the spread of fake news increases, there is a growing demand for ways to accurately detect fake news. Due to this demand, there is a lot of research done on implementing different machine learning models to detect fake news. While there is a lot of research on different machine learning models and their classification accuracy for fake news, there is not a lot of research done on the effects of sentiment information on the accuracy of the models.

The goal of this thesis was to find out what impact sentiment information has on the performance of two of the most popular models used in fake news detection. These models are the Naïve Bayes and the Support Vector Machine model. The sentiment analysis was performed by implementing the TextBlob and Vader sentiment analysis tools. The results of these tools were added to the feature vectors that were created by implementing the TF-IDF vectorizer. These vectors were then used to train and test the Naïve Bayes and SVM models.

The results indicated that for the Naïve Bayes model sentiment information caused a slight decrease of classification accuracy. This could mean that the Naïve Bayes model is not compatible with the TextBlob and Vader tools. The decrease of accuracy was larger with sentiment analysis done by Vader.

The results for the SVM model did not show any significant difference between classification accuracy when sentiment information was added to the input. The SVM model had a very slight increase in accuracy when combined with the TextBlob sentiment analysis tool and a slight decrease in accuracy when combined with the Vader tool. This would indicate that SVM is slightly more compatible with the TextBlob tool, than with the Vader tool.

The hypothesis of sentiment information increasing the accuracy of the fake news detection models was false with the models in this thesis combined with the Vader and TextBlob sentiment results. While the hypothesis was false in this thesis, there is still reason to further

research the effects of sentiment analysis on the fake news detection models, when sentiment analysis is done with some different methods. Additionally, the effects of the sentiment analysis tools used in this thesis can be further researched with different machine learning models, as the models combined with the sentiment information still produced good classification accuracy on the testing data.

# SOURCES


1. Anoop, K., Deepak, P., Lajish V, L., 2020. Emotion cognizance improves health fake news identification, in: Proceedings of the 24th Symposium on International Database Engineering & Applications. Presented at the IDEAS 2020: 24th International Database Engineering & Applications Symposium, ACM, Seoul Republic of Korea, pp. 1–10. https://doi.org/10.1145/3410566.3410595
2. Agudelo, G.E.R., Parra, O.J.S., Velandia, J.B., 2018. Raising a Model for Fake News Detection Using Machine Learning in Python, in: Al-Sharhan, S.A., Simintiras, A.C., Dwivedi, Y.K., Janssen, M., Mäntymäki, M., Tahat, L., Moughrabi, I., Ali, T.M., Rana, N.P. (Eds.), Challenges and Opportunities in the Digital Era, Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 596–604. https://doi.org/10.1007/978-3-030-02131-3_52
3. Bali, A.P.S., Fernandes, M., Choubey, S., Goel, M., 2019. Comparative Performance of Machine Learning Algorithms for Fake News Detection, in: Singh, M., Gupta, P.K., Tyagi, V., Flusser, J., Ören, T., Kashyap, R. (Eds.), Advances in Computing and Data Sciences, Communications in Computer and Information Science. Springer, Singapore, pp. 420–430. https://doi.org/10.1007/978-981-13-9942-8_40
4. Conroy, N.K., Rubin, V.L., Chen, Y., 2015. Automatic deception detection: Methods for finding fake news. Proc. Assoc. Inf. Sci. Technol. 52, 1–4. https://doi.org/10.1002/pra2.2015.145052010082
5. Granik, M., Mesyura, V., 2017. Fake news detection using naive Bayes classifier, in: 2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON). Presented at the 2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON), pp. 900–903. https://doi.org/10.1109/UKRCON.2017.8100379
6. Guo, C., Cao, J., Zhang, X., Shu, K., Yu, M., 2019. Exploiting Emotions for Fake News Detection on Social Media.
7. Guo, C., Cao, J., Zhang, X., Shu, K., Liu, H., 2019a. DEAN: Learning Dual Emotion for Fake News Detection on Social Media [WWW Document].

GroundAI. URL https://www.groundai.com/project/dean-learning-dual-emotion-for-fake-news-detection-on-social-media/2 (accessed 12.13.20).

8. Hussain, M.G., Hasan, M.R., Rahman, M., Protim, J., Hasan, S., 2020. Detection of Bangla Fake News using MNB and SVM Classifier.
9. Kesarwani, A., Chauhan, S.S., Nair, A.R., Verma, G., 2020. Supervised Machine Learning Algorithms for Fake News Detection, in: Hura, G.S., Singh, A.K., Siong Hoe, L. (Eds.), Advances in Communication and Computational Technology, Lecture Notes in Electrical Engineering. Springer, Singapore, pp. 767–778. https://doi.org/10.1007/978-981-15-5341-7_58
10. Kotsiantis, S., 2007. Supervised Machine Learning: A Review of Classification Techniques. Inform. Ljubl. 31.
11. Lewis, D.D., 1998. Naive (Bayes) at forty: The independence assumption in information retrieval, in: Nédellec, C., Rouveirol, C. (Eds.), Machine Learning: ECML-98, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 4–15. https://doi.org/10.1007/BFb0026666
12. Poddar, K., Amali D., G.B., Umadevi, K.S., 2019. Comparison of Various Machine Learning Models for Accurate Detection of Fake News, in: 2019 Innovations in Power and Advanced Computing Technologies (i-PACT). Presented at the 2019 Innovations in Power and Advanced Computing Technologies (i-PACT), pp. 1–5. https://doi.org/10.1109/i-PACT44901.2019.8960044
13. Tijare, P., 2019. A Study on Fake News Detection Using Naïve Bayes, SVM, Neural Networks and LSTM.
14. Vijayaraghavan, S., Wang, Y., Guo, Z., Voong, J., Xu, W., Nasseri, A., Cai, J., Li, L., Vuong, K., Wadhwa, E., 2020. Fake News Detection with Different Models.

# BIBLIOGRAPHY


1. Bisaillon, C., 2020. Fake and real news dataset [WWW Document]. Kaggle. URL https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset (accessed 11.28.2020).
2. George, J.A., 2020. Fake News Detection using NLP techniques [WWW Document]. Medium. URL https://medium.com/analytics-vidhya/fake-news-detection-using-nlp-techniques-c2dc4be05f99 (accessed 12.14.20).
3. Pease, C., 2018. Machine Learning tackles the Fake News problem [WWW Document]. Medium. URL https://towardsdatascience.com/machine-learning-tackles-the-fake-news-problem-c3fa75549e52 (accessed 10.6.20).
4. Support Vector Machines: A Simple Explanation, n.d.. KDnuggets. URL https://www.kdnuggets.com/support-vector-machines-a-simple-explanation.html/ (accessed 10.6.20).
5. Using Pre-trained VADER Models for NLTK Sentiment Analysis [WWW Document], n.d. URL https://www.codeproject.com/Articles/5269445/Using-Pre-trained-VADER-Models-for-NLTK-Sentiment (accessed 10.6.20)
6. Prabhakaran, S., 2018. Lemmatization Approaches with Examples in Python. ML+. URL https://www.machinelearningplus.com/nlp/lemmatization-examples-python/ (accessed 12.14.20).
7. Vázquez, F., 2020. Detecting Fake News With and Without Code [WWW Document]. Medium. URL https://towardsdatascience.com/detecting-fake-news-with-and-without-code-dd330ed449d9 (accessed 12.14.20).

## ATTACHMENT 1. Program code

```
import nltk
import string
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import numpy as np
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from nltk.corpus import wordnet
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from textblob import TextBlob
from scipy.sparse import hstack
from nltk.sentiment.vader import SentimentIntensityAnalyzer

#load data
fake = pd.read_csv('archive/Fake.csv')
real = pd.read_csv('archive/True.csv')

#create labels

real['label']=1
fake['label']=0

#combine data into one

data = pd.concat([real,fake])
data.head
data.shape

#shuffle data
data = data.sample(frac=1)
```

**(continues)**

**ATTACHMENT 1. (Continues)**

```
#lets perform basic analysis of data
#are there any empty values in the data set
#source rows 53-66: George,2020.
data.isnull().sum()
#no null values

#is data balanced
sns.countplot(data['label'])
#data appears to be balanced

#data distribution by subject and label
data['subject'].value_counts()
plt.figure(figsize = (10,10))
sns.countplot(data['subject'], color='blue')

plot=sns.countplot(x='label', hue='subject', data=data)
plot.set_xticklabels(plot.get_xticklabels(), rotation=90)


#lets clean up data and pre-process it
#lets remove all unnecessary values in the data set i.e. title, subject, dat
e
data = data.drop(['title', 'subject', 'date'], axis=1)
#source rows 73-88: Vázquez, 2020.
#text to lowercase
data['text'] = data['text'].apply(lambda x: x.lower())

#remove punctuation
def remove_punct(text):
    word_list = [char for char in text if char not in string.punctuation]
    clean = ''.join(word_list)
    return clean

data['text'] = data['text'].apply(remove_punct)

#remove stopwords
nltk.download('stopwords')
stop = stopwords.words('english')

data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.spl
it() if word not in (stop)]))
```

**(continues)**

**ATTACHMENT 1. (Continues)**

```
#tokenization
nltk.download('punkt')
data['text'] = data['text'].apply(lambda x: word_tokenize(x))

#lemmatization
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

#source for rows 101-122: Prabhakaran, 2018.
def pos_tagger(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatized_text(text):
    tagged = nltk.pos_tag(text)
    returnable = list(map(lambda x: (x[0], pos_tagger(x[1])), tagged))
    lemmatized_sentence = []
    for word, tag in returnable:
        if tag is None:
            lemmatized_sentence.append(word)
        else:
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return lemmatized_sentence

data['text'] = data['text'].apply(lambda x: lemmatized_text(x))

#split the data into training (80%) and testing (20%) data
X = data['text']
Y = data['label']

x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2, ra
ndom_state = 5)
```

**(continues)**

## ATTACHMENT 1. (Continues)

```
#TF-IDF
def tokenize_off(text):
    return text

vectorizer = TfidfVectorizer(tokenizer=tokenize_off, lowercase = False)

tfidf_train = vectorizer.fit_transform(x_train)

tfidf_test = vectorizer.transform(x_test)

#run control Naive Bayes code
NBclassifier = MultinomialNB()
NBclassifier.fit(tfidf_train, y_train)

predictionMNB = NBclassifier.predict(tfidf_test)

print(NBclassifier.score(tfidf_train, y_train))
#accuracy: 0.97372

print(NBclassifier.score(tfidf_test, y_test))
#accuracy 0.96915

#run control SVM code
SVM = SVC(kernel='linear')
SVM.fit(tfidf_train, y_train)

predictionSVM = SVM.predict(tfidf_test)

print(SVM.score(tfidf_train, y_train))
#precision: 0.99947
print(SVM.score(tfidf_test, y_test))
#precission: 0.99677
```

**(continues)**

## ATTACHMENT 1. Continues

```
#evaluate performance with accuracy and confusion matrix
#Multinomial Naive Bayes
#source rows 172-191: George, 2020.
score = metrics.accuracy_score(y_test, predictionMNB)
print("accuracy:   %0.3f" % (score*100))
cm = metrics.confusion_matrix(y_test, predictionMNB, labels=[0,1])

fig, ax = plot_confusion_matrix(conf_mat=confusion_matrix(y_test, prediction
MNB),
                                show_absolute=True,
                                show_normed=True,
                                colorbar=True)
plt.show()
#Support Vector Machine
score = metrics.accuracy_score(y_test, predictionSVM)
print("accuracy:   %0.3f" % (score*100))
cm = metrics.confusion_matrix(y_test, predictionSVM, labels=[0,1])

fig, ax = plot_confusion_matrix(conf_mat=confusion_matrix(y_test, prediction
SVM),
                                show_absolute=True,
                                show_normed=True,
                                colorbar=True)
plt.show()
#TEXTBLOB
#for the sentiment analysis we need unprocessed data
data_sentiment = pd.concat([real,fake])
data_sentiment = data_sentiment.sample(frac=1)
data_sentiment = data_sentiment.drop(['title', 'subject', 'date'], axis=1)
#lets perform sentiment analysis with Text Blob
data_sentiment['textblob'] = data_sentiment['text'].apply(lambda x: TextBlob
(x).polarity)

def sentiment_transformation(num):
    if num <=-0.05:
        return 0
    elif num >=0.05:
        return 2
    else:
        return 1
data_sentiment['textblob'] = data_sentiment['textblob'].apply(lambda x: sent
iment_transformation(x))
```

**(continues)**

## ATTACHMENT 1. Continues

```
#preprocess text data
#source rows 213-220: Vázquez, 2020.
#text to lowercase
data_sentiment['text'] = data_sentiment['text'].apply(lambda x: x.lower())

#remove punctuation
data_sentiment['text'] = data_sentiment['text'].apply(remove_punct)

#remove stopwords
data_sentiment['text'] = data_sentiment['text'].apply(lambda x: ' '.join([wo
rd for word in x.split() if word not in (stop)]))

#tokenization
data_sentiment['text'] = data_sentiment['text'].apply(lambda x: word_tokeniz
e(x))

#lemmatization
data_sentiment['text'] = data_sentiment['text'].apply(lambda x: lemmatized_t
ext(x))

#split data
XTB = np.asarray(data_sentiment[['text', 'textblob']])
YTB = data_sentiment['label']
x_trainTB, x_testTB, y_trainTB, y_testTB = train_test_split(XTB,YTB, test_si
ze = 0.2, random_state = 5)

#TF-IDF
XTB_train = x_trainTB[:,0]
tfidf_trainTB = vectorizer.fit_transform(XTB_train)
XTB_test = x_testTB[:,0]
tfidf_testTB = vectorizer.transform(XTB_test)

#append the textblob results to tf idf matrix
XTB_train = x_trainTB[:,1]
tfidf_trainTB = hstack((tfidf_trainTB, np.array(XTB_train,dtype=float)[:,Non
e]))
tfidf_trainTB.shape
#same for testing matrix
XTB_test = x_testTB[:,1]
tfidf_testTB = hstack((tfidf_testTB, np.array(XTB_test, dtype=float)[:,None]
))
tfidf_testTB.shape
```

**(continues)**

## ATTACHMENT 1. Continues

```
#lets run Naive bayes classifier w TextBlob
NBclassifier.fit(tfidf_trainTB, y_trainTB)

predictionMNBTB = NBclassifier.predict(tfidf_testTB)

print(NBclassifier.score(tfidf_trainTB, y_trainTB))
#precission: 0.974497
print(NBclassifier.score(tfidf_testTB, y_testTB))
#precision: 0.968597

#lets run SVM classifier w TextBlob
SVM.fit(tfidf_trainTB, y_trainTB)

predictionSVMTB = SVM.predict(tfidf_testTB)

print(SVM.score(tfidf_trainTB, y_trainTB))
#precision: 0.99961
print(SVM.score(tfidf_testTB, y_testTB))
#precision: 0.99699

#VADER
#for the sentiment analysis we need unprocessed data
data_sentimentV = pd.concat([real,fake])
data_sentimentV = data_sentimentV.sample(frac=1)
data_sentimentV = data_sentimentV.drop(['title', 'subject', 'date'], axis=1)

#lets perform sentiment analysis with Vader
nltk.download('vader_lexicon')
vader = SentimentIntensityAnalyzer()
data_sentimentV['sentiment'] = data_sentimentV['text'].apply(lambda x: vader
.polarity_scores(x)['compound'])
data_sentimentV['sentiment'] = data_sentimentV['sentiment'].apply(lambda x:
sentiment_transformation(x))
#preprocess text data
#source rows 307-312: Vázquez, 2020.
#text to lowercase
data_sentimentV['text'] = data_sentimentV['text'].apply(lambda x: x.lower())
#remove punctuation
data_sentimentV['text'] = data_sentimentV['text'].apply(remove_punct)
#remove stopwords
data_sentimentV['text'] = data_sentimentV['text'].apply(lambda x: ' '.join([
word for word in x.split() if word not in (stop)]))
```

**(continues)**

## ATTACHMENT 1. Continues

```
#tokenization
data_sentimentV['text'] = data_sentimentV['text'].apply(lambda x: word_token
ize(x))

#lemmatization
data_sentimentV['text'] = data_sentimentV['text'].apply(lambda x: lemmatized
_text(x))

#split data
XV = np.asarray(data_sentimentV[['text', 'sentiment']])
YV = data_sentimentV['label']
x_trainV, x_testV, y_trainV, y_testV = train_test_split(XV,YV, test_size = 0
.2, random_state = 5)

#TF-IDF
XV_train = x_trainV[:,0]
tfidf_trainV = vectorizer.fit_transform(XV_train)
XV_test = x_testV[:,0]
tfidf_testV = vectorizer.transform(XV_test)

#append the vader results to tf idf matrix
XV_train = x_trainV[:,1]
tfidf_trainV = hstack((tfidf_trainV, np.array(XV_train,dtype=float)[:,None])
)
tfidf_trainV.shape

#same for testing matrix
XV_test = x_testV[:,1]
tfidf_testV = hstack((tfidf_testV, np.array(XV_test, dtype=float)[:,None]))
tfidf_testV.shape

#lets run Naive bayes classifier w Vader
NBclassifier.fit(tfidf_trainV, y_trainV)

predictionMNBV = NBclassifier.predict(tfidf_testV)

print(NBclassifier.score(tfidf_trainV, y_trainV))
#precission:0.95860
print(NBclassifier.score(tfidf_testV, y_testV))
#precision: 0.95033
```

**(continues)**

## ATTACHMENT 1. Continues

```
#lets run SVM classifier w Vader
SVM.fit(tfidf_trainV, y_trainV)

predictionSVMV = SVM.predict(tfidf_testV)

print(SVM.score(tfidf_trainV, y_trainV))
#precission: 0.999499
print(SVM.score(tfidf_testV, y_testV))
#precission: 0.99655
```