Multiclass Supervised Text Classification

Dhairav Chhatbar

City University of New York School of Professional Studies

Abstract

Text classification or text categorization is the activity of automatically assigning/labeling documents based on the content within each document. This study goes into the various steps involved in text classification and implements 3 models (Support Vector Machines, Naïve Bayes, and Random Forest) to classify text of the 20 Newsgroups dataset, and then discusses which model is more appropriate over the other. The general steps are to determine a pre-processing strategy prior to running the models. While running the models there are several hyper-parameters per model to tune. The general findings are that after the pre-processing pipeline of cleaning, vectorization, and TF-IDF transformation, all 3 models produced similar results, but the Naïve Bayes model had a distinct advantage of execution time efficiency.

Keywords

Classification, Document Term Matrix, TF-IDF, Vectorizer, Support Vector Machines, Naïve Bayes, Random Forest

Introduction & Literature review

Text classification or text categorization is the activity of automatically assigning/labeling documents which contain relevant content to its relevant predefined class. This process of tagging a document to its appropriate category is typically very manual and time intensive. While there have been many advances in searching by keyword, the limitation of searching is that it does not discriminate by context [1]. This study looks to the industry problem of classifying documents into their respective categories so that the amount of time spent to open review a document and tag it for storage is minimized.

There have been many studies and research done on this topic and the research suggests that there is no one single strategy approach to tackling text classification. Kamran Kowsari et al go through an extensive array of different techniques for feature extraction, dimensionality reductions, classifier models, and other techniques in their paper Text Classification Algorithms: A Survey [2]. They also cover the limitations of each technique and classifier to express that the strategy that is going to be defined for text classification is dependent on understanding the content, and finding the correct mix of techniques and models can be a challenge. Mehdi Allahyari et al in their paper A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques [3] arrive at the very same conclusion.

This study will be applying well established techniques to our dataset of interest (20 Newsgroups), and of the various models out in the field, we will limit our models to those that we have academically learned in this curriculum. Korde, Mahender [4] in their paper Text Classification and Classifiers: A Survey found that for their dataset Support Vector Machines (SVM) performs well in general text classification tasks. In our study the we do not want to confirm these findings but rather determine the best possible strategy for our dataset and compare with key performance metrics.

Methodology

The general methodology of text classification is well established. We will follow the general structure established by Deepshikha Kalita[1], with the exception that we will follow a supervised approach. The general established methodology of text classification is to understand the dataset, pre-processing of the data, establish a set of features, run a set of classifier models, and then evaluate the models. While there are numerous combinations and permutations of the exact strategy to employ, this will largely depend on the dataset, available hardware, and goals of the objective. The exact strategy employed for our goals are noted in Figure 1

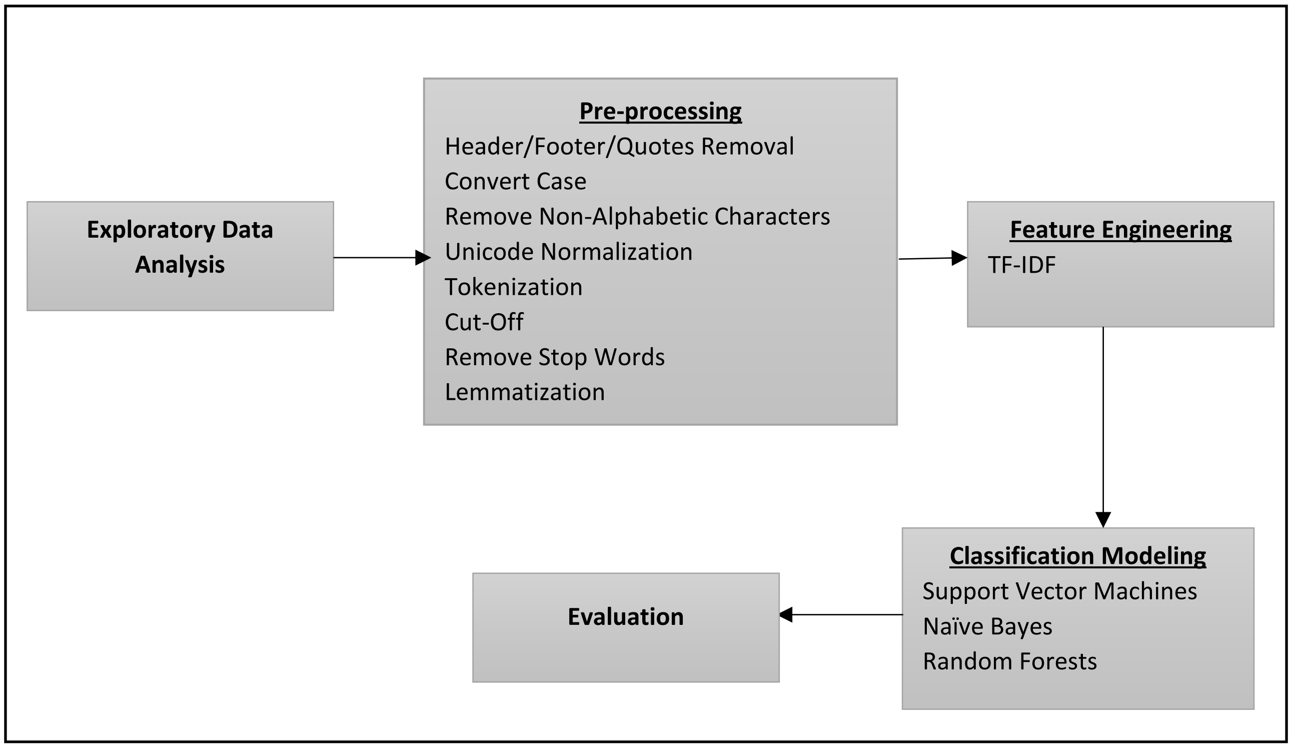


Figure 1: Classification Methodology Steps

Data Set Exploration

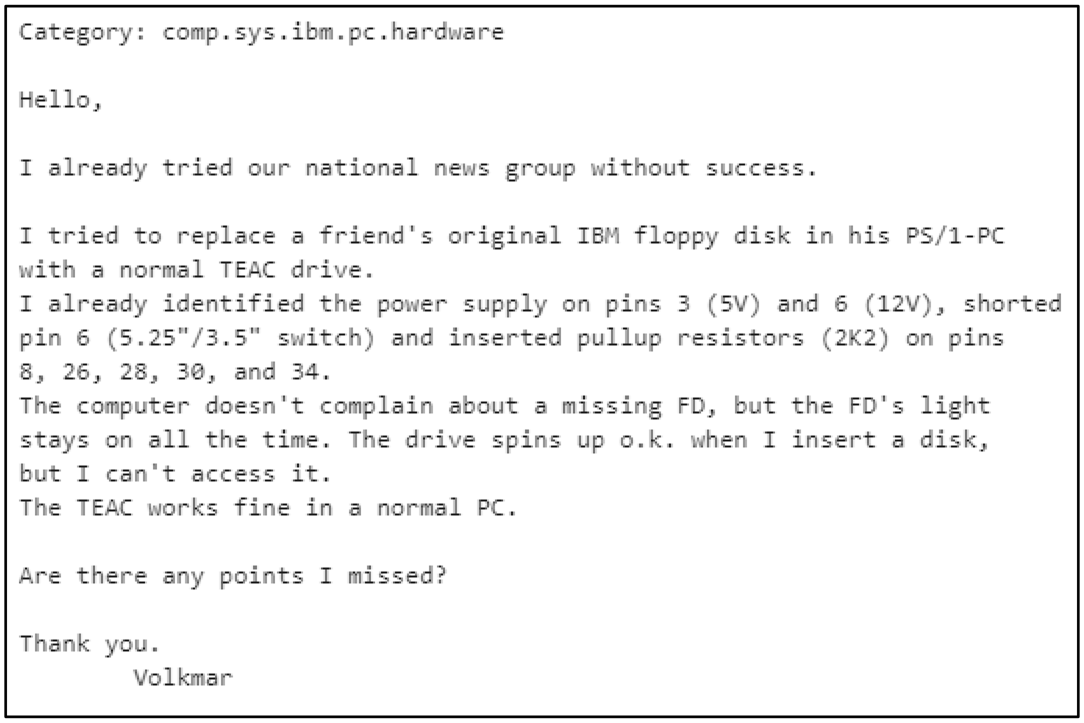
The dataset that we will use to build text classification models is the 20newsgroups data set that is openly available on http://qwone.com/~jason/20Newsgroups/

The dataset contains approximately 18,800 newsgroups documents organized across 20 different categories where each category represents a different topic of discussion the categories are in Figure 1. While there are 20 topics within the dataset, there are similarities within given topics, these similar topics are shown in Figure 2. For example, we can expect similar discussion points of interest within the topics talk.religion.misc and alt.atheism, and there would be similar discussions within the topics comp.sys.ibm.pc.hardware and comp.sys.mac.hardware.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Figure 2: Topics of Dataset

An example of a post can be seen in Figure 3 from the pc.hardware category. Note the formatting of the text is not restrictive of any characters. As such within our dataset the character content of the text is not limited to alphabets (upper and lower case), numbers nor special characters.



```
Category: comp.sys.ibm.pc.hardware

Hello,

I already tried our national news group without success.

I tried to replace a friend's original IBM floppy disk in his PS/1-PC
with a normal TEAC drive.
I already identified the power supply on pins 3 (5V) and 6 (12V), shorted
pin 6 (5.25"/3.5" switch) and inserted pullup resistors (2K2) on pins
8, 26, 28, 30, and 34.
The computer doesn't complain about a missing FD, but the FD's light
stays on all the time. The drive spins up o.k. when I insert a disk,
but I can't access it.
The TEAC works fine in a normal PC.

Are there any points I missed?

Thank you.
        Volkmar
```

Figure 3: Post Example

For the given 18.8k count of documents the distribution per topic is fairly even across all topics with approximately 1000 documents per topic except for talk.religion.misc, alt.atheism, talk.politics.misc, and talk.politics.guns. See Figure 4

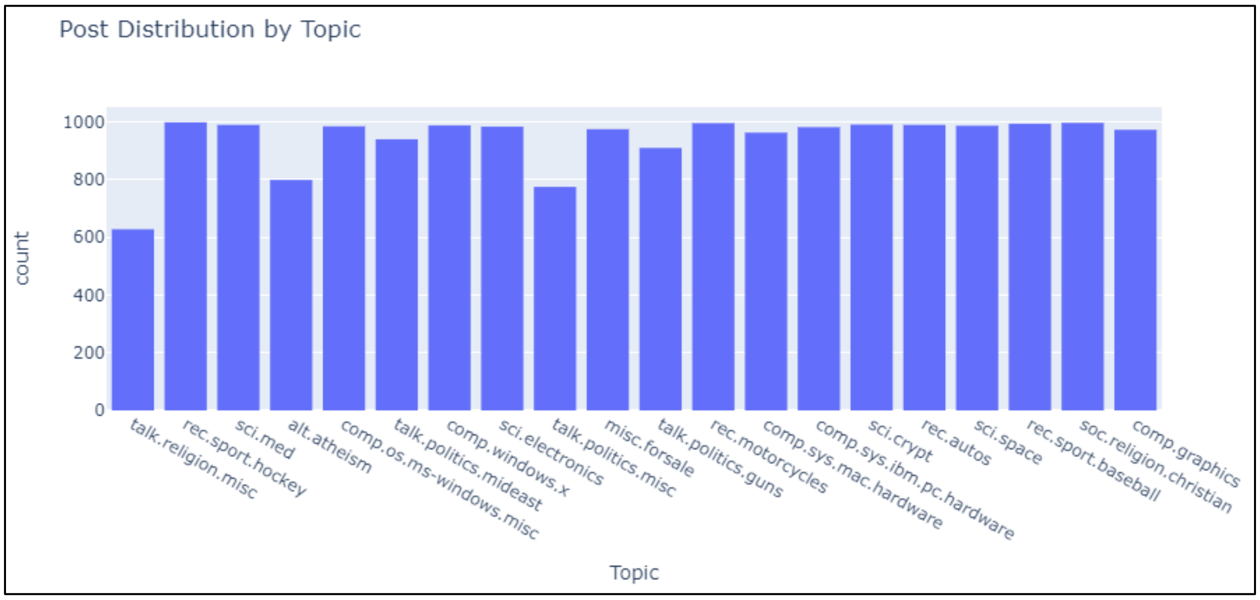


Figure 4: Topic Count Distribution

Preprocessing

As with non-language datasets we will perform the pre-processing and various cleaning methods to prior to building the features of the dataset. There are numerous strategies and methods of pre-processing textual data such as but not limited to lower casing, removal of stop words/frequent words/non-alphabetic words, stepping, lemmatization, spell corrections, emoji conversions, removal of URLs/HTML tags, etc. The application of a certain method or a set of methods (the pre-processing strategy) will largely depend on the nature of the text application at hand. In general, “cleaning” the corpus prior to running classifier models against it help reduce noise by removing “bits” of text that would otherwise be irrelevant in the grand scheme of identifying a document to the correct topic.

HaCohen-Kerner, Miller[5], Yigal in their study established that for a given dataset there exists at least one preprocessing method that can help improve text classification using a bag-of-words representations.

For our dataset the following pre-processing was done in the given order:

a) <u>Header/Footer/Quotes Removal</u>
Since our corpus of text is form online newsgroups, many posts include headers, footers, and in cases of replies to previous posts, the previous post quoted. The quotes would introduce duplicates into the training set and headers may include the name of the forum (target topic) a post is created in. For these reasons it is best to remove them from the corpus

b) <u>Convert Case & Remove Non-Alphabetic Characters</u>
All text is converted to lower case and non-alphabetic characters (numbers and special characters) are discarded

c) <u>Unicode Normalization</u>
Text characters can be represented in multiple forms in Unicode. For example, the letter “A” can be represented by U+0041 or by U+FF21. “Ä” can be represented by U+00C4 or combination of "A" (U+0041) + "¨" (U+0308). Unicode normalization decomposes these and then composes such characters to ensure that different strings that represent the same character have the same binary value after normalization. The `CountVectorizer` function of the `sklearn` package utilizes KFKD Normalization

d) <u>Tokenization</u>
Tokenization is the process of taking string and creating a list of words, where each word is called a token. Through the process some characters such as punctuation are lost. Figure 4 gives a visual indication of the Tokenization output. The output itself is go through further pre-processing

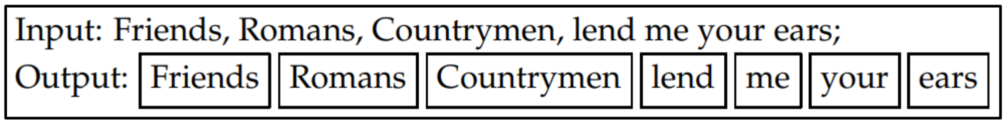


Figure 5: Tokenization [6]

e) <u>Cut-Off</u>

The goal of establishing a "cut-off" is to remove tokens that have occurred in less than a certain number of documents. This is to reduce noise and in our example a cut-off of 10 is established. This will remove tokens that occurred in less than 10 documents.

f) <u>Remove Stop Words</u>

Stop words are a subset of conjunctions, prepositions, and/or particle words. Examples are "as" "a", "is", "then", "the", and many others. Such which lend very limited mean to the context of document classification as they are generally observed across all topics, this is because stopwords are used to construct a sentence to give a sentence context, but do not contain any information about the sentence. The method of stop word removal was to utilize a set of pre-defined words list to create a custom list. The pre-defined stop words sets were from the `sklearn.feature_extraction` and `nltk.corpus` packages, combined with 4 ("don't", "like", "can't", "huh") custom words not found in the pre-defined packages.

g) <u>Lemmatization</u>

In the natural language a word will take different grammatic form, but contextually each of the forms holds the same meaning. For example, the words "change", "changing", "changes", and "changed" hold roughly the same meaning. Normalizing occurrences to a single word "change" would reduce redundant features. Generally, there are two approaches to this normalization, Stemming or Lemmatization.

Normalization via Stemming is the process reducing words to their root and discarding the suffix. In the above example the root word is considered "chang" and the suffixs are considered "e", "ed", "ing", "s". Normalization via Lemmatization is reducing words by use of its vocabulary and position in a sentence. The normalized word is referred to a lemma and is often more understandable than the same stemmed word. For example, the words the words "change", "changing", "changes", and "changed" will reduce to the lemma "change". The is more interpretable than the stemmed word "chang".

This pattern-based approach of stemming is fast and simple but is equivalent of using brute force. In comparison Lemmatization is more resource costly, but will generally produce more readable outcomes. Using Stemming over Lemmatization or vice versa has a precision/recall tradeoff, such that Stemming will increase recall at the expense of precision while Lemmatization will increase recall without hurting precision [6].

The above pre-processing steps are done and then our entire text corpus is turned into a Document Term Matrix (DTM) such that for the entirety of our tokens are represented by columns on the matrix and each post is represented as rows in the matrix. The columns are essentially features of the dataset and each row an observation. The DTM holds the term frequency per document per term such that:

$$\begin{bmatrix} & T_1 & T_2 & T_3 & T_4 & \cdots & T_n \\ P_1 & f_{11} & f_{21} & f_{31} & f_{41} & \cdots & f_{n1} \\ P_2 & f_{12} & f_{22} & f_{32} & f_{42} & \cdots & f_{n2} \\ P_3 & f_{13} & f_{23} & f_{33} & f_{43} & \cdots & f_{n3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ P_n & f_{1n} & f_{2n} & f_{3n} & f_{4n} & \cdots & f_{nn} \end{bmatrix} \qquad \begin{bmatrix} & Term_1 & Term_2 & Term_3 & \cdots & Term_n \\ Topic_1 & 0 & 2 & 3 & \cdots & 0 \\ Topic_2 & 4 & 0 & 1 & \cdots & 0 \\ Topic_3 & 0 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ Topic_n & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$$

Where P_i is a post, T_i are terms (words) that are present in the entire training dataset and f_{nn} is the frequency of term T_i in the post P_i.

Prior to running pre-processing steps our training dataset's Document Term Matrix (DTM) has the dimensions of 11314x120609, post pre-preprocessing this is reduced to 11314x8093. The pre-processing steps effectively reduced the number of features by 174.847%%. Figure 6 shows a wordcloud of the most frequent terms per topic.

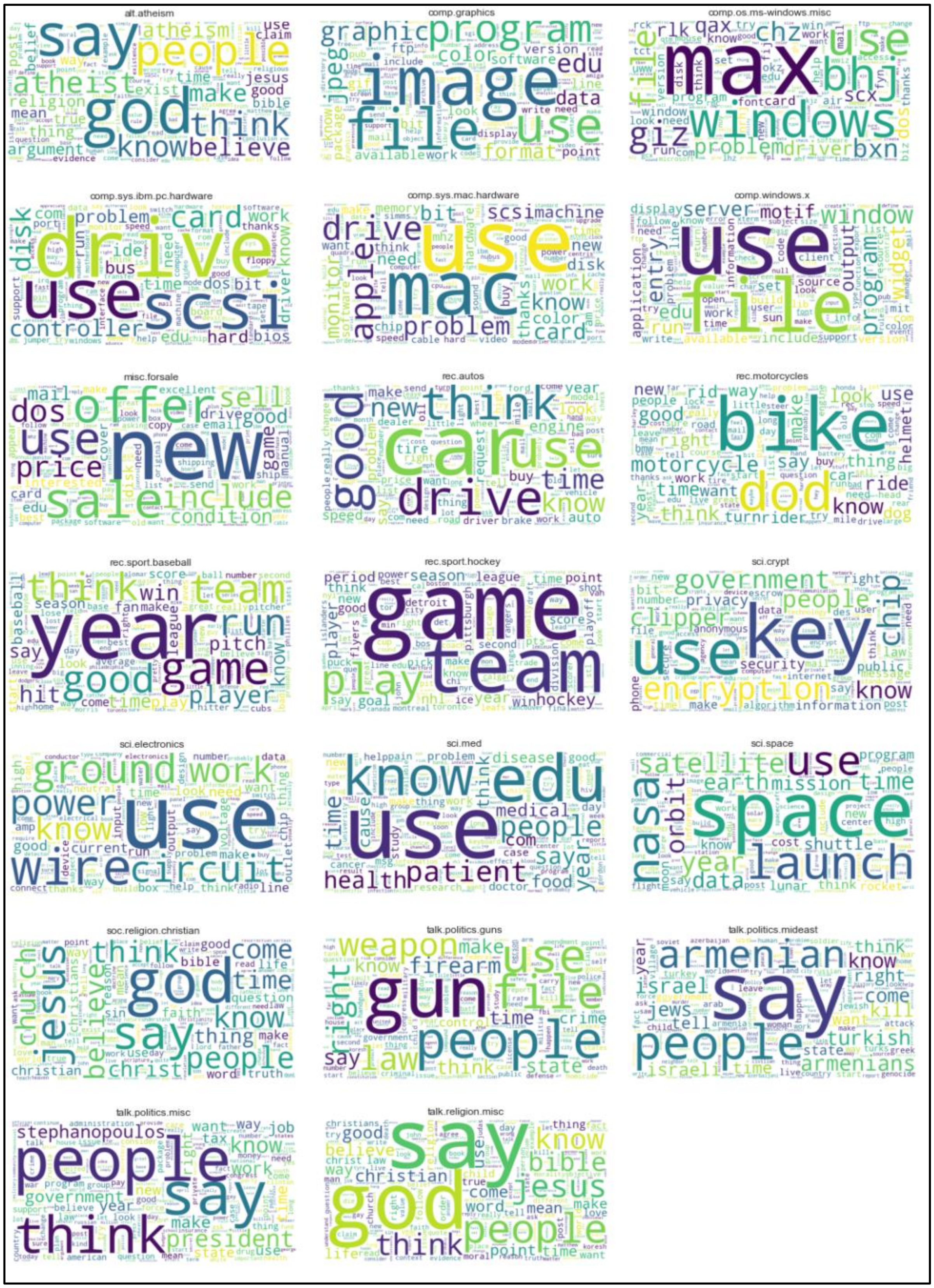


Figure 6: Wordcloud based on Document Term Matrix

Feature Engineering

At this juncture our original corpus has been transformed into a bag-of-words model via the various pre-processing steps. This can now be passed to various models for classifications. There are some additional improvements. The features created thus far via the pre-processing pipeline do not account for some nuances which can be further optimized. In particular, post length and term frequency across the corpus.

The vast majority (approx. 17.5k) of the posts in our dataset have up to 500 words, and the rest longer into the thousands. Larger posts will tend to have more term counts of a given term while a shorter post would have the same term but less frequently. It would be beneficial to normalize the term counts irrespective of the length of the post.

Another issue is that terms that appear frequently across all topics may not be very useful as they would have information to uniquely identify a topic.

To make these additional tweaks we transform our Document Term Matrix to a Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF is a measure to set the importance of a given term (t) in a post (d) given how many times t occurs in d and across the entire corpus (D) such that:

$$TFIDF(t,d) = TF(t,d) * IDF(t)$$

$$where$$

$$TF(t,d) = \frac{freq(t,d)}{\sum_i^n freq(t_i,d)} \ and \ IDF(t) = \log\left(\frac{D}{count(t)}\right)$$

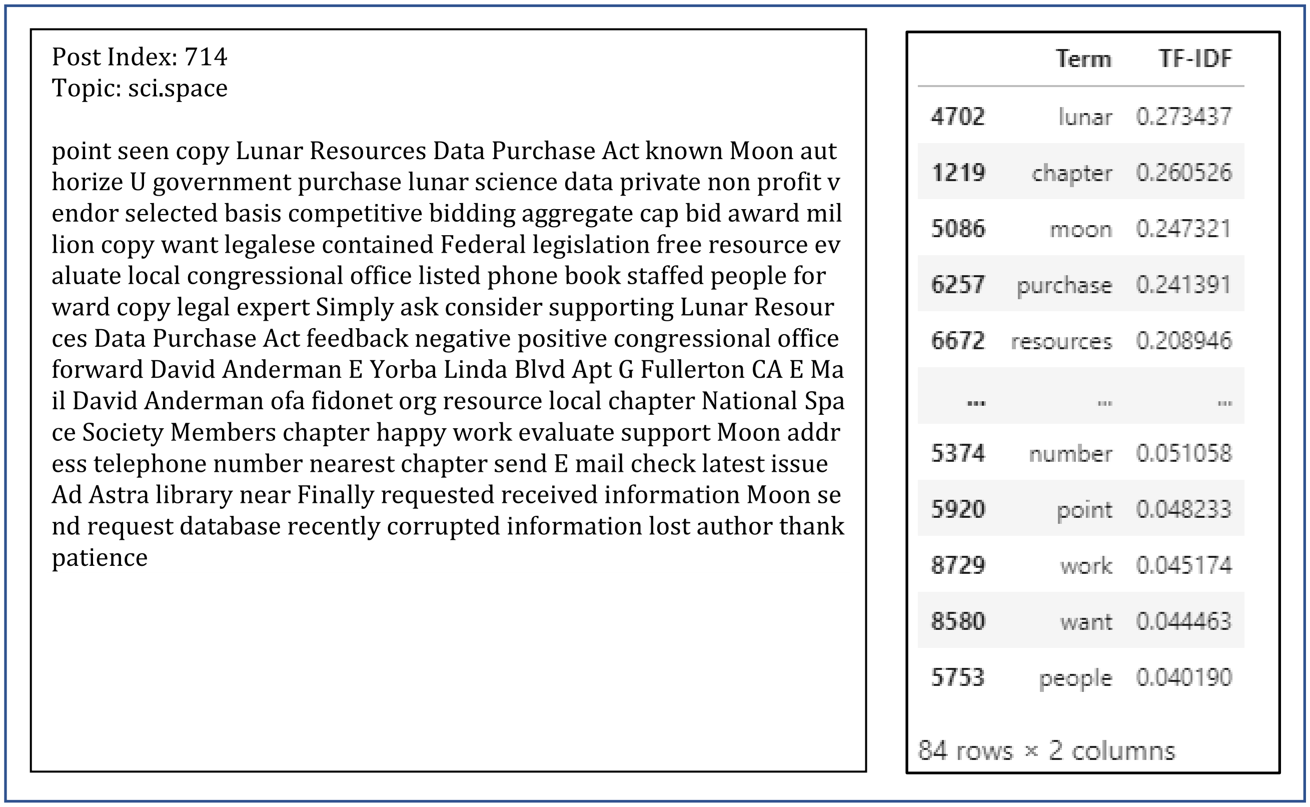
$$such \ that$$

$$freq(t,d) \ is \ number \ of \ occurances \ of \ term \ t \ in \ document \ d$$

$$count(t) \ is \ the \ number \ of \ documents \ in \ which \ term \ t \ occurs \ in$$

The resulting behavior of the above is that a term is given a high score if it appears frequently in a given topic. This is because for that specific topic this term better describes the topic than other terms in the topic. However, if this same term also occurs frequently across many other posts and topics then it’s score is penalized because a term occurring frequently everywhere holds less classification information to identify a topic. A term that occurs in 100% of the posts will have a TF-IDF score of 0 as it cannot uniquely identify any post nor topic.

See Figure 7 of a sample processed post in the sci.space topic and it’s corresponding TF-IDF. Not that the highest TF-IDF score in this post was the term “lunar” which uniquely identifies this post the best, and the term “people” received the lowest score as it is a general word with high chances of it occurring outside of the sci.space topic. And the chances of “lunar” occurring in for example the rec.motorcycles topic are fairly low.


Post Index: 714
Topic: sci.space

point seen copy Lunar Resources Data Purchase Act known Moon aut
horize U government purchase lunar science data private non profit v
endor selected basis competitive bidding aggregate cap bid award mil
lion copy want legalese contained Federal legislation free resource ev
aluate local congressional office listed phone book staffed people for
ward copy legal expert Simply ask consider supporting Lunar Resour
ces Data Purchase Act feedback negative positive congressional office
forward David Anderman E Yorba Linda Blvd Apt G Fullerton CA E Ma
il David Anderman ofa fidonet org resource local chapter National Spa
ce Society Members chapter happy work evaluate support Moon addr
ess telephone number nearest chapter send E mail check latest issue
Ad Astra library near Finally requested received information Moon se
nd request database recently corrupted information lost author thank
patience

	Term	TF-IDF
4702	lunar	0.273437
1219	chapter	0.260526
5086	moon	0.247321
6257	purchase	0.241391
6672	resources	0.208946
...	...	...
5374	number	0.051058
5920	point	0.048233
8729	work	0.045174
8580	want	0.044463
5753	people	0.040190

84 rows × 2 columns

Figure 7: TF-IDF Example

We have engineered our set of features from general occurrence counts Term Frequency Inverse Document Frequency scores. A noteworthy mention is that our feature set still has 22,069 dimensions and across 11,314 observations. These 22,069 can still be reduced further with dimensionality techniques such as Principal Component Analysis. The method seeks to project the data into a lower dimensional space where each axis (or principal component) captures the most variability in the data subject to the condition of being uncorrelated to the other axes. This last condition is important for dimensionality reduction in the sense that large datasets can contain many correlated variables which hold no additional information.

While it is possible to reduce the feature set further such approaches like Principal Component Analysis are mathematical reduction techniques. We chose to reduce our feature space as much as possible linguistically, by dropping stop words, lemmatization, etc.

Classifier Models

The step after TF-IDF Vectorization is to build classification models to which the training data is supplied. The models implemented here are Support Vector Machines, Naïve Bayes, and Random Forests.

For each model a set of parameters can be tuned to attain optimal results. We cycle through a set of hyper-parameters using `GridSearchCV` and `RandomizedSearchCV` within the `sklearn` package. GridSearch is an exhaustive method that cycle through every hyper-parameter and can be computationally expensive. Where feasible RandomizedSearch has been used to tune hyper-parameters, which have shown to be more efficient than GridSearch [7].

Each model has been build using cross validation of the training data to minimizing overfitting.

a) Support Vector Machines

The primary goal of Support Vector Machines (SVM) is to determine a decision boundary $(\overrightarrow{w})$ between classes with maximum separation. Between classes there can be an infinite number of decision boundaries. The points from each class closest to the decision boundary are called Support Vectors. SVM looks to find an optimal decision boundary such that Support Vectors from each class have a maximum decision boundary and each class is on different sides of the decision boundary. Classification of a new unseen point (X_i) is determined on which side of the decision boundary X_i resides on. Figure 8 shows a simple binary classification with linearly separable classes and two features (X_1 and X_2).

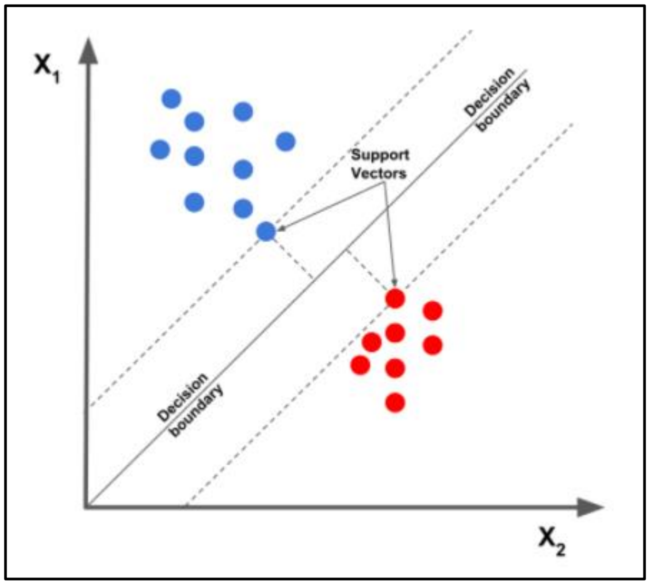


Figure 8: Linear SVM (Mallick, Satya 2018) [10]

For data that is not linearly separable due to either overlapping or in an nth dimensional space for n number of features, SVM utilizes kernel functions (also called kernel trick) to transform the feature space from n dimensions to a higher n+y dimensions to achieve better separation of classes. The separating boundary in this higher dimension is called a hyperplane. Figure 9 (left) shows data that is not linearly separable in a 2-dimensional feature space, but can be separated when transformed to a 3-dimensional feature space in Figure 9 (right).

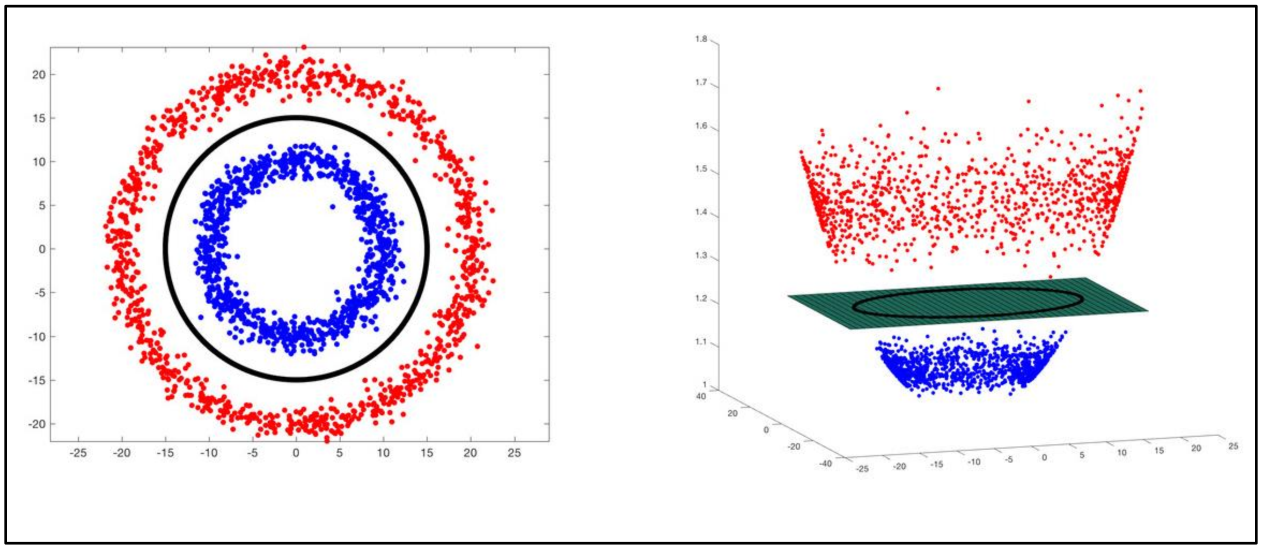
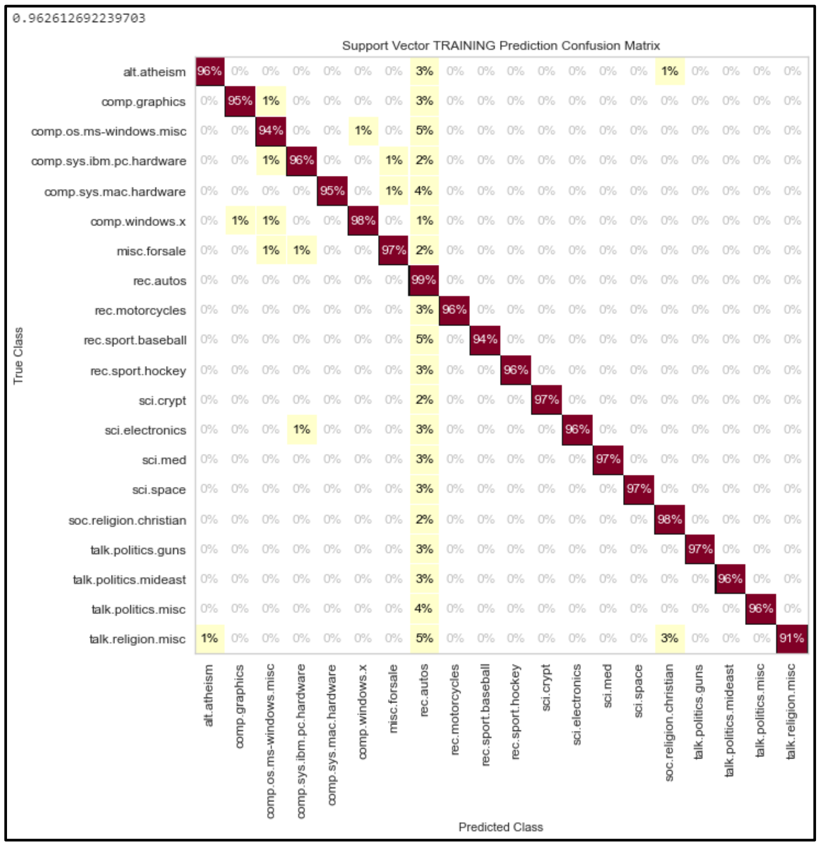


Figure 9: SVM Feature Space Transformation (Mallick, Satya 2018)



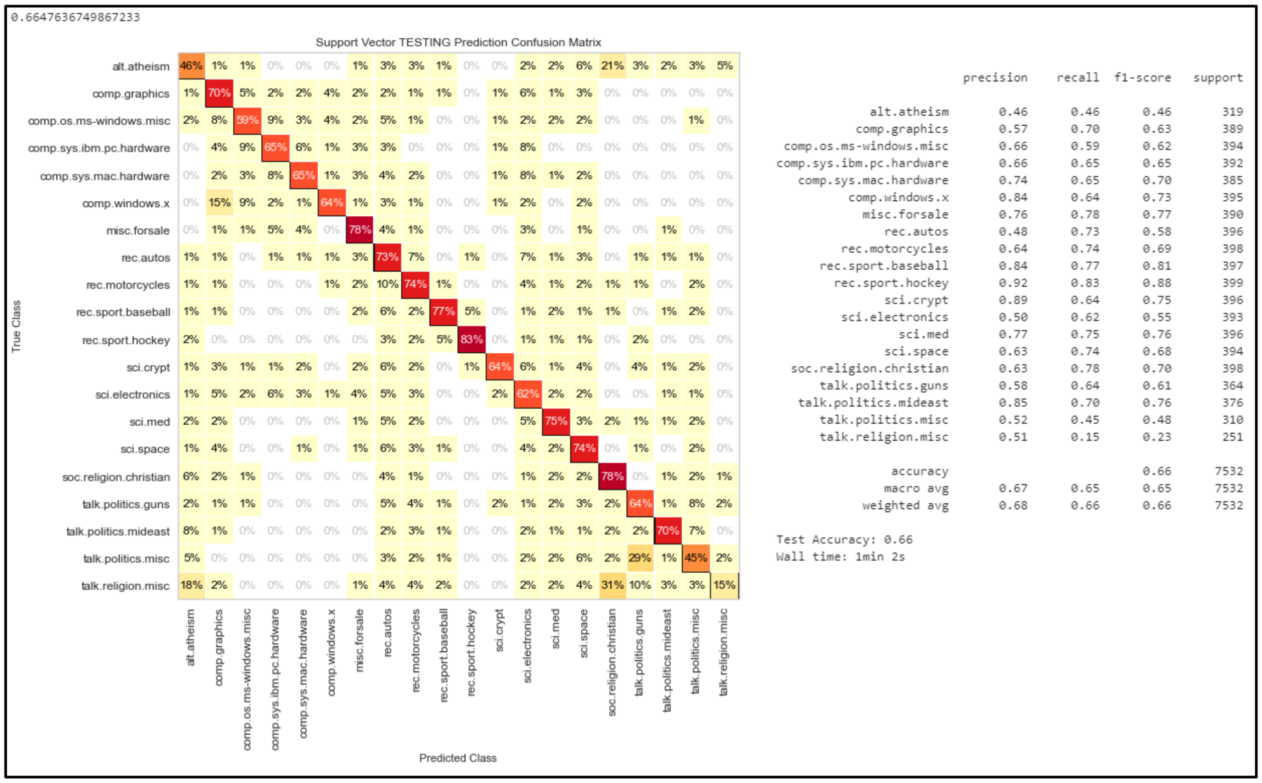
```
0.962612692239703
```

Support Vector TRAINING Prediction Confusion Matrix

True Class \ Predicted Class	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	96%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%
comp.graphics	0%	95%	1%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.os.ms-windows.misc	0%	0%	94%	0%	0%	1%	0%	5%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.sys.ibm.pc.hardware	0%	0%	1%	96%	0%	0%	1%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.sys.mac.hardware	0%	0%	0%	0%	95%	0%	1%	4%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.windows.x	0%	1%	1%	0%	0%	98%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
misc.forsale	0%	0%	1%	1%	0%	0%	97%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.autos	0%	0%	0%	0%	0%	0%	0%	99%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.motorcycles	0%	0%	0%	0%	0%	0%	0%	3%	96%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.sport.baseball	0%	0%	0%	0%	0%	0%	0%	5%	0%	94%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.sport.hockey	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	96%	0%	0%	0%	0%	0%	0%	0%	0%	0%
sci.crypt	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	97%	0%	0%	0%	0%	0%	0%	0%	0%
sci.electronics	0%	0%	0%	1%	0%	0%	0%	3%	0%	0%	0%	0%	96%	0%	0%	0%	0%	0%	0%	0%
sci.med	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	97%	0%	0%	0%	0%	0%	0%
sci.space	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	97%	0%	0%	0%	0%	0%
soc.religion.christian	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	98%	0%	0%	0%	0%
talk.politics.guns	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	97%	0%	0%	0%
talk.politics.mideast	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	96%	0%	0%
talk.politics.misc	0%	0%	0%	0%	0%	0%	0%	4%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	96%	0%
talk.religion.misc	1%	0%	0%	0%	0%	0%	0%	5%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	91%

When running the Support Vector Machine on the TF-IDF transformed corpus, the cross-validated model resulted in a 96% training accuracy. Of the 4% of misclassifications on the training data majority o of them were misclassified to the rec.autos topic. See Figure 10.

On the testing data, the cross validated SVM model performed at an over 66% accuracy, representing some overfitting, though looking more closely at the confusion matrix in Figure 11 shows that the majority of the misclassifications were among topics that would inherently have similar terms. Such as the talk.religion.misc topic had 27% of its misclassifications on the soc.religion.christian topic and 17% of it's misclassifications on the alt.atheism topic.



```
0.6647636749867233
```

Support Vector TESTING Prediction Confusion Matrix

True Class \ Predicted Class	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	46%	1%	1%	0%	0%	0%	1%	3%	3%	1%	0%	0%	2%	2%	6%	21%	3%	2%	3%	5%
comp.graphics	1%	70%	5%	2%	2%	4%	2%	2%	1%	1%	0%	1%	6%	1%	3%	0%	0%	0%	0%	0%
comp.os.ms-windows.misc	2%	8%	59%	9%	3%	4%	2%	5%	1%	0%	0%	1%	2%	2%	2%	0%	0%	0%	1%	0%
comp.sys.ibm.pc.hardware	0%	4%	9%	65%	6%	1%	3%	3%	0%	0%	0%	1%	8%	0%	0%	0%	0%	0%	0%	0%
comp.sys.mac.hardware	0%	2%	3%	8%	65%	1%	3%	4%	2%	0%	0%	1%	8%	1%	2%	0%	0%	0%	0%	0%
comp.windows.x	0%	15%	9%	2%	1%	64%	1%	3%	1%	0%	0%	1%	2%	0%	2%	0%	0%	0%	0%	0%
misc.forsale	0%	1%	1%	5%	4%	0%	78%	4%	1%	0%	0%	0%	3%	0%	1%	0%	0%	1%	0%	0%
rec.autos	1%	1%	0%	1%	1%	1%	3%	73%	7%	0%	1%	0%	7%	1%	3%	0%	1%	1%	1%	0%
rec.motorcycles	1%	1%	0%	0%	0%	1%	2%	10%	74%	1%	0%	0%	4%	1%	2%	1%	1%	0%	2%	0%
rec.sport.baseball	1%	1%	0%	0%	0%	0%	2%	6%	2%	77%	5%	0%	1%	2%	1%	1%	0%	1%	2%	0%
rec.sport.hockey	2%	0%	0%	0%	0%	0%	0%	3%	2%	5%	83%	0%	1%	1%	1%	0%	2%	0%	0%	0%
sci.crypt	1%	3%	1%	1%	2%	0%	2%	6%	2%	0%	1%	64%	6%	1%	4%	0%	4%	1%	2%	0%
sci.electronics	1%	5%	2%	6%	3%	1%	4%	5%	3%	0%	0%	2%	62%	2%	2%	0%	0%	1%	1%	0%
sci.med	2%	2%	0%	0%	0%	0%	1%	5%	2%	0%	0%	0%	5%	75%	3%	2%	1%	1%	2%	0%
sci.space	1%	4%	0%	0%	1%	0%	1%	6%	3%	1%	0%	0%	4%	2%	74%	0%	1%	0%	2%	0%
soc.religion.christian	6%	2%	1%	0%	0%	0%	0%	4%	1%	0%	0%	0%	1%	2%	2%	78%	0%	1%	2%	1%
talk.politics.guns	2%	1%	1%	0%	0%	0%	0%	5%	4%	1%	0%	2%	1%	2%	3%	2%	64%	1%	8%	2%
talk.politics.mideast	8%	1%	0%	0%	0%	0%	0%	2%	3%	1%	0%	0%	2%	1%	1%	2%	2%	70%	7%	0%
talk.politics.misc	5%	0%	0%	0%	0%	0%	0%	3%	2%	1%	0%	0%	2%	2%	6%	2%	29%	1%	45%	2%
talk.religion.misc	18%	2%	0%	0%	0%	0%	1%	4%	4%	2%	0%	0%	2%	2%	4%	31%	10%	3%	3%	15%

```
                          precision    recall  f1-score   support

             alt.atheism       0.46      0.46      0.46       319
           comp.graphics       0.57      0.70      0.63       389
 comp.os.ms-windows.misc       0.66      0.59      0.62       394
comp.sys.ibm.pc.hardware       0.66      0.65      0.65       392
   comp.sys.mac.hardware       0.74      0.65      0.70       385
          comp.windows.x       0.84      0.64      0.73       395
            misc.forsale       0.76      0.78      0.77       390
               rec.autos       0.48      0.73      0.58       396
         rec.motorcycles       0.64      0.74      0.69       398
      rec.sport.baseball       0.84      0.77      0.81       397
        rec.sport.hockey       0.92      0.83      0.88       399
               sci.crypt       0.89      0.64      0.75       396
         sci.electronics       0.50      0.62      0.55       393
                 sci.med       0.77      0.75      0.76       396
               sci.space       0.63      0.74      0.68       394
  soc.religion.christian       0.63      0.78      0.70       398
      talk.politics.guns       0.58      0.64      0.61       364
   talk.politics.mideast       0.85      0.70      0.76       376
      talk.politics.misc       0.52      0.45      0.48       310
      talk.religion.misc       0.51      0.15      0.23       251

                accuracy                           0.66      7532
               macro avg       0.67      0.65      0.65      7532
            weighted avg       0.68      0.66      0.66      7532

Test Accuracy: 0.66
Wall time: 1min 2s
```

Figure 11: SVM Testing Confusion Matrix

b) Multinomial Naïve Bayes

Naïve Bayes is a set of probabilistic algorithms that utilizes Bayes' Theorem and probability theory to make predictions such that the algorithm looks to determine the probability of an event A given the probability that another event B has already occurred.

Within the Naïve Bayes family of classifiers, the Multinomial Naïve Bayes classifier is most suited for text classification. This is because while both algorithms function similarly simple Naïve Bayes treats a post of a given class as a set of terms that are either present or not present, the Multinomial Naïve Bayes algorithm considers term frequency [8].

The term "Naïve" is coined for the name of this algorithm because its underlying assumption is that the features/terms in the dataset are mutually independent. Mutually independence means that an occurrence of one term in a does not impact the probability of another term occurring. In the example of text classification this independence assumption does not hold [9]. With that said Naïve Bayes is still a widely used for text classification.

```
0.8967650698249956
```

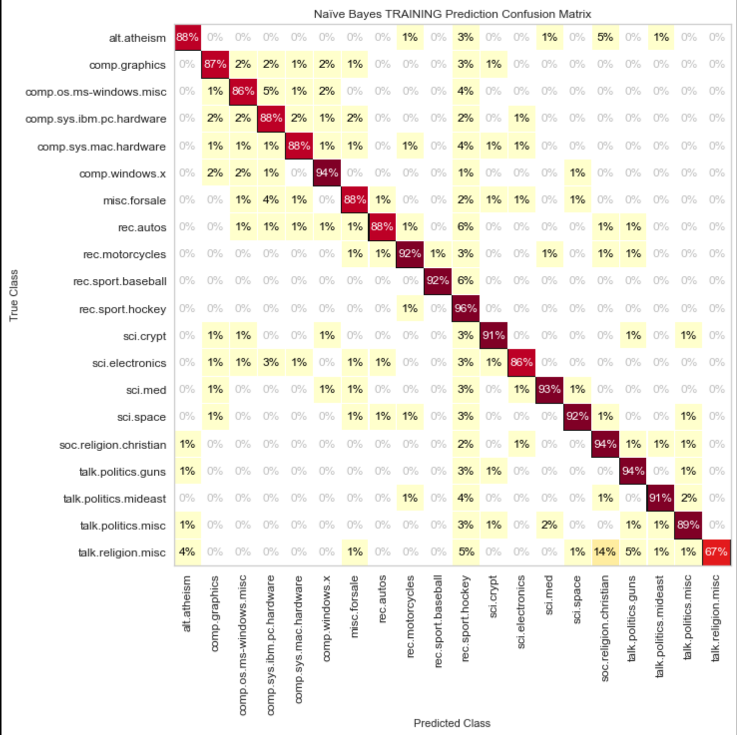
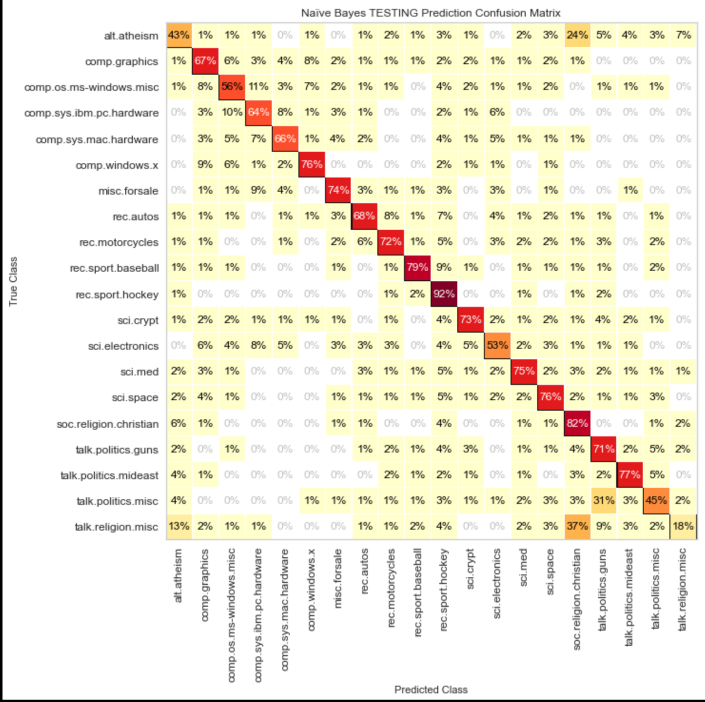


Figure 12: Naive Bayes Training Confusion Matrix

The Naïve Bayes model scored an overall 89% on the training accuracy, see Figure 12. The training results was not as fitted as the Support Vector Machine model and on the testing data the Naïve Bayes model showed misclassifications on the same topics that the SVM model had issues with on the testing data. Interestingly, as with the SVM model the Naïve Bayes model had one topic for which every topic had a misclassification in. This was the rec.sport.hockey topic. Interestingly the rec.sport.hockey topic was the most accurate topic on the SVM training data. On the testing data, the Naïve Bayes model showed similar results to the SVM model with a slightly better accuracy at 67%. See Figure 13.

```
0.6776420605416887
```



```
                          precision    recall  f1-score   support

             alt.atheism       0.52      0.43      0.47       319
           comp.graphics       0.60      0.67      0.63       389
 comp.os.ms-windows.misc       0.59      0.56      0.58       394
comp.sys.ibm.pc.hardware       0.60      0.64      0.62       392
   comp.sys.mac.hardware       0.69      0.66      0.67       385
          comp.windows.x       0.78      0.76      0.77       395
            misc.forsale       0.76      0.74      0.75       390
               rec.autos       0.74      0.68      0.71       396
         rec.motorcycles       0.75      0.72      0.73       398
      rec.sport.baseball       0.89      0.79      0.84       397
        rec.sport.hockey       0.57      0.92      0.70       399
               sci.crypt       0.79      0.73      0.76       396
         sci.electronics       0.64      0.53      0.58       393
                 sci.med       0.81      0.75      0.78       396
               sci.space       0.76      0.76      0.76       394
  soc.religion.christian       0.57      0.82      0.67       398
      talk.politics.guns       0.56      0.71      0.63       364
   talk.politics.mideast       0.83      0.77      0.80       376
      talk.politics.misc       0.58      0.45      0.51       310
      talk.religion.misc       0.48      0.18      0.27       251

                accuracy                           0.68      7532
               macro avg       0.68      0.66      0.66      7532
            weighted avg       0.68      0.68      0.67      7532

Test Accuracy: 0.68
Wall time: 330 ms
```

Figure 13: Naive Bayes Testing Confusion Matrix

c) <u>Random Forest</u>

A Decision Tree is a type of supervised learning model where the data is recursively split into two or more sub-populations given a criterion. Each split is headed by a node, where the upper most node is called the root node, and nodes which cannot be split further are called terminal (leaf) nodes. All other nodes are considered internal nodes. Based on a given population of observations, the population is split into sub-populations with the criteria that each split separates the sub-population better than the previous split. This recursive splitting cycles through the different predictor variables in order to find leaf nodes that make the purest class separation or most accurate predictions.

The Random Forest model is an ensemble method built from many individual decision trees and the classification output is determined by majority voting. The method aims to reduce the variance of individual trees and is particularly suited when predictors exhibit collinearity. Each tree in the forest is constructed using bootstrapped observations and a randomized subset of features. The forest growing procedure is controlled by two tuning parameters that we seek to optimize the number of trees in the forest and the number of random variables used to build each tree

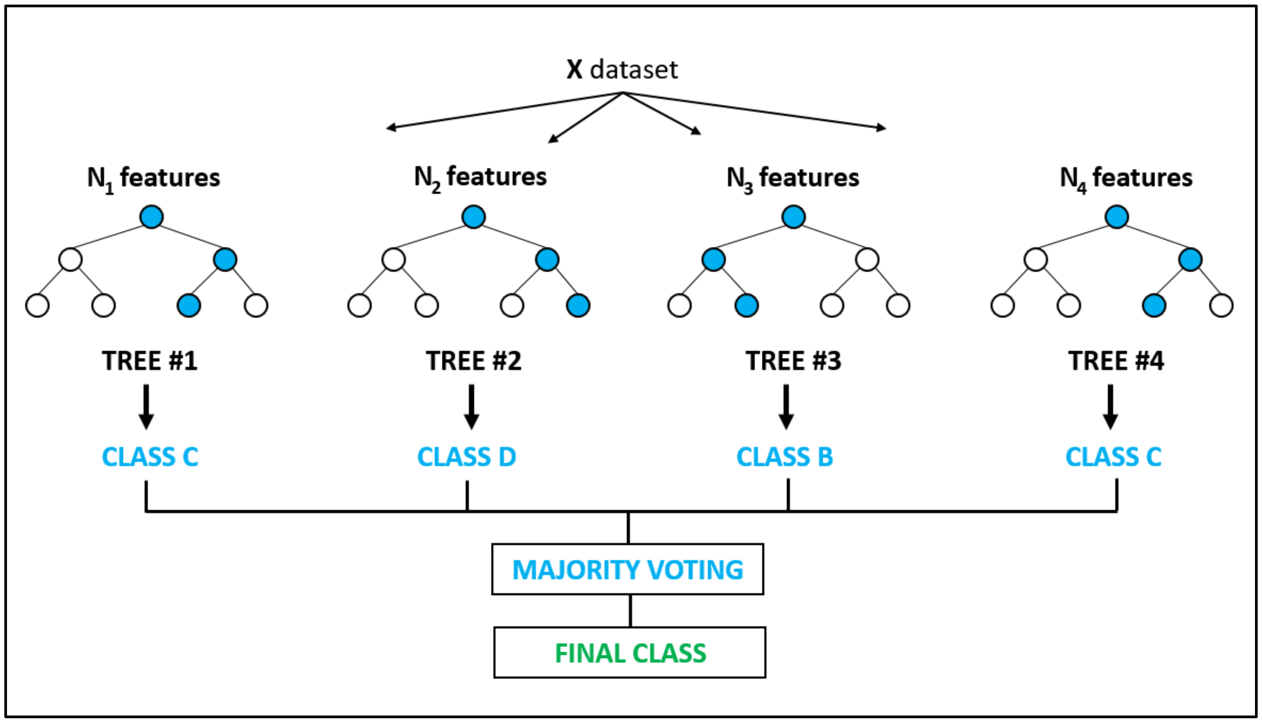
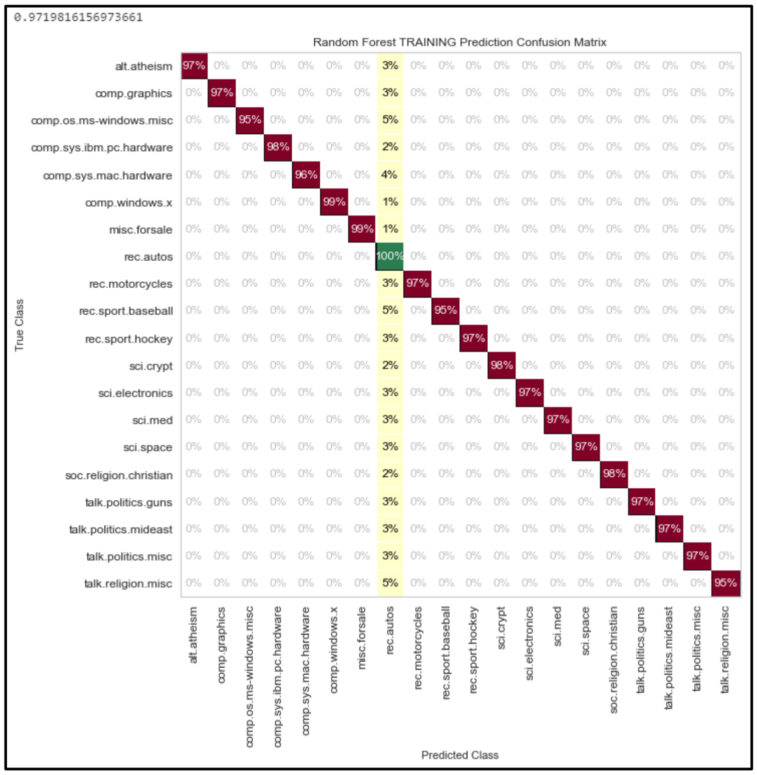


Figure 14: Random Forest Example (Suryadeepti, 2021)

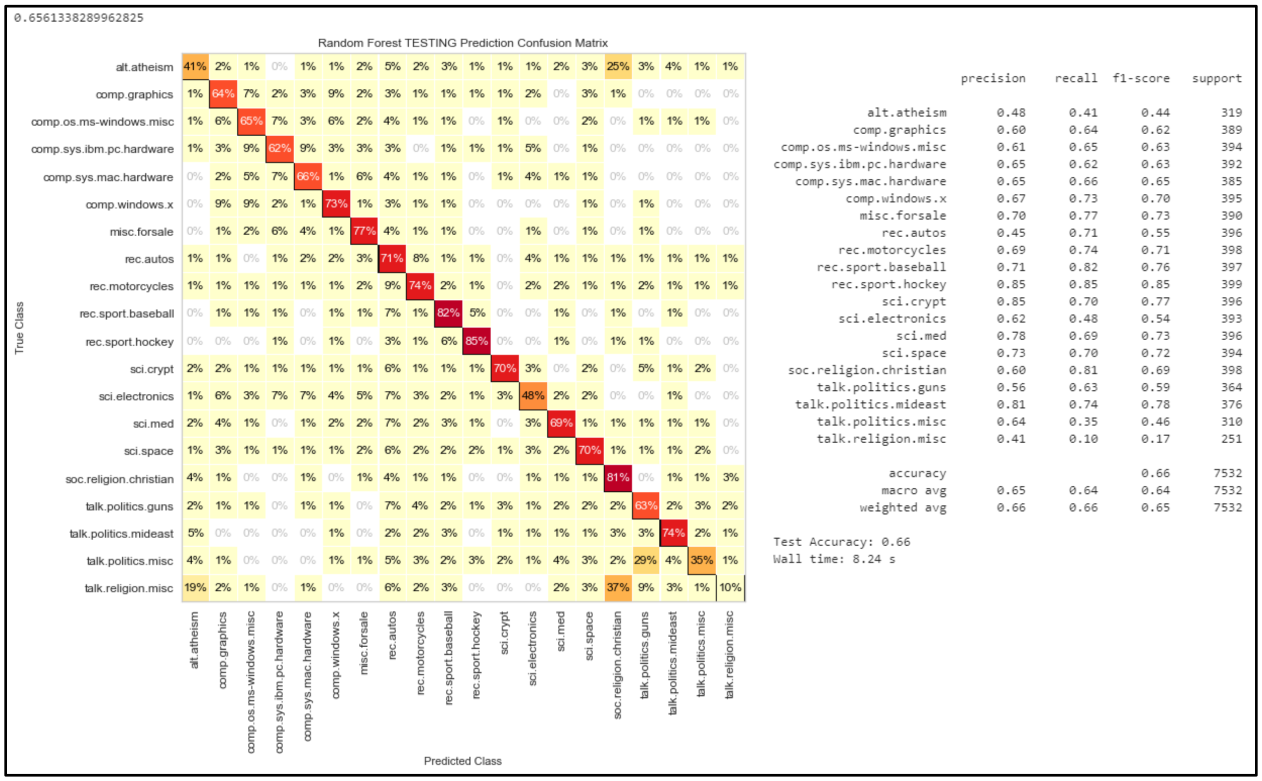
The Random Forest training results are almost identical to the training results of the SVM model, all misclassifications on the training data were on the rec.autos topic, see Figure 15. Even the percentages of misclassifications per topic are almost identical. On the testing data (Figure 16), the results are similar to SVM and Naïve Bayes models with an overall accuracy of 65%. The misclassifications were also generally clustered around similar topics.



```
0.9719816156973661
```

	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	97%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.graphics	0%	97%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.os.ms-windows.misc	0%	0%	95%	0%	0%	0%	0%	5%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.sys.ibm.pc.hardware	0%	0%	0%	98%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.sys.mac.hardware	0%	0%	0%	0%	96%	0%	0%	4%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
comp.windows.x	0%	0%	0%	0%	0%	99%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
misc.forsale	0%	0%	0%	0%	0%	0%	99%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.autos	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.motorcycles	0%	0%	0%	0%	0%	0%	0%	3%	97%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.sport.baseball	0%	0%	0%	0%	0%	0%	0%	5%	0%	95%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
rec.sport.hockey	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	97%	0%	0%	0%	0%	0%	0%	0%	0%	0%
sci.crypt	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	98%	0%	0%	0%	0%	0%	0%	0%	0%
sci.electronics	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	97%	0%	0%	0%	0%	0%	0%	0%
sci.med	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	97%	0%	0%	0%	0%	0%	0%
sci.space	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	97%	0%	0%	0%	0%	0%
soc.religion.christian	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	98%	0%	0%	0%	0%
talk.politics.guns	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	97%	0%	0%	0%
talk.politics.mideast	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	97%	0%	0%
talk.politics.misc	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	97%	0%
talk.religion.misc	0%	0%	0%	0%	0%	0%	0%	5%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	95%

Figure 15: Random Forest Training Confusion Matrix



```
0.6561338289962825
```

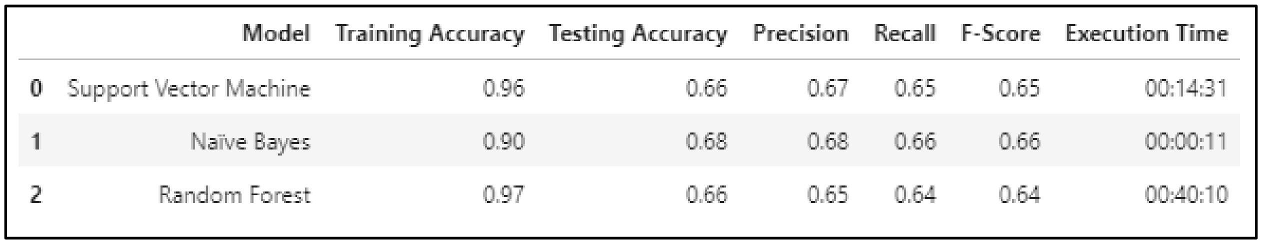
	alt.atheism	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	comp.windows.x	misc.forsale	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey	sci.crypt	sci.electronics	sci.med	sci.space	soc.religion.christian	talk.politics.guns	talk.politics.mideast	talk.politics.misc	talk.religion.misc
alt.atheism	41%	2%	1%	0%	1%	1%	2%	5%	2%	3%	1%	1%	1%	2%	3%	25%	3%	4%	1%	1%
comp.graphics	1%	64%	7%	2%	3%	9%	2%	3%	1%	1%	1%	1%	2%	0%	3%	1%	0%	0%	0%	0%
comp.os.ms-windows.misc	1%	6%	65%	7%	3%	6%	2%	4%	1%	1%	0%	1%	0%	0%	2%	0%	1%	1%	1%	0%
comp.sys.ibm.pc.hardware	1%	3%	9%	62%	9%	3%	3%	3%	0%	1%	1%	1%	5%	0%	1%	0%	0%	0%	0%	0%
comp.sys.mac.hardware	0%	2%	5%	7%	66%	1%	6%	4%	1%	1%	0%	1%	4%	1%	1%	0%	0%	0%	0%	0%
comp.windows.x	0%	9%	9%	2%	1%	73%	1%	3%	1%	1%	0%	0%	0%	0%	1%	0%	1%	0%	0%	0%
misc.forsale	0%	1%	2%	6%	4%	1%	77%	4%	1%	1%	0%	0%	1%	0%	1%	0%	1%	0%	0%	0%
rec.autos	1%	1%	0%	1%	2%	2%	3%	71%	8%	1%	1%	0%	4%	1%	1%	1%	1%	1%	1%	1%
rec.motorcycles	1%	1%	1%	1%	1%	1%	2%	9%	74%	2%	1%	0%	2%	2%	1%	1%	2%	1%	1%	1%
rec.sport.baseball	0%	1%	1%	1%	0%	1%	1%	7%	1%	82%	5%	0%	0%	1%	0%	1%	0%	1%	0%	0%
rec.sport.hockey	0%	0%	0%	1%	0%	1%	0%	3%	1%	6%	85%	0%	0%	1%	0%	1%	1%	0%	0%	0%
sci.crypt	2%	2%	1%	1%	1%	1%	1%	6%	1%	1%	1%	70%	3%	0%	2%	0%	5%	1%	2%	0%
sci.electronics	1%	6%	3%	7%	7%	4%	5%	7%	3%	2%	1%	3%	48%	2%	2%	0%	0%	1%	0%	0%
sci.med	2%	4%	1%	0%	1%	2%	2%	7%	2%	3%	1%	0%	3%	69%	1%	1%	1%	1%	1%	0%
sci.space	1%	3%	1%	1%	1%	1%	2%	6%	2%	2%	2%	1%	3%	2%	70%	1%	1%	1%	2%	0%
soc.religion.christian	4%	1%	0%	0%	1%	0%	1%	4%	1%	1%	0%	0%	1%	1%	1%	81%	0%	1%	1%	3%
talk.politics.guns	2%	1%	1%	0%	1%	1%	0%	7%	4%	2%	1%	3%	1%	2%	2%	2%	63%	2%	3%	2%
talk.politics.mideast	5%	0%	0%	0%	0%	1%	0%	2%	2%	3%	0%	1%	1%	1%	1%	3%	3%	74%	2%	1%
talk.politics.misc	4%	1%	0%	0%	0%	1%	1%	5%	3%	2%	3%	2%	1%	4%	3%	2%	29%	4%	35%	1%
talk.religion.misc	19%	2%	1%	0%	1%	0%	0%	6%	2%	3%	0%	0%	0%	2%	3%	37%	9%	3%	1%	10%

```
                          precision    recall  f1-score   support

             alt.atheism       0.48      0.41      0.44       319
           comp.graphics       0.60      0.64      0.62       389
 comp.os.ms-windows.misc       0.61      0.65      0.63       394
comp.sys.ibm.pc.hardware       0.65      0.62      0.63       392
   comp.sys.mac.hardware       0.65      0.66      0.65       385
          comp.windows.x       0.67      0.73      0.70       395
            misc.forsale       0.70      0.77      0.73       390
               rec.autos       0.45      0.71      0.55       396
         rec.motorcycles       0.69      0.74      0.71       398
      rec.sport.baseball       0.71      0.82      0.76       397
        rec.sport.hockey       0.85      0.85      0.85       399
               sci.crypt       0.85      0.70      0.77       396
         sci.electronics       0.62      0.48      0.54       393
                 sci.med       0.78      0.69      0.73       396
               sci.space       0.73      0.70      0.72       394
  soc.religion.christian       0.60      0.81      0.69       398
      talk.politics.guns       0.56      0.63      0.59       364
   talk.politics.mideast       0.81      0.74      0.78       376
      talk.politics.misc       0.64      0.35      0.46       310
      talk.religion.misc       0.41      0.10      0.17       251

                accuracy                           0.66      7532
               macro avg       0.65      0.64      0.64      7532
            weighted avg       0.66      0.66      0.65      7532

Test Accuracy: 0.66
Wall time: 8.24 s
```

Figure 16: Random Forest Testing Confusion Matrix

Model Selection

We’ve ran 3 models on the same training and testing data. Figure 17 shows a summary for the training and testing metrics of each of the models



	Model	Training Accuracy	Testing Accuracy	Precision	Recall	F-Score	Execution Time
0	Support Vector Machine	0.96	0.66	0.67	0.65	0.65	00:14:31
1	Naïve Bayes	0.90	0.68	0.68	0.66	0.66	00:00:11
2	Random Forest	0.97	0.66	0.65	0.64	0.64	00:40:10

Figure 17: Model Metrics Summary

All 3 had similar Training and Testing accuracies, the Precision, Recall, and also F-Scores are all similar where each is defined as:

$$Precision = \frac{TP}{(TP + FP)} \qquad Recall = \frac{TP}{(TP + FN)} \qquad F1\ Score = \frac{2\ (Precision * Recall)}{Precision + Recall}$$

Precision is defined as the ratio of correctly predicted positive observations to the overall total predicted positive observations. In other words, of the ones that were predicted as a topic, how many where actually that topic. Recall is defined as ratio of correctly predicted positive observations compared to all observations in this class. In other words, Recall is the ratio of how good the model is at identifying a topic. The F1-Score is the weighted average of the Precision and Recall Scores. For our dataset, we will want to maximize the F1-Score and while Support Vector Machine and Naïve Bayes models have the same F1-Scores, we can look at the individual F1-Scores per Topic (Figure 18).

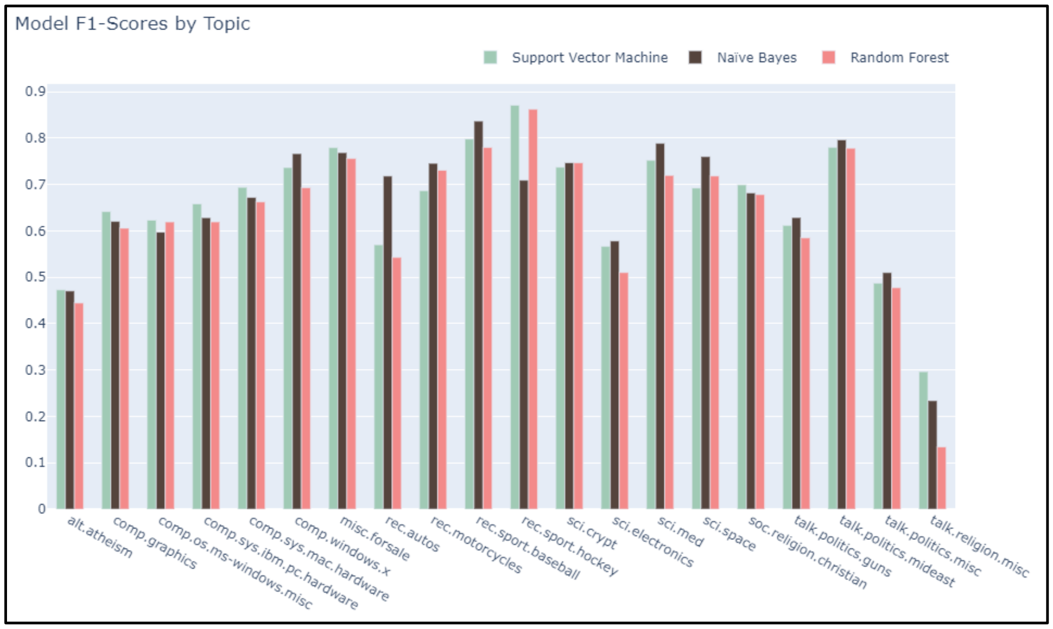


Figure 18: Model F1-Scores by Topic

Figure 18, shows that the F1-Score is generally same across all models with some slight variation. All models performed poorly on talk.religion.misc and alt.athesim. talk.politics was another topic that performed poorly. This is because for these were subtopics of general topics seen in the table in Figure 2. While it can be said that certain models can be beneficial for certain topics the current results do not show a significant different between the models. This is except for execution time per model. Random Forest and Support Vector Models were computationally expensive and the execution time (~40mins, ~11mins respectively) of each show this. The Naïve Bayes model on the same testing and training datasets executed in seconds providing far less computational expense for similar accuracy results.

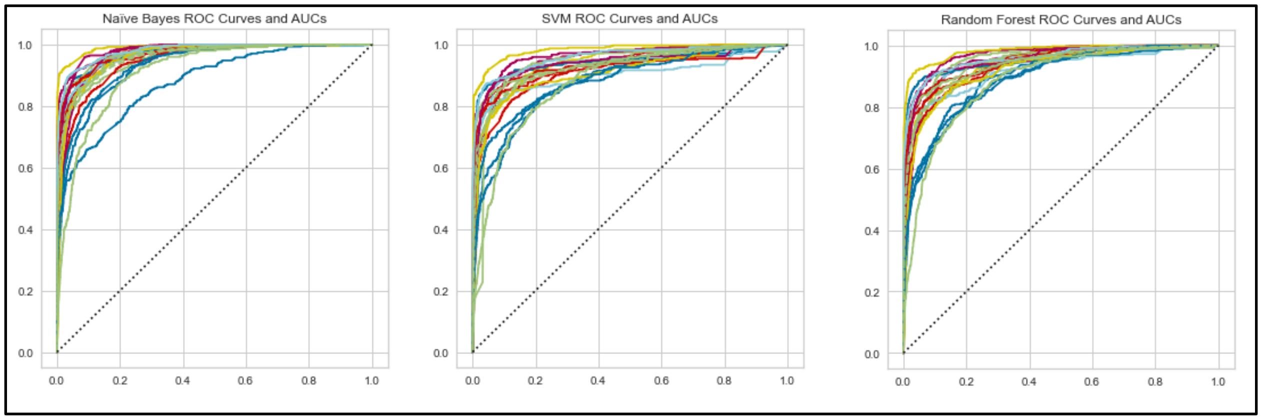


Figure 19: AUC-ROC Curves

The AUC-ROC curve shows that while Naïve Bayes performed poorly on one topic, it was exceptionally well on the other topics on par with SVM and Random Forest. Given its simplicity and time efficiency for this dataset and given hardware, Naïve Bayes would be the model of choice.

Conclusion

The comparison of 3 (Support Vector Machine, Naïve Bayes, and Random Forest) models on the 20 Newsgroups dataset yields similar results in terms of Accuracy, Precision, Recall, and F1-Score. All models behaved similarly and generally misclassified more on a set of topics that were related to each other. These topics have similar terms and therefore the models generally did poorly on these topics. No model had a clear and distinct advantage in terms of prediction. While the models did differ in terms of execution time. We saw that the of the 3 models Naïve Bayes had a very impactful advantage in terms of execution. This is because Naïve Bayes' main strength is its efficiency: Training and classification can be accomplished with one pass over the data. Because it combines efficiency with good accuracy it is often used as a baseline in text classification research. It is often the method of choice [9]. The overall strategy and approach taken were similar to other well-known approaches and we can see that while it is not possible to build an exhaustive list of pre-processing strategies and all given models, the best approach is to understand the given dataset and then determine a strategy.

Future Work

There are additional approaches and adjustments that can be made to see if accuracy and F1-Scores can be further increases based on other studies referenced in this study. These additional areas are:

- Use of N-Grams. In the existing approach, only single terms were considered as part of the overall feature set. Via N-Grams multiple words can be considered as a feature
- Named Entity Removal. Named entities such as names, can either be added or removed as features
- Principal Component Analysis. The dimensions of the feature set can be further reduced mathematically via Principal Component Analysis
- Validation Cures. Tune between over and underfitting of a model to find a balance between bias and variance

References (10 sources)

[1] Deepshikha Kalita - Supervised and Unsupervised Document Classification-A survey. International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 6 (2), 2015, 1971-1974

[2] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown (2019) - Text Classification Algorithms: A Survey, MDPI

[3] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, Krys Kochut (2017) - A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques, KDD Bigdas, August 2017, Halifax, Canada

[4] Vandana Korde, C Namrata Mahender (2012) – Text Classification and Classifiers: A Survey, International Journal of Artificial Intelligence & Applications (IJAIA), Vol.3, No.2, March 2012

[5] HaCohen-Kerner Y, Miller D, Yigal Y (2020) The influence of preprocessing on text classification using a bag-of-words representation. PLoS ONE 15(5): e0232525.

[6] Cambridge University Press (2009) Information Retrieval: Chapter 2 – The Term Vocabulary and Postings Lists

[7] Bergstra, Bengio (2012) - Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research 13, 281-305

[8] McCallum, Nigam (1998) - A Comparison of Event Models for Naive Bayes Text Classification. Association for the Advancement of Artificial Intelligence

[9] Cambridge University Press (2009) Information Retrieval: Chapter 15 – Text Classification and Naïve Bayes

[10] Satya Mallick, Support Vector Machines (SVM) (2018), Learn OpenCV

Appendices

1. Data Exploration

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
import plotly.graph_objects as go
import plotly.express as px
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import matplotlib.pylab as plt
import numpy as np
from wordcloud import WordCloud, STOPWORDS
import re
from sklearn.cluster import KMeans
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn import metrics
from yellowbrick.classifier import ROCAUC
import nltk
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
lemma = WordNetLemmatizer()
from yellowbrick.classifier import ConfusionMatrix
from sklearn.feature_extraction import text
from nltk.corpus import stopwords
from nltk.tokenize import regexp_tokenize
import time
from textblob import TextBlob, Word
```

```
def exec_time(start, end):
   diff_time = end - start
   m, s = divmod(diff_time, 60)
   h, m = divmod(m, 60)
   s,m,h = int(round(s, 0)), int(round(m, 0)), int(round(h, 0))
   return ("{0:02d}:{1:02d}:{2:02d}".format(h, m, s))
```

```


data_train = fetch_20newsgroups(data_home='/path/to/data',
                                subset='train', remove = ("headers", "footers", "quotes"),
                                shuffle=True, random_state=5)


data_test = fetch_20newsgroups(data_home='/path/to/data',
                               subset='test', remove = ("headers", "footers", "quotes"),
                               shuffle=True, random_state=5)

data_all = fetch_20newsgroups(data_home='/path/to/data',
                              subset='all',
                              shuffle=True, random_state=5)
```

```
i = 714
print("Category: " + data_train.target_names[data_train.target[i]] + "\n")
print("\n".join(data_train.data[i].split("\n")[:]))
```

```
%%time
word_count_vectorizer = CountVectorizer()
word_counts=word_count_vectorizer.fit_transform(data_all.data)
```

```
from scipy.sparse import csr_matrix
A = csr_matrix(word_counts)
d = {"Count":np.array(A.sum(axis=1).flatten())[0], "Topic":[data_all.target_names[r] for r in data_all.target]}

count_df = pd.DataFrame(d)
```

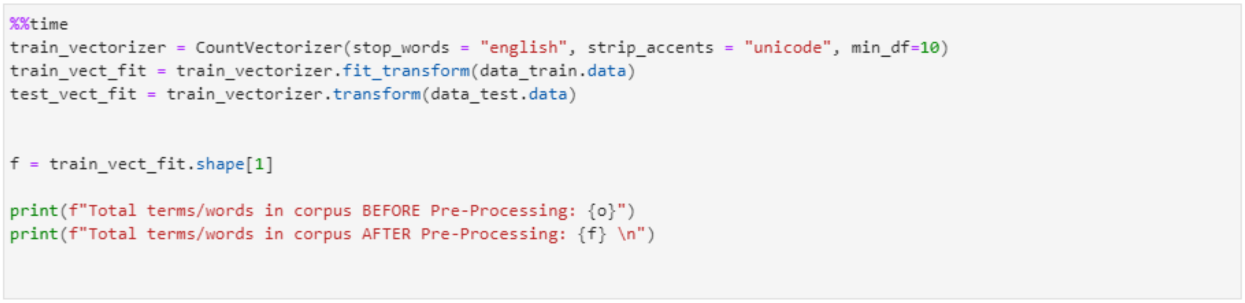
```
#print(count_df.sort_values("Count"))
#fig = px.histogram(count_df, x="Count", nbins=1000)
fig = px.strip(count_df, y="Topic", x="Count", color = "Topic", orientation="v", title="Word Counts per Post")
fig.update_layout(height=700, width=1500, title="Word Counts", bargap=.25)
fig.update_xaxes(nticks=30)
fig.update_yaxes(ticks="outside", tickwidth=1, ticklen=10, col=1)
fig.show()
```

2. Pre-Processing

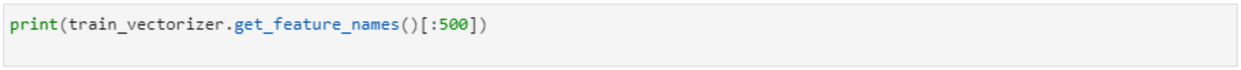
```
example_vectorizer = CountVectorizer(lowercase=False)
example_vect_fit = example_vectorizer.fit_transform(data_train.data)
print(example_vect_fit.shape)
o=example_vect_fit.shape[1]
print(f"Total terms/words in corpus: {o} \n")
print("List of words in corpus (first 500): ")
print(example_vectorizer.get_feature_names()[:500])
```

```
%%time

#Credit: The Natural Language Processing Workshop Chapter 2: Stop Words
#---------------------------------------------------------------------
def remove_stop_words_regex(text,stop_word_list):
    return ' '.join([word for word in regexp_tokenize(text, "[\w']+") if word.lower() not in stop_word_list])
#----------------------------------------------------------------------

custom_stopwords = list(text.ENGLISH_STOP_WORDS) + stopwords.words('english') + ["don't", "like", "can't", "huh"]
custom_stopwords = list(np.unique(np.array(custom_stopwords)))


for i in range(len(data_train.data)):
    data_train.data[i] = remove_stop_words_regex(data_train.data[i],custom_stopwords)

for i in range(len(data_test.data)):
    data_test.data[i] = remove_stop_words_regex(data_test.data[i],custom_stopwords)
```

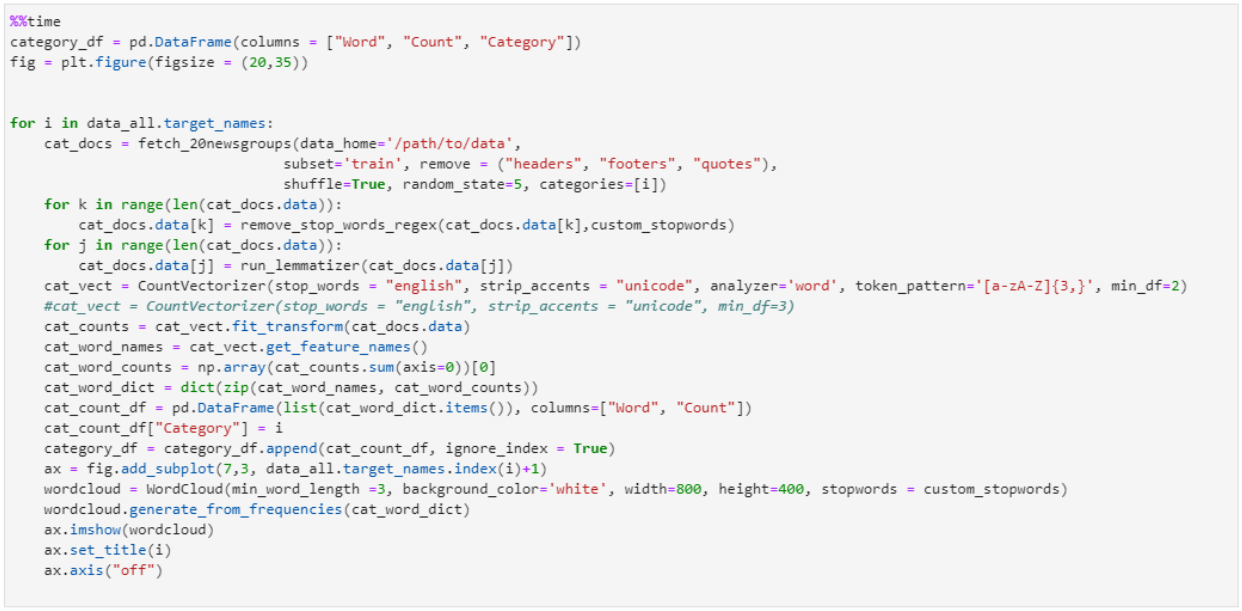
```
%%time
#Credit: Lemmatization Approaches with Examples in Python
#----------------------------------------------------
def run_lemmatizer(t):
    t = re.sub(r'[^A-Za-z ]', ' ', t)
    blob = TextBlob(t)
    tag_dict = {"J": 'a', "N": 'n', "V": 'v', "R": 'r'}
    wt = [(w, tag_dict.get(pos[0], 'n')) for w, pos in blob.tags]
    lemmatized = [wd.lemmatize(tag) for wd, tag in wt]
    return " ".join(lemmatized)
#----------------------------------------------------

for i in range(len(data_train.data)):
    data_train.data[i] = run_lemmatizer(data_train.data[i])

for i in range(len(data_test.data)):
    data_test.data[i] = run_lemmatizer(data_test.data[i])
```



```
%%time
train_vectorizer = CountVectorizer(stop_words = "english", strip_accents = "unicode", min_df=10)
train_vect_fit = train_vectorizer.fit_transform(data_train.data)
test_vect_fit = train_vectorizer.transform(data_test.data)


f = train_vect_fit.shape[1]

print(f"Total terms/words in corpus BEFORE Pre-Processing: {o}")
print(f"Total terms/words in corpus AFTER Pre-Processing: {f} \n")
```

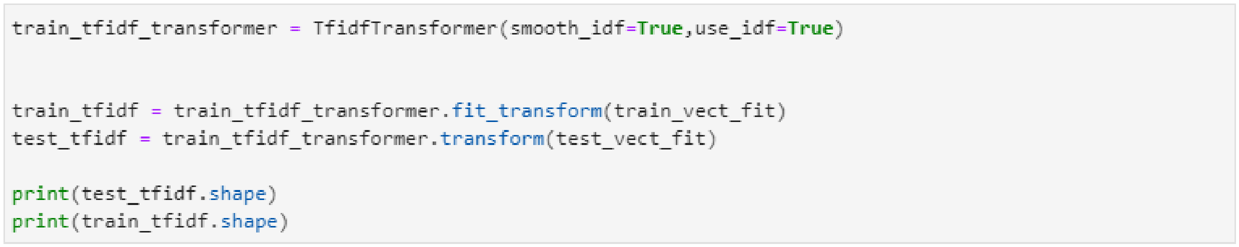


```
print(train_vectorizer.get_feature_names()[:500])
```

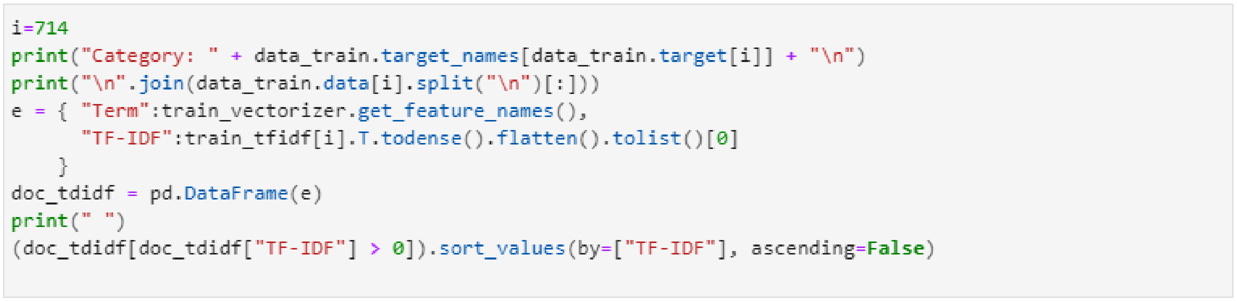


```
%%time
category_df = pd.DataFrame(columns = ["Word", "Count", "Category"])
fig = plt.figure(figsize = (20,35))


for i in data_all.target_names:
    cat_docs = fetch_20newsgroups(data_home='/path/to/data',
                                  subset='train', remove = ("headers", "footers", "quotes"),
                                  shuffle=True, random_state=5, categories=[i])
    for k in range(len(cat_docs.data)):
        cat_docs.data[k] = remove_stop_words_regex(cat_docs.data[k],custom_stopwords)
    for j in range(len(cat_docs.data)):
        cat_docs.data[j] = run_lemmatizer(cat_docs.data[j])
    cat_vect = CountVectorizer(stop_words = "english", strip_accents = "unicode", analyzer='word', token_pattern='[a-zA-Z]{3,}', min_df=2)
    #cat_vect = CountVectorizer(stop_words = "english", strip_accents = "unicode", min_df=3)
    cat_counts = cat_vect.fit_transform(cat_docs.data)
    cat_word_names = cat_vect.get_feature_names()
    cat_word_counts = np.array(cat_counts.sum(axis=0))[0]
    cat_word_dict = dict(zip(cat_word_names, cat_word_counts))
    cat_count_df = pd.DataFrame(list(cat_word_dict.items()), columns=["Word", "Count"])
    cat_count_df["Category"] = i
    category_df = category_df.append(cat_count_df, ignore_index = True)
    ax = fig.add_subplot(7,3, data_all.target_names.index(i)+1)
    wordcloud = WordCloud(min_word_length =3, background_color='white', width=800, height=400, stopwords = custom_stopwords)
    wordcloud.generate_from_frequencies(cat_word_dict)
    ax.imshow(wordcloud)
    ax.set_title(i)
    ax.axis("off")
```

3. Feature Engineering



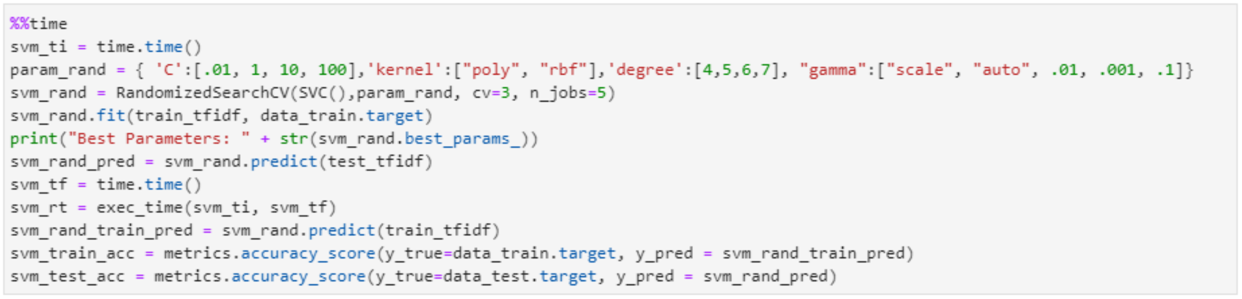
```
train_tfidf_transformer = TfidfTransformer(smooth_idf=True,use_idf=True)


train_tfidf = train_tfidf_transformer.fit_transform(train_vect_fit)
test_tfidf = train_tfidf_transformer.transform(test_vect_fit)

print(test_tfidf.shape)
print(train_tfidf.shape)
```



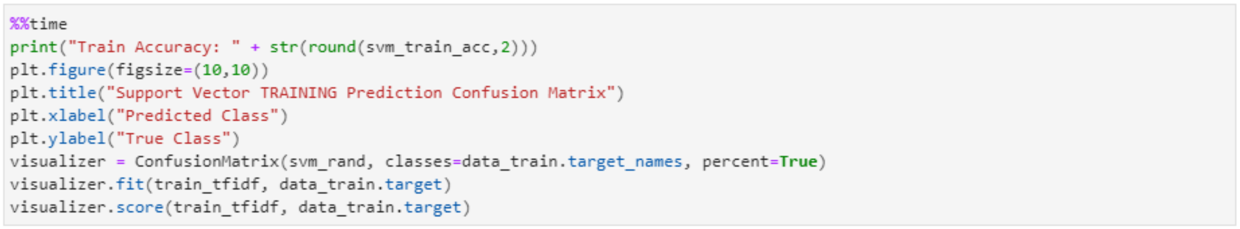
```
i=714
print("Category: " + data_train.target_names[data_train.target[i]] + "\n")
print("\n".join(data_train.data[i].split("\n")[:]))
e = { "Term":train_vectorizer.get_feature_names(),
      "TF-IDF":train_tfidf[i].T.todense().flatten().tolist()[0]
    }
doc_tdidf = pd.DataFrame(e)
print(" ")
(doc_tdidf[doc_tdidf["TF-IDF"] > 0]).sort_values(by=["TF-IDF"], ascending=False)
```

4. Classification Models

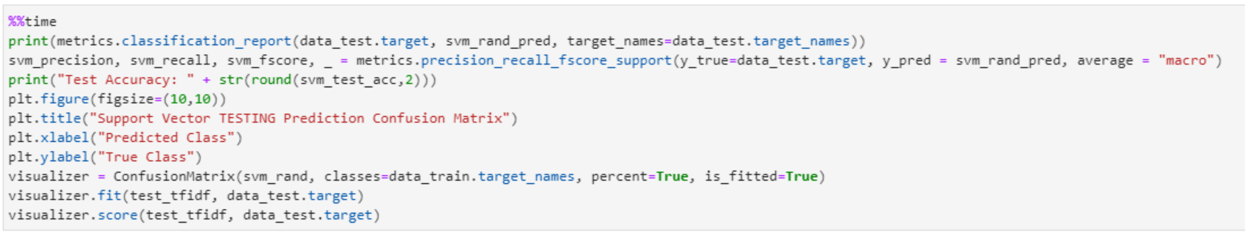
a. SVM



```python
%%time
svm_ti = time.time()
param_rand = { 'C':[.01, 1, 10, 100],'kernel':["poly", "rbf"],'degree':[4,5,6,7], "gamma":["scale", "auto", .01, .001, .1]}
svm_rand = RandomizedSearchCV(SVC(),param_rand, cv=3, n_jobs=5)
svm_rand.fit(train_tfidf, data_train.target)
print("Best Parameters: " + str(svm_rand.best_params_))
svm_rand_pred = svm_rand.predict(test_tfidf)
svm_tf = time.time()
svm_rt = exec_time(svm_ti, svm_tf)
svm_rand_train_pred = svm_rand.predict(train_tfidf)
svm_train_acc = metrics.accuracy_score(y_true=data_train.target, y_pred = svm_rand_train_pred)
svm_test_acc = metrics.accuracy_score(y_true=data_test.target, y_pred = svm_rand_pred)
```

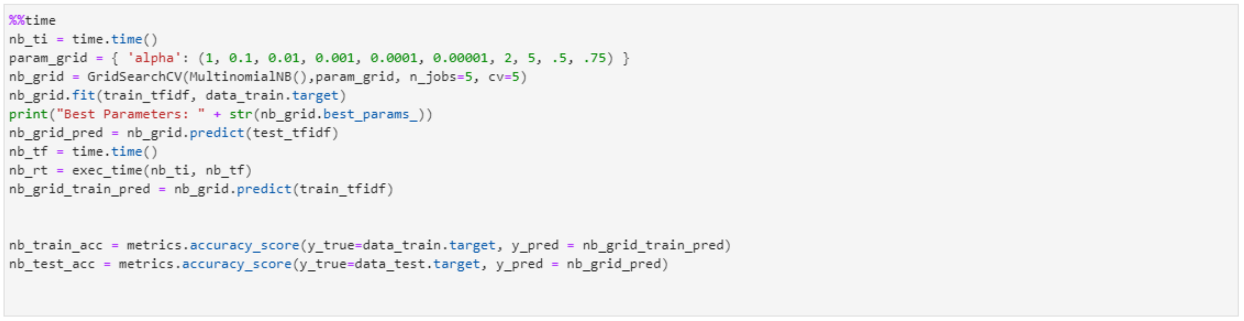


```python
%%time
print("Train Accuracy: " + str(round(svm_train_acc,2)))
plt.figure(figsize=(10,10))
plt.title("Support Vector TRAINING Prediction Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
visualizer = ConfusionMatrix(svm_rand, classes=data_train.target_names, percent=True)
visualizer.fit(train_tfidf, data_train.target)
visualizer.score(train_tfidf, data_train.target)
```

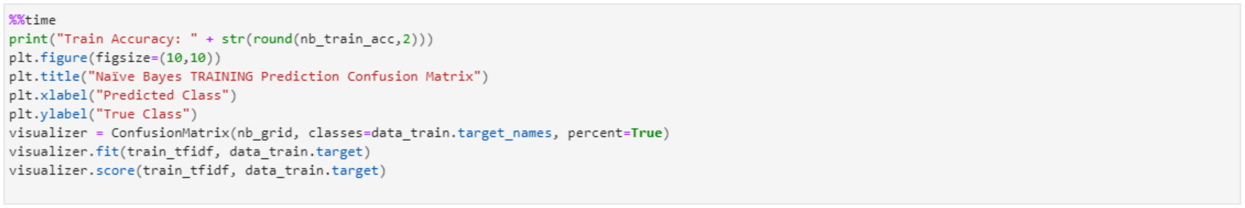


```python
%%time
print(metrics.classification_report(data_test.target, svm_rand_pred, target_names=data_test.target_names))
svm_precision, svm_recall, svm_fscore, _ = metrics.precision_recall_fscore_support(y_true=data_test.target, y_pred = svm_rand_pred, average = "macro")
print("Test Accuracy: " + str(round(svm_test_acc,2)))
plt.figure(figsize=(10,10))
plt.title("Support Vector TESTING Prediction Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
visualizer = ConfusionMatrix(svm_rand, classes=data_train.target_names, percent=True, is_fitted=True)
visualizer.fit(test_tfidf, data_test.target)
visualizer.score(test_tfidf, data_test.target)
```

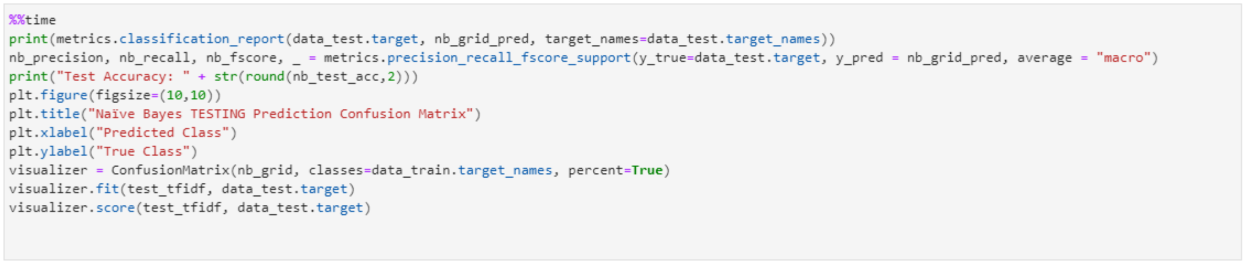
b. Naïve Bayes



```python
%%time
nb_ti = time.time()
param_grid = { 'alpha': (1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 2, 5, .5, .75) }
nb_grid = GridSearchCV(MultinomialNB(),param_grid, n_jobs=5, cv=5)
nb_grid.fit(train_tfidf, data_train.target)
print("Best Parameters: " + str(nb_grid.best_params_))
nb_grid_pred = nb_grid.predict(test_tfidf)
nb_tf = time.time()
nb_rt = exec_time(nb_ti, nb_tf)
nb_grid_train_pred = nb_grid.predict(train_tfidf)


nb_train_acc = metrics.accuracy_score(y_true=data_train.target, y_pred = nb_grid_train_pred)
nb_test_acc = metrics.accuracy_score(y_true=data_test.target, y_pred = nb_grid_pred)
```

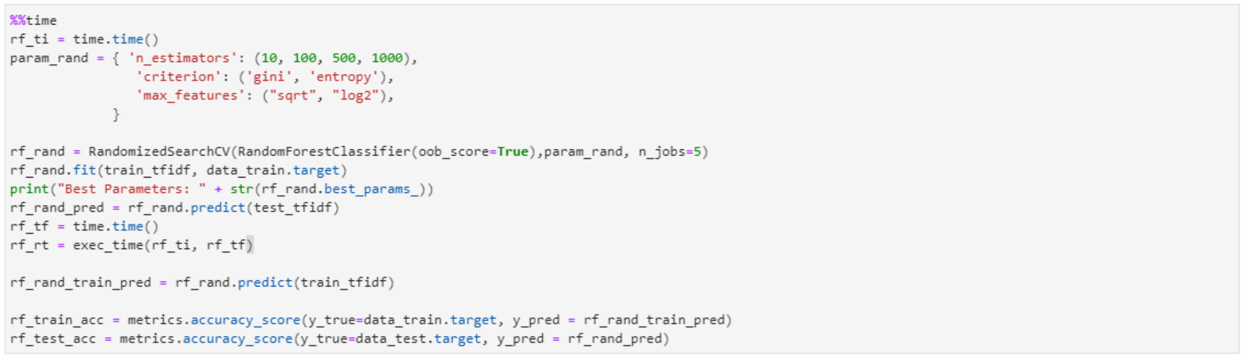


```python
%%time
print("Train Accuracy: " + str(round(nb_train_acc,2)))
plt.figure(figsize=(10,10))
plt.title("Naïve Bayes TRAINING Prediction Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
visualizer = ConfusionMatrix(nb_grid, classes=data_train.target_names, percent=True)
visualizer.fit(train_tfidf, data_train.target)
visualizer.score(train_tfidf, data_train.target)
```

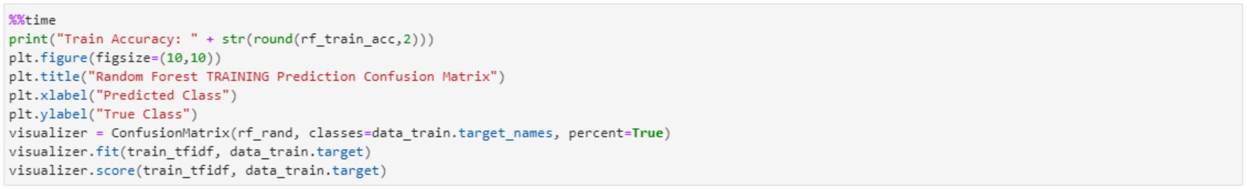


```python
%%time
print(metrics.classification_report(data_test.target, nb_grid_pred, target_names=data_test.target_names))
nb_precision, nb_recall, nb_fscore, _ = metrics.precision_recall_fscore_support(y_true=data_test.target, y_pred = nb_grid_pred, average = "macro")
print("Test Accuracy: " + str(round(nb_test_acc,2)))
plt.figure(figsize=(10,10))
plt.title("Naïve Bayes TESTING Prediction Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
visualizer = ConfusionMatrix(nb_grid, classes=data_train.target_names, percent=True)
visualizer.fit(test_tfidf, data_test.target)
visualizer.score(test_tfidf, data_test.target)
```

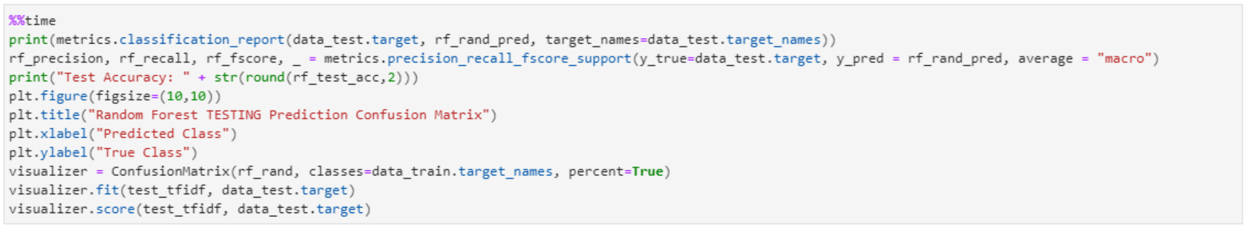
c. Random Forest



```
%%time
rf_ti = time.time()
param_rand = { 'n_estimators': (10, 100, 500, 1000),
               'criterion': ('gini', 'entropy'),
               'max_features': ("sqrt", "log2"),
             }

rf_rand = RandomizedSearchCV(RandomForestClassifier(oob_score=True),param_rand, n_jobs=5)
rf_rand.fit(train_tfidf, data_train.target)
print("Best Parameters: " + str(rf_rand.best_params_))
rf_rand_pred = rf_rand.predict(test_tfidf)
rf_tf = time.time()
rf_rt = exec_time(rf_ti, rf_tf)

rf_rand_train_pred = rf_rand.predict(train_tfidf)

rf_train_acc = metrics.accuracy_score(y_true=data_train.target, y_pred = rf_rand_train_pred)
rf_test_acc = metrics.accuracy_score(y_true=data_test.target, y_pred = rf_rand_pred)
```

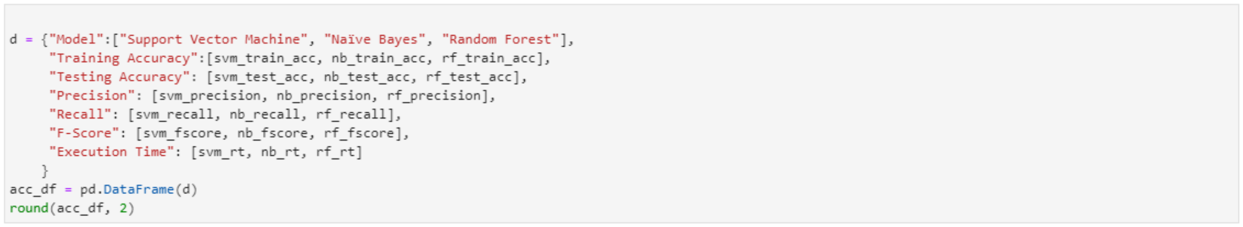


```
%%time
print("Train Accuracy: " + str(round(rf_train_acc,2)))
plt.figure(figsize=(10,10))
plt.title("Random Forest TRAINING Prediction Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
visualizer = ConfusionMatrix(rf_rand, classes=data_train.target_names, percent=True)
visualizer.fit(train_tfidf, data_train.target)
visualizer.score(train_tfidf, data_train.target)
```

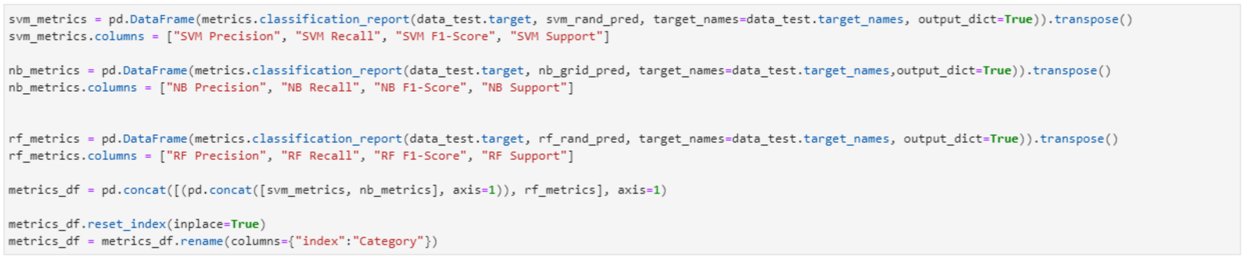


```
%%time
print(metrics.classification_report(data_test.target, rf_rand_pred, target_names=data_test.target_names))
rf_precision, rf_recall, rf_fscore, _ = metrics.precision_recall_fscore_support(y_true=data_test.target, y_pred = rf_rand_pred, average = "macro")
print("Test Accuracy: " + str(round(rf_test_acc,2)))
plt.figure(figsize=(10,10))
plt.title("Random Forest TESTING Prediction Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
visualizer = ConfusionMatrix(rf_rand, classes=data_train.target_names, percent=True)
visualizer.fit(test_tfidf, data_test.target)
visualizer.score(test_tfidf, data_test.target)
```

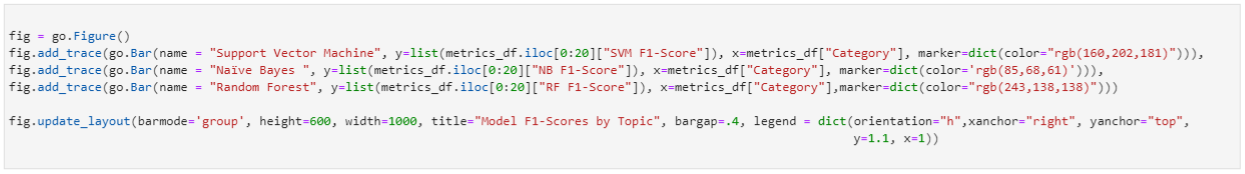
5. Evaluation



```
d = {"Model":["Support Vector Machine", "Naïve Bayes", "Random Forest"],
     "Training Accuracy":[svm_train_acc, nb_train_acc, rf_train_acc],
     "Testing Accuracy": [svm_test_acc, nb_test_acc, rf_test_acc],
     "Precision": [svm_precision, nb_precision, rf_precision],
     "Recall": [svm_recall, nb_recall, rf_recall],
     "F-Score": [svm_fscore, nb_fscore, rf_fscore],
     "Execution Time": [svm_rt, nb_rt, rf_rt]
    }
acc_df = pd.DataFrame(d)
round(acc_df, 2)
```



```
svm_metrics = pd.DataFrame(metrics.classification_report(data_test.target, svm_rand_pred, target_names=data_test.target_names, output_dict=True)).transpose()
svm_metrics.columns = ["SVM Precision", "SVM Recall", "SVM F1-Score", "SVM Support"]

nb_metrics = pd.DataFrame(metrics.classification_report(data_test.target, nb_grid_pred, target_names=data_test.target_names,output_dict=True)).transpose()
nb_metrics.columns = ["NB Precision", "NB Recall", "NB F1-Score", "NB Support"]


rf_metrics = pd.DataFrame(metrics.classification_report(data_test.target, rf_rand_pred, target_names=data_test.target_names, output_dict=True)).transpose()
rf_metrics.columns = ["RF Precision", "RF Recall", "RF F1-Score", "RF Support"]

metrics_df = pd.concat([(pd.concat([svm_metrics, nb_metrics], axis=1)), rf_metrics], axis=1)

metrics_df.reset_index(inplace=True)
metrics_df = metrics_df.rename(columns={"index":"Category"})
```



```
fig = go.Figure()
fig.add_trace(go.Bar(name = "Support Vector Machine", y=list(metrics_df.iloc[0:20]["SVM F1-Score"]), x=metrics_df["Category"], marker=dict(color="rgb(160,202,181)"))),
fig.add_trace(go.Bar(name = "Naïve Bayes ", y=list(metrics_df.iloc[0:20]["NB F1-Score"]), x=metrics_df["Category"], marker=dict(color='rgb(85,68,61)'))),
fig.add_trace(go.Bar(name = "Random Forest", y=list(metrics_df.iloc[0:20]["RF F1-Score"]), x=metrics_df["Category"],marker=dict(color="rgb(243,138,138)")))

fig.update_layout(barmode='group', height=600, width=1000, title="Model F1-Scores by Topic", bargap=.4, legend = dict(orientation="h",xanchor="right", yanchor="top",
                                                                                                                            y=1.1, x=1))
```

```
%%time
figx, axes = plt.subplots(2,2, figsize=(12, 12))

nb_AUC= ROCAUC(nb_grid, classes=data_all.target_names, per_class=True, ax=axes[0][0])
nb_AUC.fit(test_tfidf, data_test.target)
nb_AUC.score(test_tfidf, data_test.target)
axes[0][0].set(title = "Naïve Bayes ROC Curves and AUCs")
axes[0][0].legend().set_visible(False)

svm_AUC= ROCAUC(svm_rand, classes=data_all.target_names, per_class=True, ax=axes[0][1])
svm_AUC.fit(test_tfidf, data_test.target)
svm_AUC.score(test_tfidf, data_test.target)
axes[0][1].set(title = "SVM ROC Curves and AUCs")
axes[0][1].legend().set_visible(False)

rf_AUC= ROCAUC(rf_rand, classes=data_all.target_names, per_class=True, ax=axes[1][0])
rf_AUC.fit(test_tfidf, data_test.target)
rf_AUC.score(test_tfidf, data_test.target)
axes[1][0].set(title = "Random Forest ROC Curves and AUCs")
axes[1][0].legend().set_visible(False)

# xf_AUC= ROCAUC(nb_grid, classes=data_all.target_names, per_class=True, ax=axes[1][1])
# xf_AUC.fit(test_tfidf, data_test.target)
# xf_AUC.score(test_tfidf, data_test.target)
# axes[1][1].legend().set_visible(True)

figx.show()
```

```
s = [""]

s[0] = remove_stop_words_regex(s[0],custom_stopwords)
s[0] = run_lemmatizer(s[0])
s = train_vectorizer.transform(s)
s = train_tfidf_transformer.transform(s)

d = {
    "Classifier": ["Support Vector Machine", "Naïve Bayes", "Random Forest"],
    "Prediction": [data_all.target_names[svm_rand.predict(s)[0]], data_all.target_names[nb_grid.predict(s)[0]],data_all.target_names[rf_rand.predict(s)[0]]]
}
pred_df = pd.DataFrame(d)
pred_df = pred_df.set_index("Classifier")

display(pred_df)
S = csr_matrix(s)

e = { "Term":train_vectorizer.get_feature_names(),
      "TF-IDF":(pd.DataFrame.sparse.from_spmatrix(s).T)[0] }
rand_tdidf = pd.DataFrame(e)
print(" ")
(rand_tdidf[rand_tdidf["TF-IDF"] > 0]).sort_values(by=["TF-IDF"], ascending=False)
```