# Text Classification

# MLND Project Report

**Prasann Pandya**
**Prasann6897@gmail.com**

# Definition

## Project Overview

Understanding context from a given document by a computer is an age old problem under the field of natural language processing. Since my interests lie in the field of natural language processing, I decided to choose one of the fundamental tasks of natural language processing: **Document classification** or **document categorization**. Document classification is an age-old problem in information retrieval, and it plays an important role in a variety of applications for effectively managing text and large volumes of unstructured information. Automatic document classification can be defined as content-based assignment of one or more predefined categories (topics) to documents. This makes it easier to find the relevant information at the right time and for filtering and routing documents directly to users. The task is to assign a document to one or more classes or categories.

Every news website, scientific journals, online digital library, etc. needs to categorize document so that it can be easily found by users. Usually, this is done by manually screening and tagging of documents by humans. This task is not only time consuming, it can also be error prone and subject to bias as one person's categorization can be different from other person's categorization. Also, the screener may not be familiar with a particular topic. Since this task is redundant, manual and labour intensive (requires lot of reading), I decided to use Machine Learning techniques to automate this task. Machine Learning for classifying and categorizing documents can save time, improve accuracy and also remove bias.

There are many other applications for a document classification software namely: spam email filtering, genre classification (automatically determine genre of text), sentiment analysis, help librarians to categorize scientific papers by providing additional information beyond authors' keywords, assigning disease code in medical documents, etc.

There are many different approaches that have been used previously to tackle this problem previously. However, there is not standardized approach as far as I have found to classify documents. Thus, my purpose for this project is to try different supervised and unsupervised learning approaches to find the best model to classify documents.

## Problem Statement

The problem I will be working on is automatic document classification. Document classification is a problem faced by all scientific journals, news organizations and digital libraries. The software will use documents as input and the software will automatically recognize which category it is related to from a list of categories.
To solve this problem, I will be preprocessing text from each article to remove stop words and punctuations. Then, I will convert words from each article to vectors using

two different approaches (Tf-Idf and Doc2Vec). The vectors for each article will then be trained using supervised learning classifiers Naïve Bayes, Support Vector Machines and Logistic Regression. The performance of the software will be measured by how many documents it classifies correctly (accuracy) and also F1 score (precision and recall).

## Datasets and Inputs

The dataset I will be using to solve this is popular 20 Newsgroups dataset. It is one of the datasets available in sklearn. The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics. Each document is labelled to be in one of the 20 categories.

Information on how to fetch 20 Newsgroups dataset from sklearn: http://scikit-learn.org/stable/datasets/index.html#the-20-newsgroups-text-dataset

This is a list of the 20 newsgroups:

1. comp.graphics
2. comp.os.ms-windows.misc
3. comp.sys.ibm.pc.hardware
4. comp.sys.mac.hardware
5. comp.windows.x rec.autos
6. rec.motorcycles
7. rec.sport.baseball
8. rec.sport.hockey sci.crypt
9. sci.electronics
10. sci.med
11. sci.space
12. misc.forsale talk.politics.misc
13. talk.politics.guns
14. talk.politics.mideast talk.religion.misc
15. alt.atheism
16. soc.religion.christian

I will be training and testing my models using this labelled data.

## Metrics

Most text classification models are measured on accuracy. However, I want to use both Accuracy and F1 score as evaluation for the model since F1 score can provide better understanding of overall performance while taking into account the false positives and false negatives.

The equation for accuracy is given as:

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

Accuracy provides a score of number of correct predictions made over all the predictions.

The F1 score is calculated as follow:

$$F1\ Score = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

$$Precision = \frac{TP}{TP + FP}$$

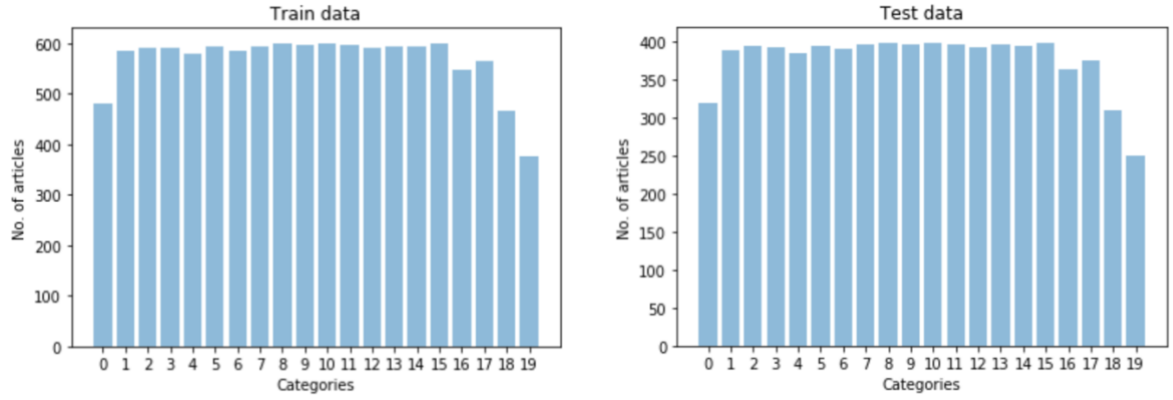
$$Recall = \frac{TP}{TP + FN}$$

Precision is measure of how many predictions are correct among all correct predictions and Recall gives a measure of how many correct predictions are selected. F1 score is used to combine both into a single score. Thus, F1 score can give a very good measure of correct classifications and how many articles are classified correctly.

# II. Analysis

## Data Exploration

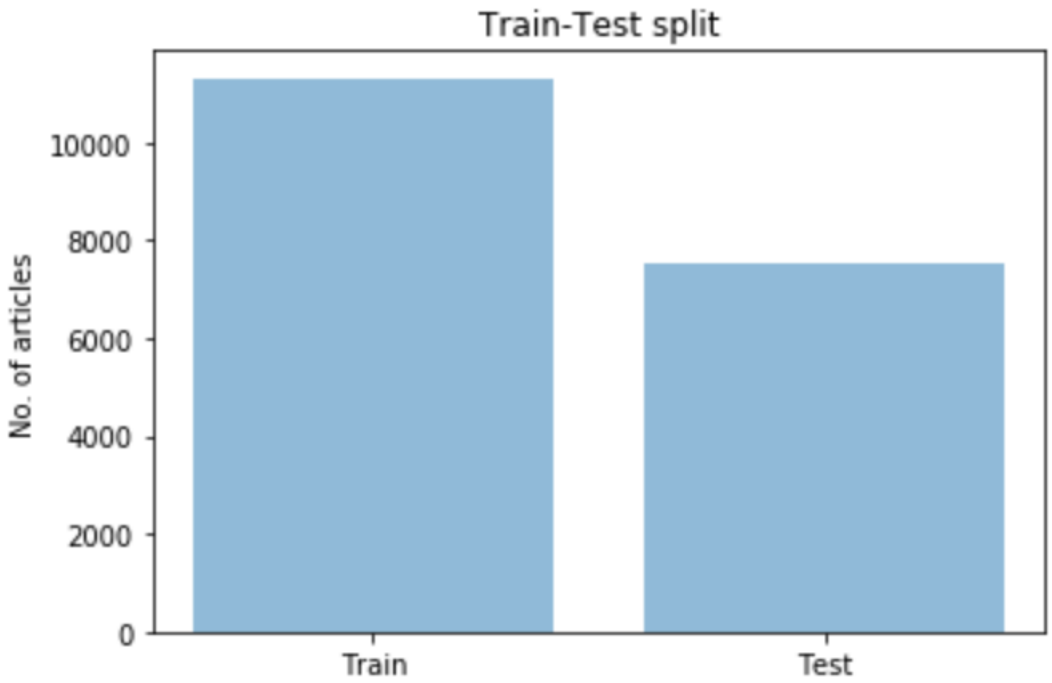
The dataset contains articles from 20 newsgroups. The data is fetched from sklearn datasets. The number of articles for each category in train and test data are shown as follows:




The categories are numbered as in the following list:

```
['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.chris
tian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc'
, 'talk.religion.misc']
```

As it can be noticed in the above plots, the number of articles for each category in test and train data are proportionally the same. The category ‘talk.religion.misc’ has less number of articles in both train and test data. Thus, there is little to no chance of bias while training data.




The train-test split in the data is 60-40. The number of articles in the training data is 11,314 and the number of articles in the test data is 7,532.

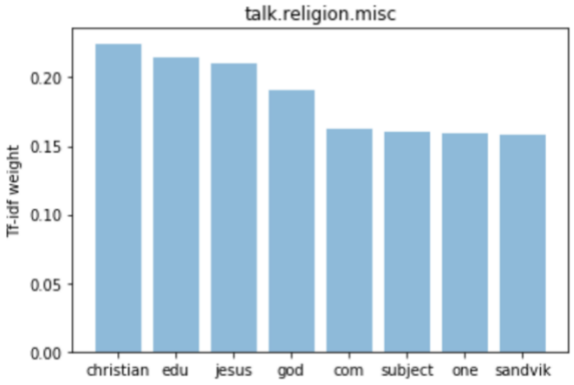
One example of news article from data related to category 'comp.sys.mac.hardware' as shown below:

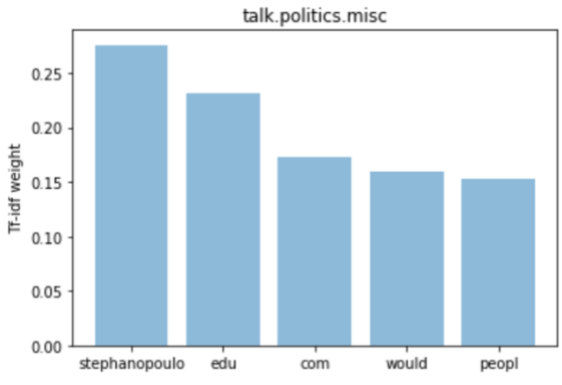
```
From: guykuo@carson.u.washington.edu (Guy Kuo)
Subject: SI Clock Poll - Final Call
Summary: Final call for SI clock reports
Keywords: SI,acceleration,clock,upgrade
Article-I.D.: shelley.1qvfo9INNc3s
Organization: University of Washington
Lines: 11
NNTP-Posting-Host: carson.u.washington.edu

A fair number of brave souls who upgraded their SI clock oscillator have
shared their experiences for this poll. Please send a brief message detailing
your experiences with the procedure. Top speed attained, CPU rated speed,
add on cards and adapters, heat sinks, hour of usage per day, floppy disk
functionality with 800 and 1.4 m floppies are especially requested.

I will be summarizing in the next two days, so please add to the network
knowledge base if you have done the clock upgrade and haven't answered this
poll. Thanks.
```

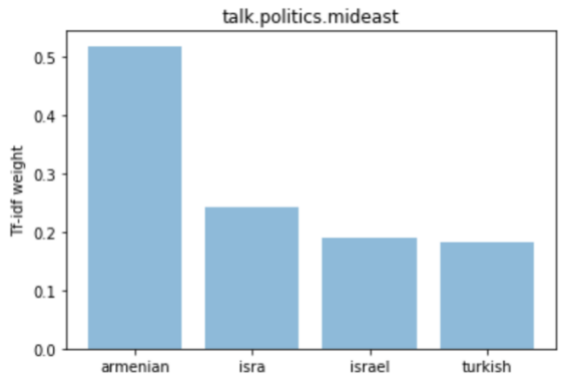
As you can notice, each article contains header such as this and content as shown above.

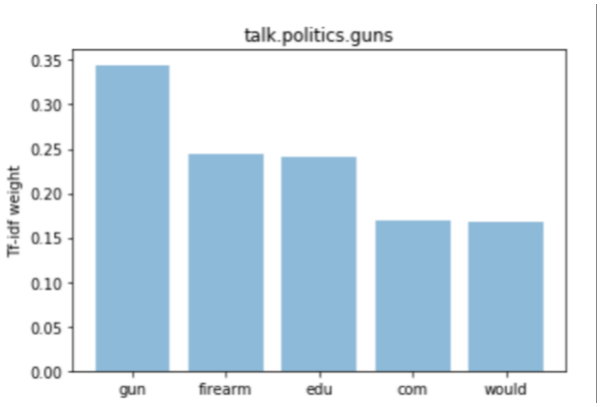
## Exploratory Visualization

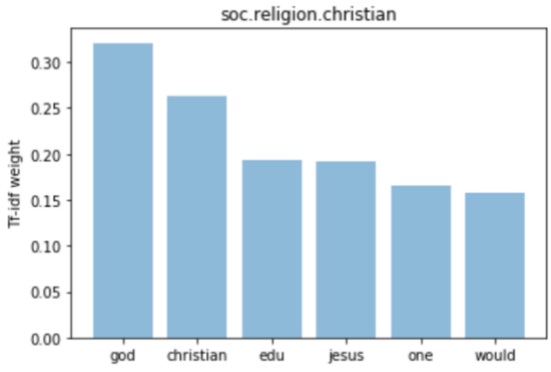
To visualize the data, I applied tf-Idf (Term frequency – Inverse document frequency) weightings to words in each category to find most important words of each category. The tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general [6]. After applying tf-idf, the most important words in each category were visualized as below:

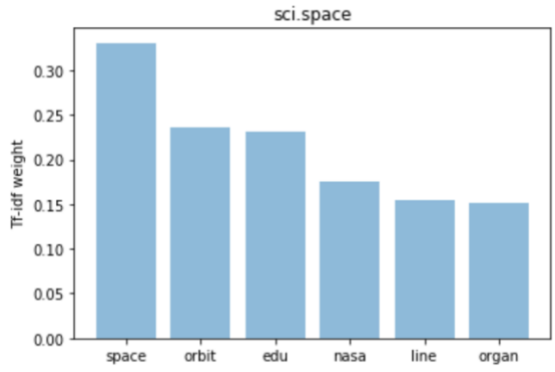

talk.religion.misc
TF-idf weight
0.00
0.05
0.10
0.15
0.20
christian
edu
jesus
god
com
subject
one
sandvik

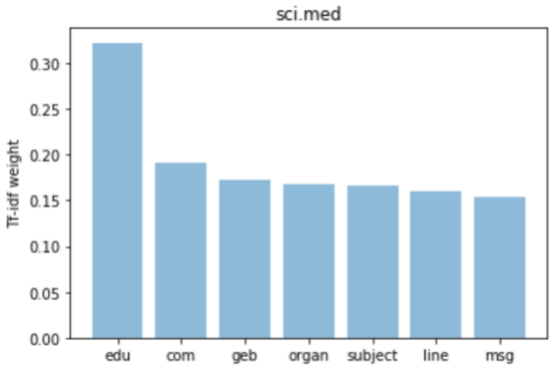

talk.politics.misc
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
stephanopoulo
edu
com
would
peopl

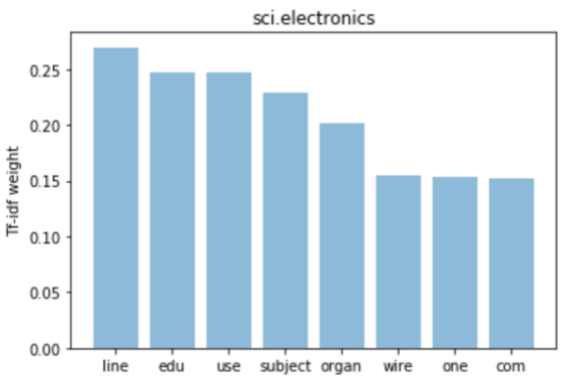

talk.politics.mideast
TF-idf weight
0.0
0.1
0.2
0.3
0.4
0.5
armenian
isra
israel
turkish

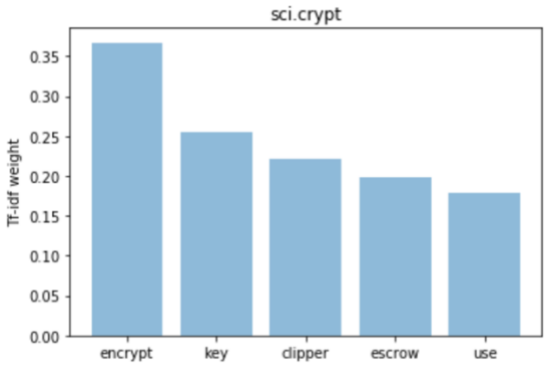

talk.politics.guns
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
gun
firearm
edu
com
would

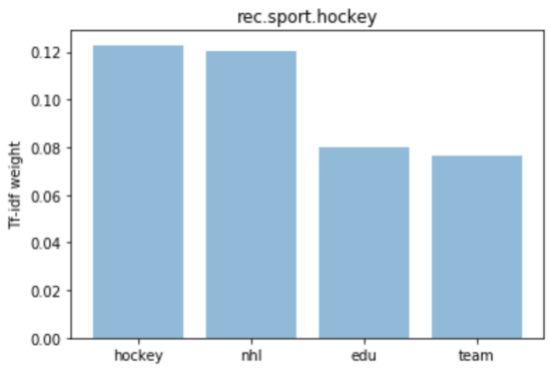

soc.religion.christian
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
god
christian
edu
jesus
one
would

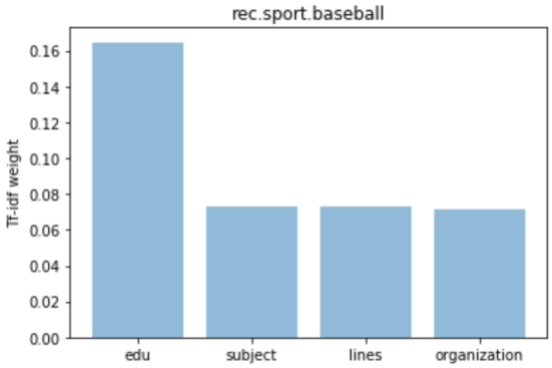

sci.space
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
space
orbit
edu
nasa
line
organ

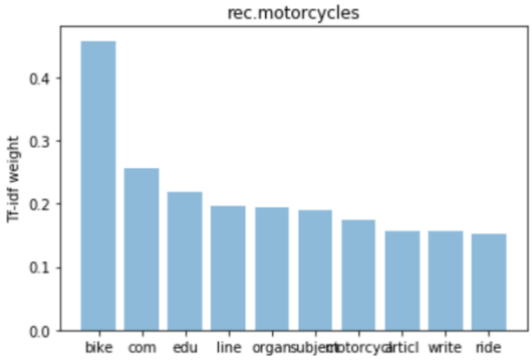

sci.med
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
edu
com
geb
organ
subject
line
msg

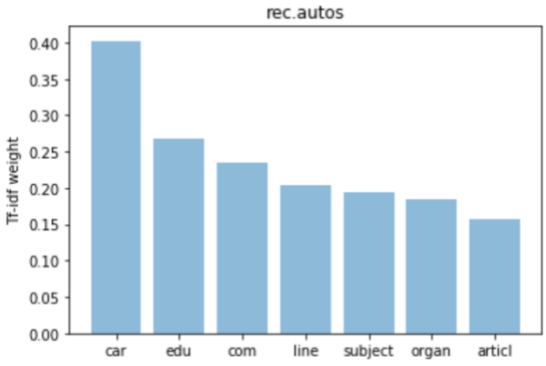

sci.electronics
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
line
edu
use
subject
organ
wire
one
com

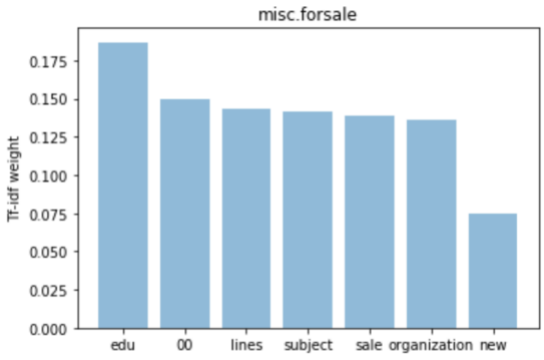

sci.crypt
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
encrypt
key
clipper
escrow
use

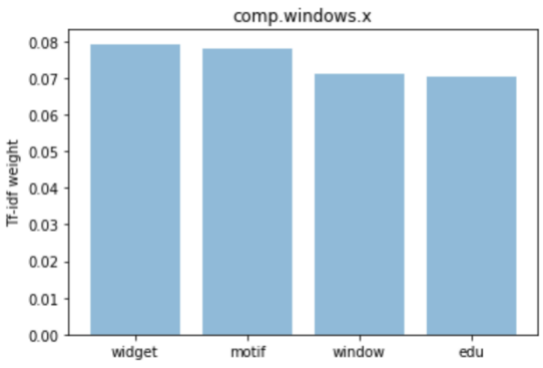

rec.sport.hockey
TF-idf weight
0.00
0.02
0.04
0.06
0.08
0.10
0.12
hockey
nhl
edu
team

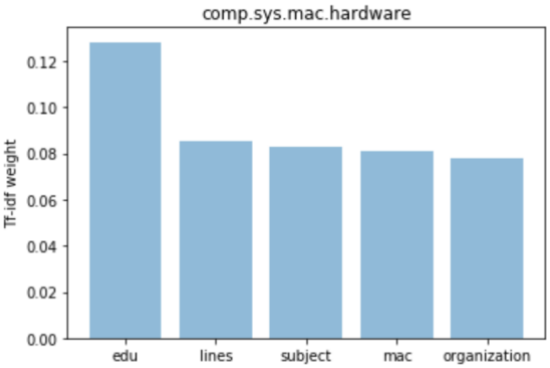

rec.sport.baseball
TF-idf weight
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
edu
subject
lines
organization

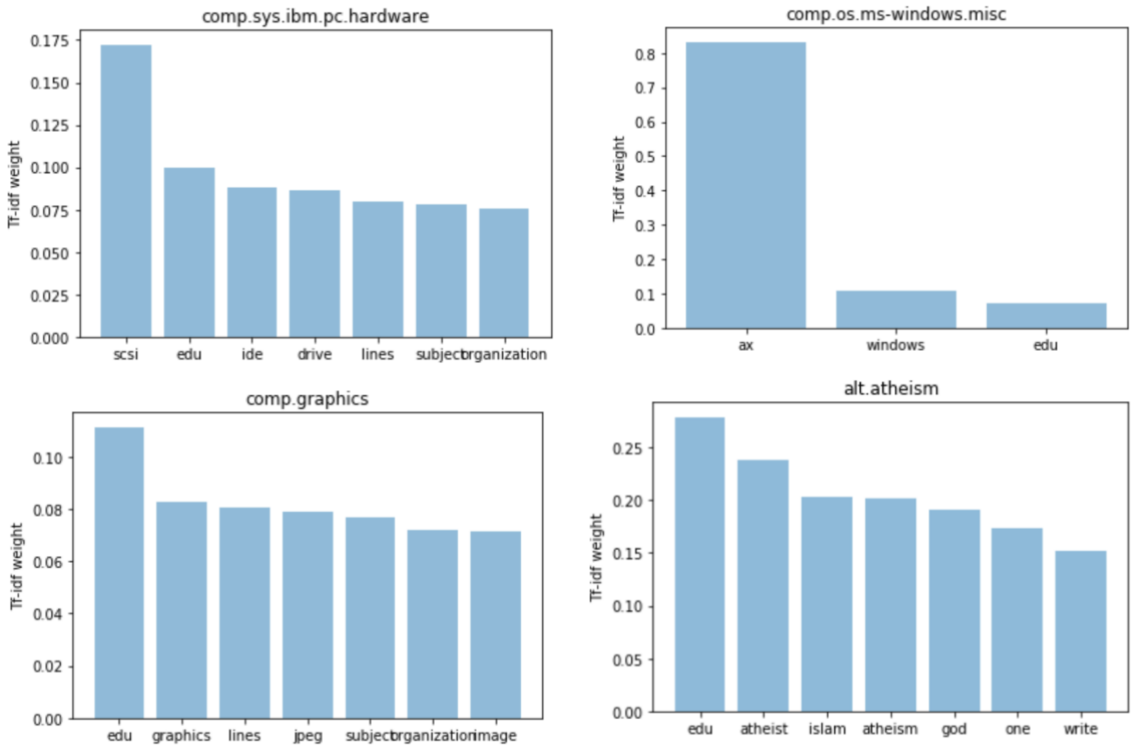

rec.motorcycles
TF-idf weight
0.0
0.1
0.2
0.3
0.4
bike
com
edu
line
organ
subject
motorcycl
articl
write
ride


rec.autos
TF-idf weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
car
edu
com
line
subject
organ
articl


misc.forsale
TF-idf weight
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
edu
00
lines
subject
sale
organization
new


comp.windows.x
TF-idf weight
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
widget
motif
window
edu


comp.sys.mac.hardware
TF-idf weight
0.00
0.02
0.04
0.06
0.08
0.10
0.12
edu
lines
subject
mac
organization




Analyzing the plots above, we can notice that there are certain words that are repeated in many categories and are picked up by tf-idf as important words. There words include “edu”, “organ”, “line”, “subject”, that are picked up in multiple categories. However, there are other words in those categories that are specific to the particular categories. The words “edu” and “organ” occurs in pretty much every category and it does not add any meaning to any category. However, in order to create a model that can be applied to any new data, it’s best not to remove those specific words. Word “god” is repeated in both “Atheism” and “Christian” categories which might lead to misclassification. There are also similar words between “Windows” and “Hardware” categories.

Thus, these plots provide good insight into steps needed from preprocessing. Also, looking at the similarities between certain categories, there can be some misclassification.

## Algorithms and Techniques

There are two techniques for converting documents into vectors that I decided to use to solve this problem:

1. **Tf-Idf (Term frequency – Inverse document frequency)**: The tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency

of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general [6].

2. **Doc2Vec**: Tf-Idf has many disadvantages. The word order is lost, and thus different sentences can have exactly the same representation, as long as the same words are used [3]. To combat this, a new approach called Doc2Vec (also called paragraph vector) was introduced. In this approach, word order is taken into consideration. The vectors of words with similar context (surrounding words) are similar and thus the word meanings are taken into account in this approach. This Doc2Vec provides numerical representation of a document which represents concept of the document. More details on Doc2Vec approach are in [3].

Once the words were converted to vectors, many different classifiers can be used for document classification. However, I trained the model using the following three classifiers which are known to word well with text data:

1. **Naïve Bayes**: Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features [7]. Naïve Bayes has been used previously for text classification tasks such as spam filtering and sentiment classification and hence I decided to use this classifier.

2. **Linear Support Vector Machines (Linear SVM)**: A linear support vector machine constructs a linear hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection [8]. After much research, I found that text is often linearly separable and SVMs are most widely used for text classification. A good visualization of Linear SVM is shown below which confirms its usefulness for classification tasks:

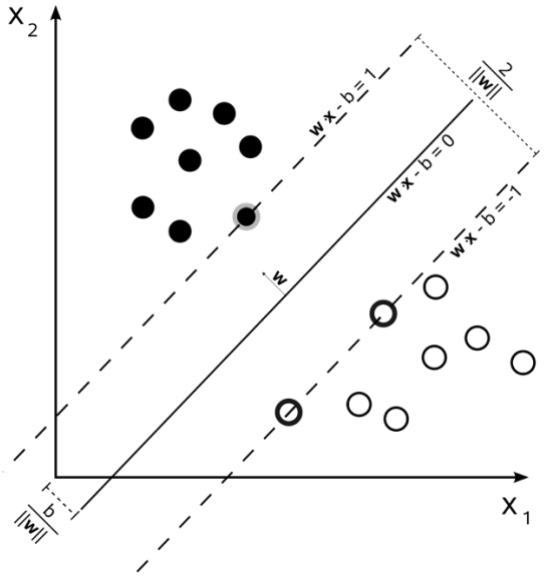



Image taken from: https://en.wikipedia.org/wiki/Support_vector_machine

3. **Logistic Regression:** Logistic regression is another technique which is very good for classification problems. Logistic Regression uses a logistic function which is an S shaped curve that can take any real-valued number and map it to a value between 0 and 1. The S shaped curve can be shown as:

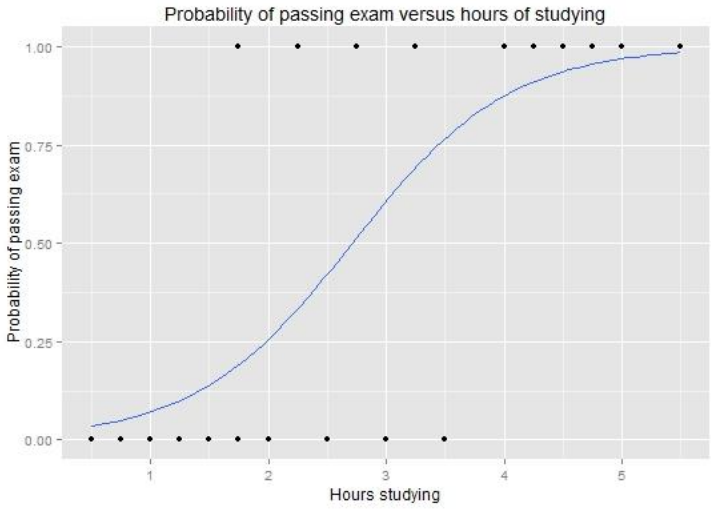



Image take from: https://en.wikipedia.org/wiki/Logistic_regression

Logistic regression gives a discrete binary outcome between 0 and 1. It does this by measuring the relationship between the dependant variable and independent variable. This give probability of a relationship which is mapped into the S curve and converted to

either “0” or “1”. Thus, this algorithm can be very effective for text classification problems. [11][12]

## Benchmark

The traditional way of performing text classification is to use a bag of words model, convert the words to vectors using TF-IDF and using a classifier mainly SVM. This usually gives an accuracy of around 90% in document classification of 2-4 categories and around 80-85% when there are more that 4 categories.

# III. Methodology

## Data Preprocessing

The data was preprocessed using a tokenizer from NLTK (Natural Language Toolkit) library which breaks down every word inside the article into a separate entry in a list. All the words were further converted into lower case. The example of words of an article after tokenization is shown below:

```
['from', 'guykuo', 'carson', 'u', 'washington', 'edu', 'guy', 'kuo', 'subject', 'si', 'clock', 'poll', 'final', 'call
', 'summary', 'final', 'call', 'si', 'clock', 'reports', 'keywords', 'si', 'acceleration', 'clock', 'upgrade', 'artic
le', 'i', 'd', 'shelley', '1qvfo9innc3s', 'organization', 'university', 'washington', 'lines', '11', 'nntp', 'posting
', 'host', 'carson', 'u', 'washington', 'edu', 'a', 'fair', 'number', 'brave', 'souls', 'upgraded', 'si', 'clock', 'o
scillator', 'shared', 'experiences', 'poll', 'please', 'send', 'brief', 'message', 'detailing', 'experiences', 'proce
dure', 'top', 'speed', 'attained', 'cpu', 'rated', 'speed', 'add', 'cards', 'adapters', 'heat', 'sinks', 'hour', 'usa
ge', 'per', 'day', 'floppy', 'disk', 'functionality', '800', '1', '4', 'floppies', 'especially', 'requested', 'i', 's
ummarizing', 'next', 'two', 'days', 'please', 'add', 'network', 'knowledge', 'base', 'done', 'clock', 'upgrade', 'ans
wered', 'poll', 'thanks']
```

## Implementation

## First Approach

### Converting Words to Vectors:

After the data was tokenized, words were converted to vectors using tf-idf. Tf-idf creates a vector with higher values for important words in a particular article. These vectors are created for each article. The vectors from each article are used as input for Naïve Bayes and SVM classifiers.

### Classification:

After the data was converted to vectors using tf-idf, the data was then used as input in the following classifiers:

1. Multinomial Naïve Bayes
2. Linear Support Vector Machines
3. Logistic Regression

I used MultinomialNB() classifier from sklearn.naivebayes library. Multinomial Naïve Bayes classifier is used specifically for text classification tasks as it is suitable for classification with discrete features [9].

Naïve Bayes has a parameter called alpha that can be changed to different values. The values for the parameter are decided using Grid Search functionality which checks for various combinations of parameter values to find the combination with best accuracy. A pipeline was developed for Naïve Bayes with CountVectorizer(), TfidfTransformer() and MultinomialNB(). The parameters for each of these steps are decided using GridSearchCV functionality from scikit-learn shown as below:

```
parameters_NB = {'vect__ngram_range': [(1, 1), (1, 2)],
                 'tfidf__use_idf': (True, False),
                 'clf__alpha': (1e-2, 1e-3),
}

gs_clf_NB = GridSearchCV(text_clf_NB, parameters_NB, n_jobs=-1)
```

The best parameters after performing grid search for Naïve Bayes pipeline were found to be ngram_range=(1,2), use_idf=True, alpha=0.01. The classifier model was then fit to the training data and tested on test data. The model accuracy found using the default values was 81%. Using the grid search parameters, the accuracy of the model on test data increased by more than 2 percent to 83%.

For the LinearSVC() classifier, a similar pipeline was developer containing CountVectorizer(), TfidfTransformer() and LinearSVC(). The parameters for these were again found using grid search to be ngram_range=(1,2), use_idf=True, C=10. The model was fit to the train data and tested on test data. After testing the model accuracy on test data using grid search found parameters, the model accuracy was found to be 86%.

Similar approach was used for Logistic Regression classifier. The accuracy was found to be around 83%.

During implementation of this approach, a number of parameters were tried. The most difficult part was deciding the parameters to use for the different classifiers. This problem was solved using Grid Search functionality of sklearn library. It was also difficult to determine the type of tokenization to perform on text. This problem was solved after doing a lot of trial and error with stemming, lemmatization, removal of stop words, etc. and using accuracy of the model to determine which processing to do.

## Second Approach

Since the tf-idf approach does not take into account order of words in an article and their semantic relationships, I decided to try out an approach called Doc2Vec which takes them into account. More details of this model are in [3].

In terms of preprocessing, the words are tokenized in a similar way. The stop words are not removed as they account to provide relationships between words which is the main purpose of using doc2vec. Sample of tokenization done using Doc2Vec is shown below:

```
['from', 'mb', 'ubvmsd', 'cc', 'buffalo', 'edu', 'neil', 'gandler', 'subject', 'need', 'info', 'on', 'bonneville', 'o
rganization', 'university', 'at', 'buffalo', 'lines', 'news', 'software', 'vax', 'vms', 'vnews', 'nntp', 'posting', '
host', 'ubvmsd', 'cc', 'buffalo', 'edu', 'am', 'little', 'confused', 'on', 'all', 'of', 'the', 'models', 'of', 'the',
'bonnevilles', 'have', 'heard', 'of', 'the', 'le', 'se', 'lse', 'sse', 'ssei', 'could', 'someone', 'tell', 'me', 'the
', 'differences', 'are', 'far', 'as', 'features', 'or', 'performance', 'am', 'also', 'curious', 'to', 'know', 'what',
'the', 'book', 'value', 'is', 'for', 'prefereably', 'the', 'model', 'and', 'how', 'much', 'less', 'than', 'book', 'va
lue', 'can', 'you', 'usually', 'get', 'them', 'for', 'in', 'other', 'words', 'how', 'much', 'are', 'they', 'in', 'dem
and', 'this', 'time', 'of', 'year', 'have', 'heard', 'that', 'the', 'mid', 'spring', 'early', 'summer', 'is', 'the',
'best', 'time', 'to', 'buy', 'neil', 'gandler']
```

Library called Gensim was used for implementing Doc2Vec [4]. It is one of the popular libraries for this purpose.

To convert all documents to vectors, the documents were assigned an id relating to their index. The vectors were then generated using doc2vec model. Each vector size was kept to 50. The vectors were trained for 25 epochs. The size of vectors and number of epochs was decided after much trial and error.

The time taken to convert documents to vectors using doc2vec was around 10-15 minutes. So, this was significantly high.

The new document vectors were then used as input into SVM and tested on test data. The accuracy of this model was around 55% which is much less than approach using tf-idf. Since doc2vec is a deep learning technique, it needs large amounts of data for each category to work well. The data in this case is not large enough for doc2vec to understand the ordering of words. Each category only has 400-500 articles and doc2vec generally needs thousands of input articles to understand the semantic relationships better. Also, the content in each article is very little which again leads to less training data. Thus, tf-idf is a better approach for this corpus.

During implementation, it was difficult to determine how to convert articles to vectors using Doc2Vec since I had never Gensim library before. Also, the documentation for this library was very limited and there were very few online tutorials. Another difficulty I faced was that it took around 10-15 minutes to train it which means it was difficult to try different parameter combinations.

## Refinement

There were a lot of refinements made to the model over time. There were many things tried through trial and error and the evaluation criteria was model accuracy and F1 scores.

**Refinement 1: Stemming**

During tokenization of words from articles, stemming was tried to reduce each word to its root form. This can help with considering words such as “babies” and “baby” as same thing. However, the result before and after stemming made little to no difference (less than 1%). So, stemming was not performed during tokenization.

**Refinement 2: Removal of stop words**

The tf-idf model was tried after with and without the removal of stop words. This also made no difference in accuracy and f1 scores (less than 1%). This can be due to the fact that tf-idf on its own is pretty good at removing common words from all categories which are generally stop words. However, for purposes of making a good model, stop words were removed in the final solution as they add no value in this approach.

**Refinement 3: Parameter tuning using Grid Search**

The parameters of classifiers were tuned using Grid Search functionality in scikit-learn. Grid search goes through all the parameter combinations to find the combination with best accuracy. The best parameters for SVM and Naïve Bayes classifiers were found to be:

```
# Best parameters for classfiers
print(gs_clf_svm.best_params_)
print(gs_clf_NB.best_params_)
```

```
{'tfidf__use_idf': True, 'clf__C': 10, 'vect__ngram_range': (1, 2)}
{'clf__alpha': 0.01, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
```

The ngram_range variable for both was (1,2). This means that taking pairs of words (bigrams) were also found to increase accuracy than just unigrams while performing tf-idf. To further test this hypothesis, I tried using just unigrams and then both unigrams and bigrams to train the models. The increase in accuracy while using bigrams was 2-3%.

**Refinement 4: Finding appropriate size of vector and number of epochs while training Doc2Vec**

The vectors for doc2vec were found using training article of each category. While training, the size of vectors and number of epochs for training were found after much trial and error to be 50 and 25 respectively. Model accuracy and f1 score were used as basis to decide these parameters for doc2vec model.

After trying out all these refinements, the data for the final model was preprocessed by removing the stop words with ngram range of 2. The tf-idf bag of words model was used to convert words to vectors. LinearSVC was used as a classifier with parameter C=10 (found using grid search).

# IV. Results

## Model Evaluation and Validation

As mentioned in the previous section, the final model is Tf-Idf bag of words model with ngram range of (1,2) which is then used as input for LinearSVC classifier with C=10. The accuracy and f1 score of this model are as shown below:

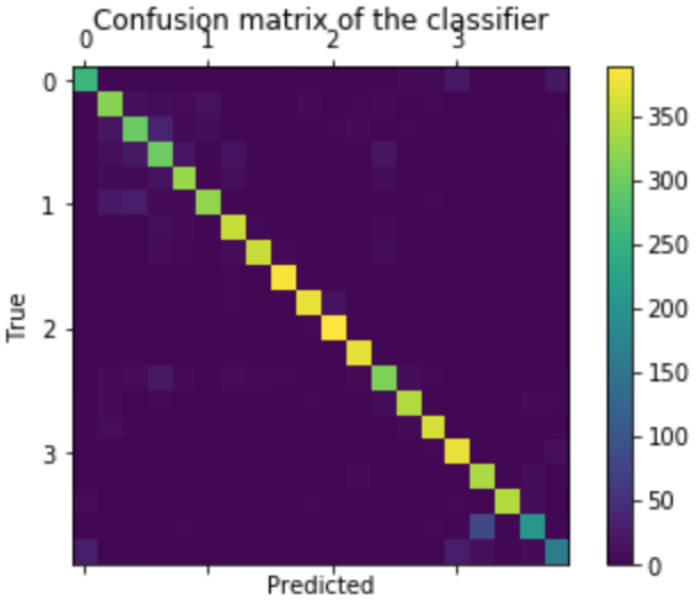
```
Accuracy: 0.861789697292
F1 Score: 0.860979294015
```

As shown above, the accuracy and f1 score of the model are 86%. This is better than any other model that I tried on the data. The detailed classification report of the model can be seen below:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| alt.atheism | 0.85 | 0.81 | 0.83 | 319 |
| comp.graphics | 0.77 | 0.81 | 0.79 | 389 |
| comp.os.ms-windows.misc | 0.77 | 0.76 | 0.76 | 394 |
| comp.sys.ibm.pc.hardware | 0.73 | 0.77 | 0.75 | 392 |
| comp.sys.mac.hardware | 0.83 | 0.85 | 0.84 | 385 |
| comp.windows.x | 0.89 | 0.82 | 0.85 | 395 |
| misc.forsale | 0.84 | 0.91 | 0.87 | 390 |
| rec.autos | 0.94 | 0.90 | 0.92 | 396 |
| rec.motorcycles | 0.95 | 0.96 | 0.95 | 398 |
| rec.sport.baseball | 0.91 | 0.94 | 0.93 | 397 |
| rec.sport.hockey | 0.94 | 0.97 | 0.95 | 399 |
| sci.crypt | 0.93 | 0.94 | 0.93 | 396 |
| sci.electronics | 0.83 | 0.79 | 0.81 | 393 |
| sci.med | 0.91 | 0.86 | 0.88 | 396 |
| sci.space | 0.91 | 0.92 | 0.91 | 394 |
| soc.religion.christian | 0.87 | 0.94 | 0.90 | 398 |
| talk.politics.guns | 0.76 | 0.93 | 0.84 | 364 |
| talk.politics.mideast | 0.97 | 0.91 | 0.94 | 376 |
| talk.politics.misc | 0.85 | 0.66 | 0.74 | 310 |
| talk.religion.misc | 0.78 | 0.66 | 0.71 | 251 |
| avg / total | 0.86 | 0.86 | 0.86 | 7532 |

As you can see from the classification report of different categories above, there are certain categories with very high precision and recall and certain with very low precision and recall. The categories autos, motorcycles, hockey, Mideast politics have good precision meaning that there is very less false positives. However, some categories like such as politics.misc and religion.misc have many false negatives leading to lower recall than precision. Overall, the f1 score of the model comes out to be 86%.

The confusion matrix of result is shown below:




```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'com
p.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']
```

The confusion matrix tells us that for most of the categories there is no overlap between categories which is surprising as there are many categories with very similar concepts such as atheism, Christian, religion and hardware, electronics. However, there is some overlap seen between category of guns and politics. There is also some overlap between categories “windows.x” and “windows.misc”.

Overall, with the exception of few categories such as religion, politics and hardware, the model performs very well on other categories.

**Testing on Unseen data**

To check for robustness of the model, the model was also tested on new data. I tried the model on a number of new sentences which I knew should be included in one category or the other. For example “Pray to Jesus” was correctly classified as under Christian category and "Toronto Maple Leafs keep losing every single year" was correctly classified under “Hockey”. Here’s a table of all the tests:

| Sentence | Classification |
|---|---|
| "Pray to Jesus" | soc.religion.christian |
| "There is war going on in Syria" | soc.religion.christian |
| "Toronto Maple Leafs keep losing every single year" | rec.sport.hockey |
| "I am atheist" | alt.atheism |
| "I don't believe in god" | soc.religion.christian |
| "Macbook has the best battery life and memory" | comp.sys.mac.hardware |
| "Summer is best time to drive a motorcycle" | rec.motorcycles |
| "I need to do exercise to improve my heart health" | sci.med |
| "All religions essentially say the same truth" | soc.religion.christian |
| "Stronger gun laws need to me made" | talk.politics.guns |
| "There have been many mass shootings in US" | sci.space |

As it can be noticed from above table, most of the categories are correctly classified which is good news. However, during testing I noticed that the use of specific keywords related to each category is very important. For example, “I am atheist” is correctly classified as under atheism category but "I don't believe in god" is classified under Christian category instead of atheism. This is one of the drawbacks of Tf-Idf bag of words model as there is no understanding of semantics of sentences by the model. Also, some models such as space misclassifies a sentence with shootings as under the space category. Also due to many similar categories, there is some overlap. For example the phrase “All religions are essentially the same” should be under religion.misc but it is classifies as religion.christian.

Thus, the model performs well when there are specific keywords mentioned in the piece of text related to a particular category. The model does not understand the sematic meaning of the sentence and model cannot distinguish well between similar categories with small differences.

## Justification

The benchmark model for 20 newsgroups dataset trained for all 20 categories has accuracy of around 80-85% as show in [10]. My approach provides 86% accuracy and f1 score which is among the high end of what has been achieved previously. There have also been high accuracy (90%) achieved previously for models trained on small number of categories which are very different from each other. For example, I tried the same model on 'alt.atheism', 'rec.autos', 'talk.politics.mideast' and 'rec.sport.hockey' categories which are very distinct from each other. This gave me the accuracy and f1 score of around 97%.

```
print(svm_accuracy)
print(svm_f1_score)
```

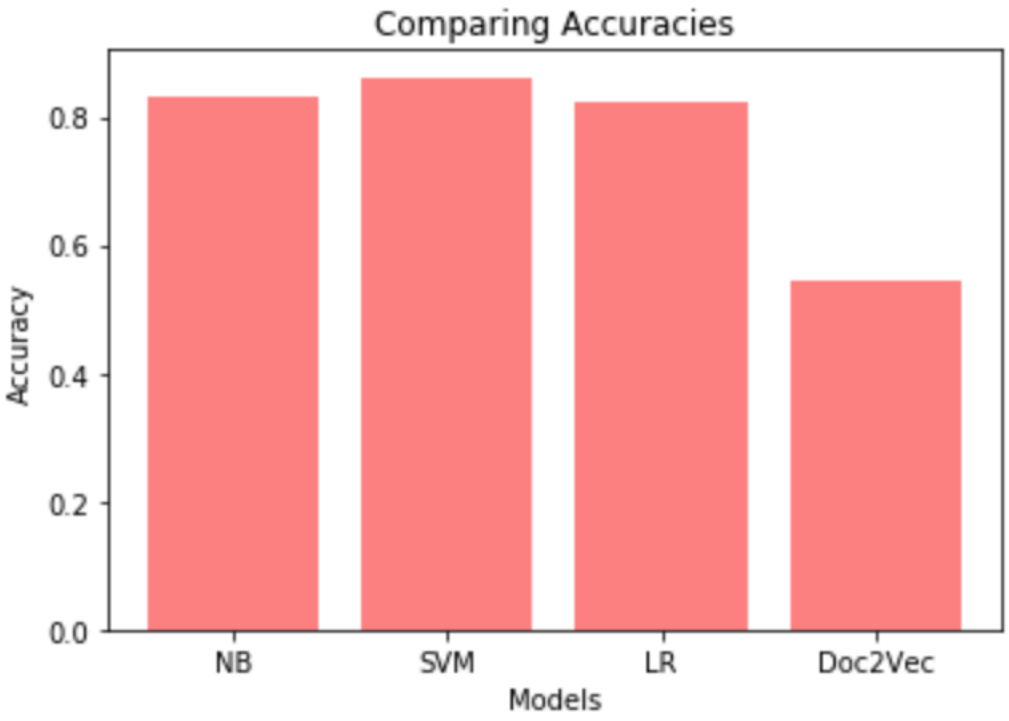
```
0.974496644295
0.974391665593
```

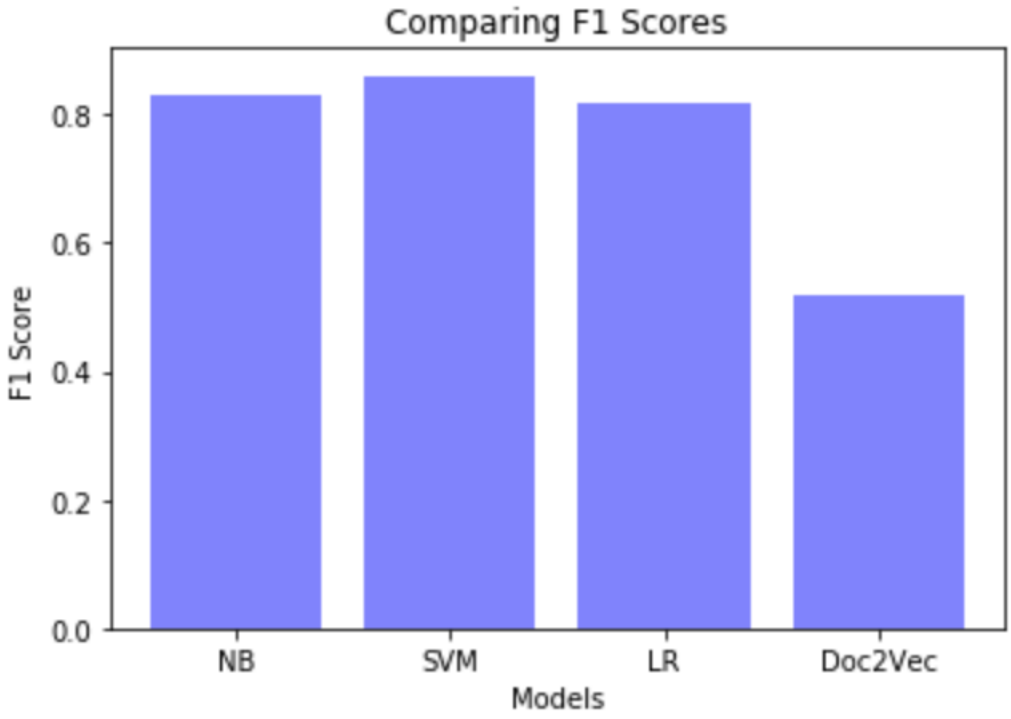
Thus, since many of the 20 categories are very similar, an accuracy of 86% is satisfactory.

### V. Conclusion

## Free-Form Visualization

For the Free-Form visualization, I chose to compare different approaches I tried and used for this project. The plots below show the comparison between different models in terms of the accuracy and f1 score. The first three models are Naïve Bayes, SVM and Logistic Regression which are widely used classifiers for text classification and are trained using tf-idf vectors. The fourth one is Doc2Vec trained vectors used as input into Linear SVM classifier. You can clearly notice that the performance of Doc2Vec is far worse than other approaches. The first three approaches give very similar results and SVM is clearly the best one out of all four.


Comparing Accuracies
Accuracy
0.0
0.2
0.4
0.6
0.8
NB
SVM
LR
Doc2Vec
Models


Comparing F1 Scores
F1 Score
0.0
0.2
0.4
0.6
0.8
NB
SVM
LR
Doc2Vec
Models

## Reflection

Overall, the entire project involved a lot of trial and error. The project also required a lot of reading of online blog posts to get familiar with NLP terminologies, models, techniques and classifiers. I started the project with preprocessing the data by removing stop words, tokenizing the words and making all words lower case. After preprocessing of the data, I used two different approached to convert words from each article to vectors. First approach was based on keywords or main concepts which second was based on semantics and ordering of the words and their relationships. A number of classifiers were tried on the tf-idf approach and Linear SVM was found to be the best among them. The same classifier was also used for the Doc2Vec approach. Tf-idf approach was found to be much better and Linear SVM classifier with input vectors from tf-idf was the best model chosen. The best classifier parameters were then found using grid search. The final chosen model was then thoroughly evaluated using classification report and confusion matrix. The final model was also evaluated on new unseen data to check the robustness and sensitivity. To justify the model, it was compared with benchmarks I found online on from people who have tried the text classification on same data before. Since the model gave similar and in some cases better accuracies than previously tried, and since it classified most of the new data very well, I am satisfied with the final model and my approach.

## Improvement

There are many new techniques for text classification and natural language processing which I did not know how to implement properly as there was no good documentation online. There are techniques for converting words to vectors called Glove and Word2Vec which take into account the semantics and context of words in an article. But these techniques were mostly presented in research papers and so it was hard to understand how to implement them properly for text classification task. There is also something called “InferSent” from facebook which is another very recent technique for converting words to vectors and it puts more weight on the main words in a sentence. These methods or combination of them with methods I used might have improved the performance of my final model but since there was no good documentation on how to implement them, I decided to go instead with conventional approaches.

## References


1. Evaluation of Text Classification (Stanford NLP Group): https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-text-classification-1.html
2. Working with text data: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
3. Distributed Representations of Sentences and Documents: https://cs.stanford.edu/~quocle/paragraph_vector.pdf
4. Gensim Doc2Vec tutorial: https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-IMDB.ipynb
5. Document Classification Wikipedia: https://en.wikipedia.org/wiki/Document_classification
6. Tf-Idf Wiki: https://en.wikipedia.org/wiki/Tf–idf
7. Naïve Bayes Wiki: https://en.wikipedia.org/wiki/Naive_Bayes_classifier
8. Linear SVM: https://en.wikipedia.org/wiki/Support_vector_machine
9. Multinomial Naïve Bayes: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
10. 20 newsgroups dataset benchmark model: https://nlp.stanford.edu/wiki/Software/Classifier/20_Newsgroups
11. Logistic Regression: https://towardsdatascience.com/the-logistic-regression-algorithm-75fe48e21cfa
12. Logistic Regression: https://machinelearningmastery.com/logistic-regression-for-machine-learning/