

IBM PROJECT AI101
GROUP -3
FAKE NEWS
DETECTION USING
NLP

Fake news detection is a hot topic in the field of natural language processing. In this article, we are using this [dataset](#) for news classification using NLP techniques. We are given two input files. One with real news and the other one with fake news. Let us dig further into the given data.

| | title | text | subject | date |
|---|---|---|--------------|-------------------|
| 0 | As U.S. budget fight looms, Republicans flip t... | WASHINGTON (Reuters) - The head of a conservat... | politicsNews | December 31, 2017 |
| 1 | U.S. military to accept transgender recruits o... | WASHINGTON (Reuters) - Transgender people will... | politicsNews | December 29, 2017 |
| 2 | Senior U.S. Republican senator: 'Let Mr. Muell... | WASHINGTON (Reuters) - The special counsel inv... | politicsNews | December 31, 2017 |
| 3 | FBI Russia probe helped by Australian diplomat... | WASHINGTON (Reuters) - Trump campaign adviser ... | politicsNews | December 30, 2017 |
| 4 | Trump wants Postal Service to charge 'much mor... | SEATTLE/WASHINGTON (Reuters) - President Donal... | politicsNews | December 29, 2017 |

real.head

| | title | text | subject | date |
|---|--|---|---------|-------------------|
| 0 | Donald Trump Sends Out Embarrassing New Year'... | Donald Trump just couldn't wish all Americans ... | News | December 31, 2017 |
| 1 | Drunk Bragging Trump Staffer Started Russian ... | House Intelligence Committee Chairman Devin Nu... | News | December 31, 2017 |
| 2 | Sheriff David Clarke Becomes An Internet Joke... | On Friday, it was revealed that former Milwauk... | News | December 30, 2017 |
| 3 | Trump Is So Obsessed He Even Has Obama's Name... | On Christmas day, Donald Trump announced that ... | News | December 29, 2017 |
| 4 | Pope Francis Just Called Out Donald Trump Dur... | Pope Francis used his annual Christmas Day mes... | News | December 25, 2017 |

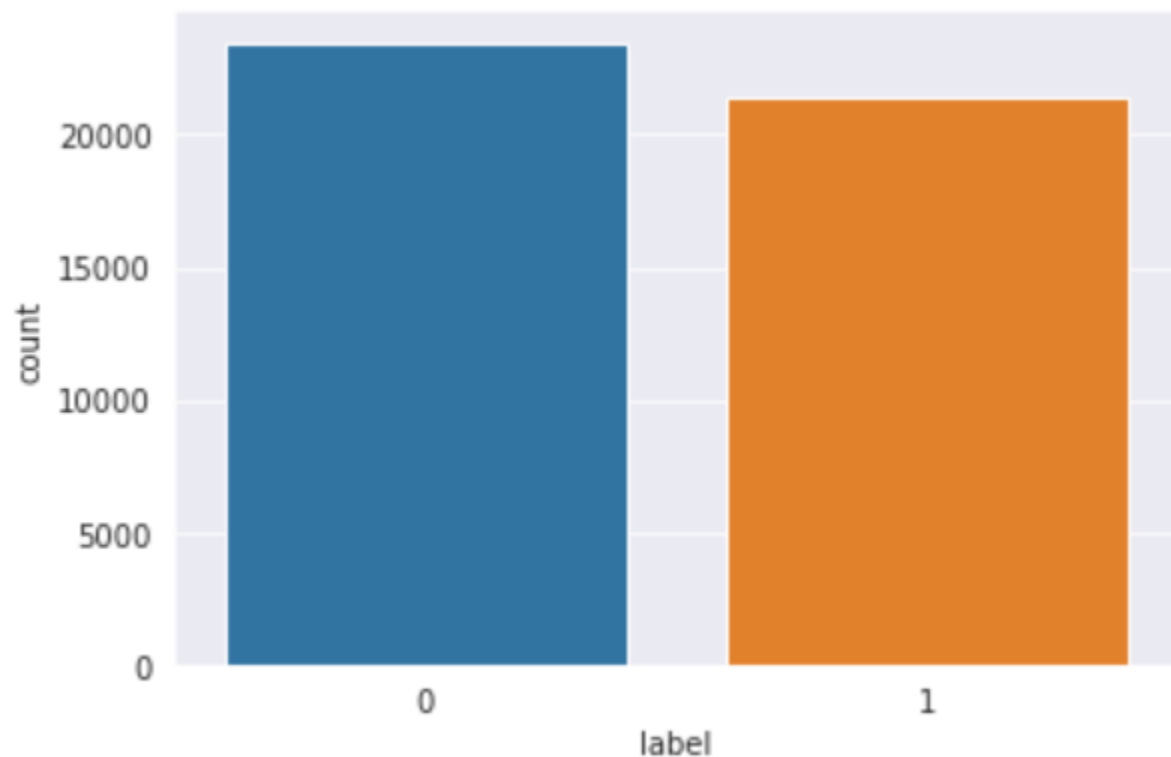
fake.head

Our job is to create a model which predicts whether a given news is real or fake. As this is a supervised learning problem, we are creating a target column named 'label' in both real and fake news data and concatenating them.

```
real['label'] = 1
fake['label'] = 0
data = pd.concat([real, fake])
```

Now we have the input where real news has the value of label as 1 and fake news have the value of label as 0. We have to check whether our data is balanced. We use seaborn library to plot the counts of real and fake news.

```
import seaborn as sns
sns.set_style("darkgrid")
sns.countplot(data['label']);
```



countplot for 'label' column

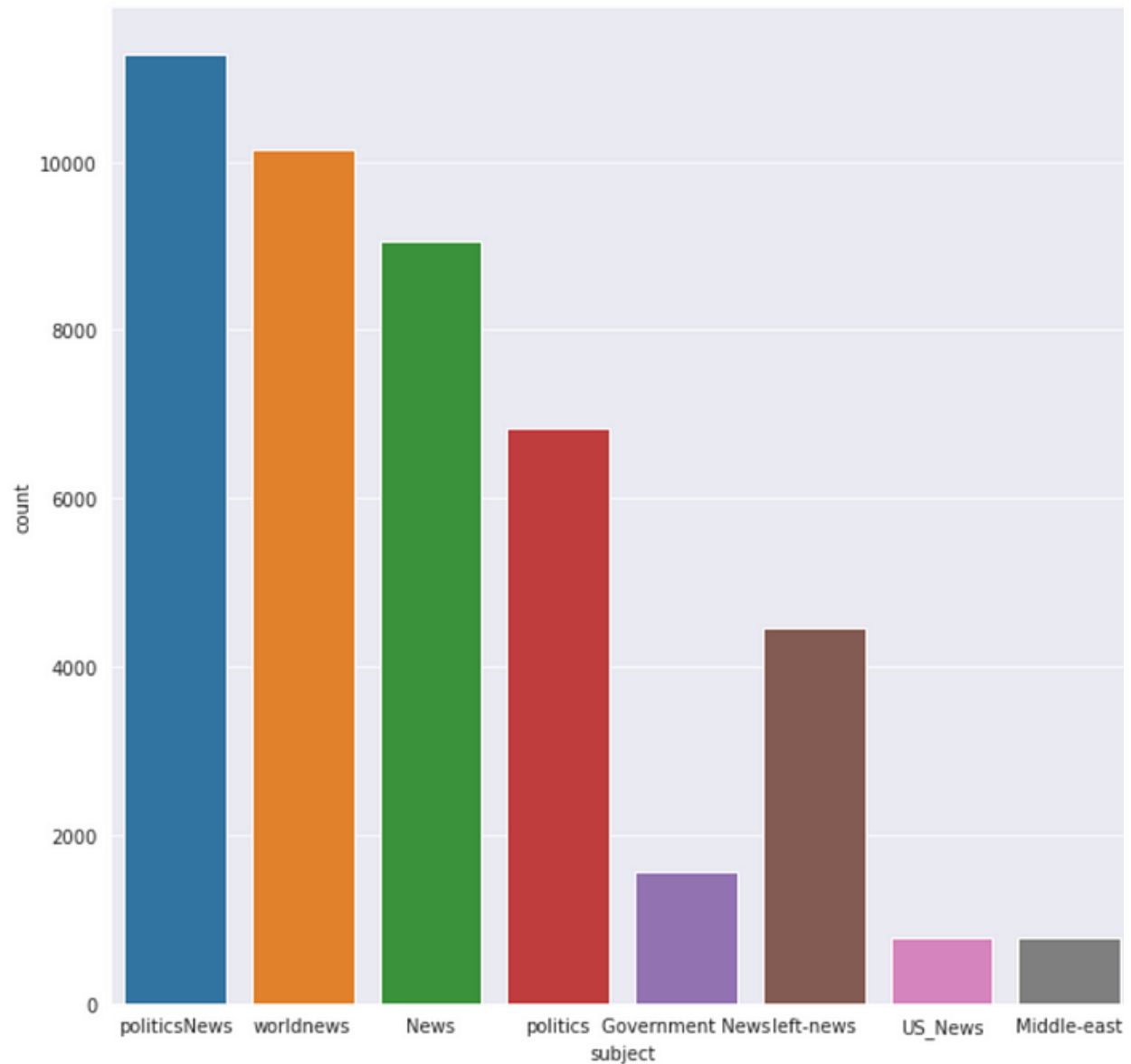
We can conclude from the plot that the data is balanced. This is an important step because there are lot of real world datasets that are imbalanced. Let us check for null values in the data.

```
data.isnull().sum()
```

```
title      0
text       0
subject    0
date       0
label      0
dtype: int64
```

Luckily, we have no null values. Then, we look at all the columns in the data. There are 5 columns in the data- title, text, subject, date and label. Let us examine the subjects.

```
import matplotlib.pyplot as plt
data['subject'].value_counts()
plt.figure(figsize = (10,10))
sns.set_style("darkgrid")
sns.countplot(data['subject']);
```

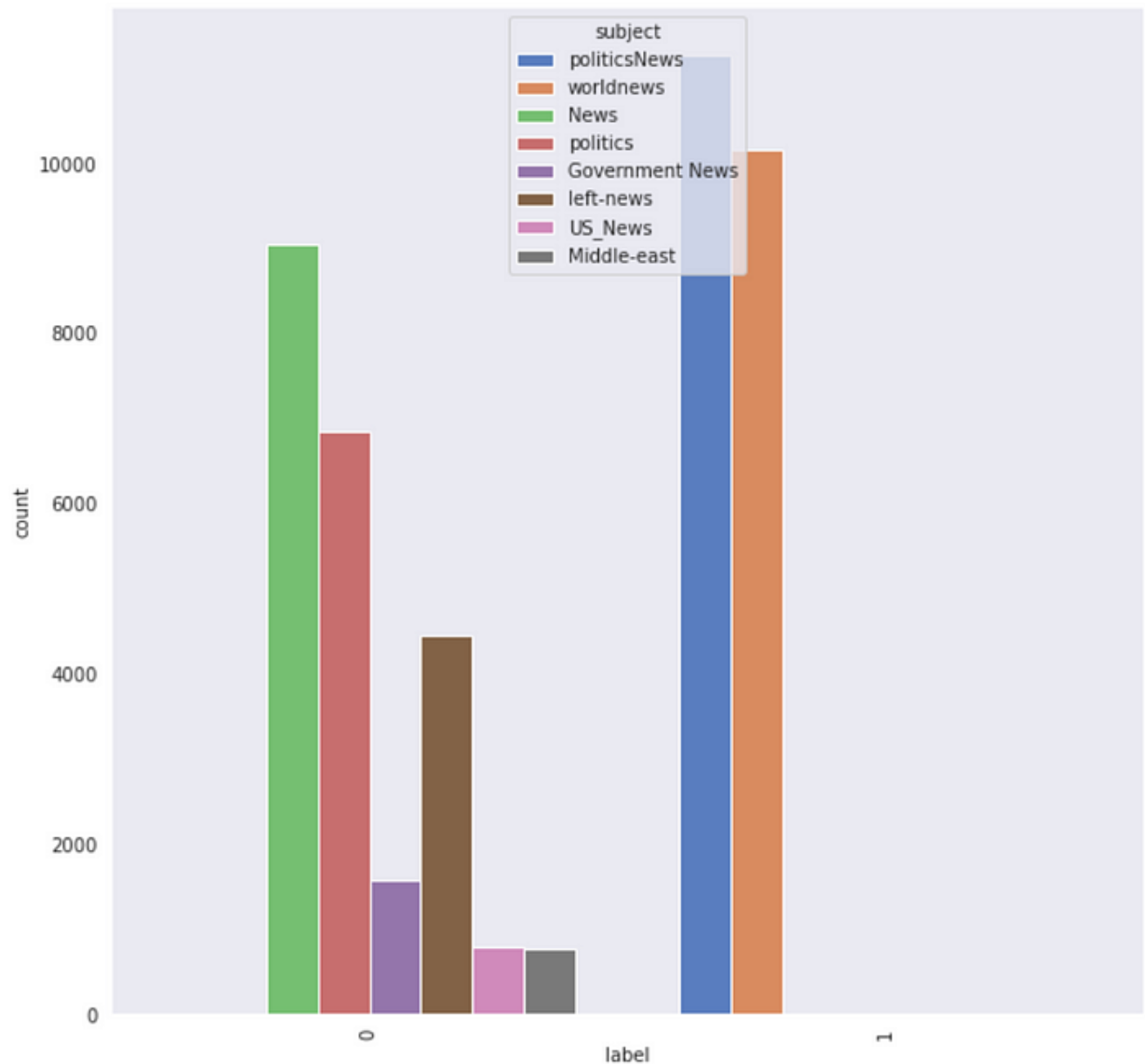


Countplot of subjects

There are news about 8 subjects. We have the largest number of news from politicsNews. Let us dig further into this.

```
plt.figure(figsize = (10,10))  
sns.set_style("dark")
```

```
chart = sns.countplot(x = "label", hue = "subject" , data = data , palette =  
'muted')  
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
```



Contplot of subject with label

It is clear from the plot that all our real news belongs to 2 subjects. That seems to be strange. It might be because our data is taken only from a small period of time. Let us concatenate title and text fields into one column and drop all other columns.

```
data['text'] = data['title'] + " " + data['text']  
data = data.drop(['title', 'subject', 'date'], axis=1)
```

Now, let us create a word cloud to analyse the most frequent words in our data. The stop words are removed from the data, and the word clouds are generated. Stop words are commonly used words in a language. Search engines ignore the stop words while indexing data as well as retrieving results for search queries.

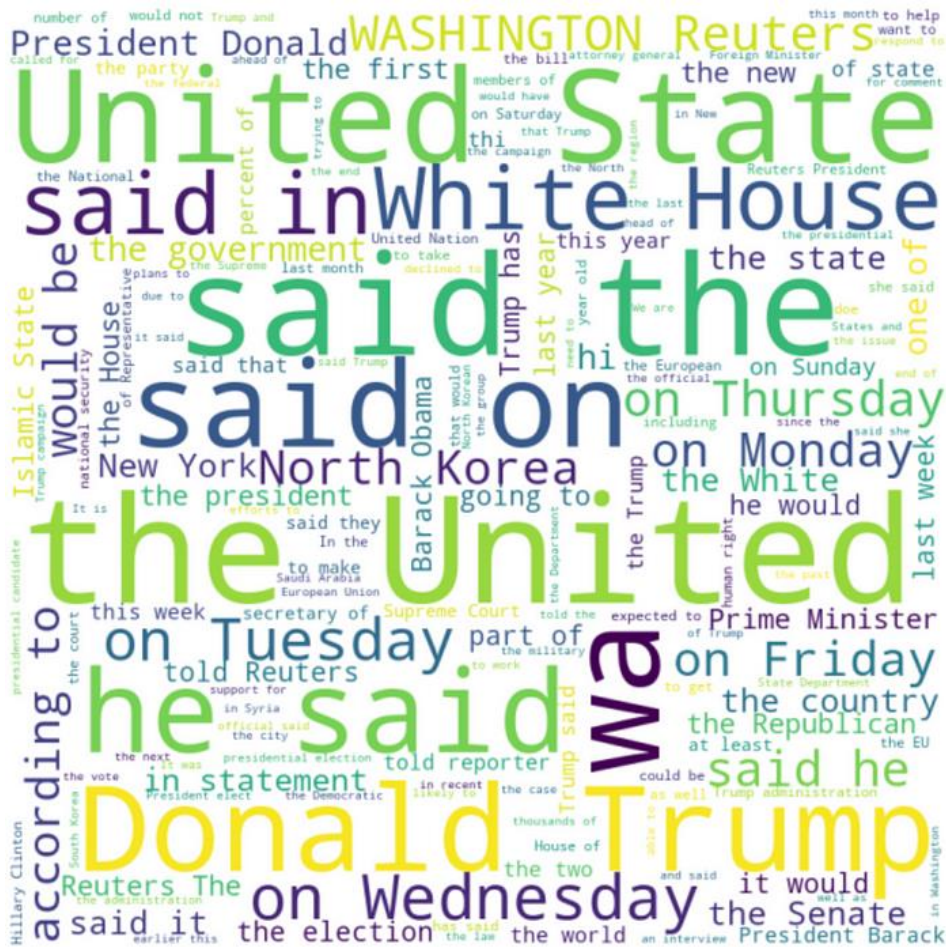
```
from nltk.corpus import stopwords  
from wordcloud import WordCloud  
  
wordcloud = WordCloud(width = 800, height = 800,  
                        background_color = 'white',  
                        stopwords = stopwords.words('english'),  
                        min_font_size = 10).generate(" ".join(data[data['label'] == 0].text))  
  
# plot the word cloud for fake news data  
plt.figure(figsize = (8, 8), facecolor = None)  
plt.imshow(wordcloud)  
plt.axis("off")  
plt.tight_layout(pad = 0)  
plt.show()
```



```
wordcloud = WordCloud(width = 800, height = 800,  
    background_color = 'white',  
    stopwords = stopwords.words('english'),  
    min_font_size = 10).generate(" ".join(data[data['label'] ==  
1].text))
```



```
# plot the WordCloud image for genuine news data
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



word cloud for real news

We can conclude from the word clouds that there are lot of real and fake news about Donald Trump and United States.

Classification

Now, let us move into the classification models. We will try different models and evaluate the performance. As classification is a supervised learning, we have to first split the data into training and test data. We train the model using the train data, and test the performance of our model using test data. Generally, the data is split in such a way that we have 80% of our data in train set, and 20% of our data in test set. This is because of the fact that the more the training set, the more the model learns from the data.

#splitting data for training and testing

import sklearn

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test =

train_test_split(data['text'],data['label'],test_size=0.2, random_state = 1)

Now, we will look into some basic classification models.

- **MultiNomial Naive Bayes**

Naive Bayes are mostly used in natural language processing. Naive Bayes classifier algorithm is a family of algorithms which use [Bayes Theorem](#). It uses the naive assumption that all the features are independent of each other. Bayes theorem calculates the probability $P(c|x)$ where c is the class of possible outcomes and x is the given instance which has to be classified.

$$P(c|x) = P(x|c) * P(c) / P(x)$$

According to our data, the class is 0 or 1, where 0 implies fake news and 1 implies true news. Given a news x , we will compute $P(\text{true news}|x)$ as well as $P(\text{fake news}|x)$. If $P(\text{true news}|x) > P(\text{false news}|x)$, the algorithm predicts it is a true news. Otherwise, the news will be predicted as fake.

- **Support Vector Machine**

Support Vector Machine or SVM is a linear model for classification and regression problems. SVM model takes the data in the training set, and maps it to data points in space so that there is a clear gap between points belonging to different categories. This gap is made as wide as possible to improve the performance of the model. Whenever a new data point is given to the model, it maps the point to the same space, and predict the category based on the side of the gap on which they fall.

- **Passive Aggressive Classifier**

Passive aggressive classifier is an online algorithm that learns from massive streams of data. The idea is to get an example, update the classifier, and throw away the example. It is fast and easy to implement, but does not provide global guarantees like SVM.

Now we can apply these models to our data. But, we cannot give the text directly as an input to the classifier. Instead, we will convert the text to numbers. Machine learning uses a simple model called bag-of-words to deal with text data. The idea is to find all the unique words in the

document, and create a vector of size equal to the number of unique words. Each word is assigned an index in the vector. The index corresponding to the word is filled with the frequency of that word in the document. The main drawback with this approach is that it ignores all the information related to the order of the words, and only takes into account the frequency of the words. We are using [CountVectorizer](#) and [TfidfTransformer](#) for the transformation.

Count Vectorizer

The count vectorizer tokenizes a collection of documents and builds a vocabulary of unique words. It can also encode new documents using this vocabulary.

Tfidf Transformer

Consider the following set of words-‘the’, ‘in’, ‘on’, ‘a’, ‘an’. This is an example of a set of words which doesn’t have any meaning on its own, but they occur a lot in every document. Including these kind of words in our algorithm reduces the performance of the model. So, we have to eliminate these words. TF-IDF or *Term Frequency-Inverse Document Frequency* is used to remove such words. Term frequency refers to the frequency of the word in a document. Inverse document frequency reduces the scores of the words that appear too much across all the documents. In short, TF-IDF give frequency score to words by highlighting the ones which are more frequent in a document, but not across documents. The TfidfVectorizer tokenizes documents, learns the

vocabulary and inverse document frequency scores. It also encode new documents. Alternately, if you have already used CountVectorizer, you can use it with a [TfidfTransformer](#) to just calculate the inverse document frequencies and start encoding documents.

We use [Pipeline](#) object, a utility used to automate machine learning workflow. A pipeline allows several transformers to be chained together. Data flows from the start of the pipeline to the end, and the output of each transformer is given as the input of the next transformer. A pipeline has two main methods:

- `fit_transform`: this method is called for each transformer and each time the result is fed into the next transformer.
- `fit_predict`: if your pipeline ends with an estimator, `fit_predict` is called on the estimator.

There are several metrics to evaluate the performance of a model. In this project, we use accuracy and confusion matrix. Accuracy is the ratio of number of correct predictions to the total number of predictions. Accuracy is considered as a good metrics only if your data is balanced. Confusion matrix is a matrix which gives more insights into our model. It compares predicted values and the actual values. We use 4 measures to evaluate the performance.

- True positive: The cases in which the predicted values and the actual values are the same, and the value is positive.

- True Negative: The cases in which the predicted values and the actual values are the same, and the value is negative.
- False Positive: The cases in which the prediction is 'YES' ,but the actual value is 'NO'.
- False Negative: The cases in which the prediction is 'NO', but the actual value is 'YES'.

Finally, let us apply various models and evaluate the performance.
Multinomial NB

```
#Multinomial NB
```

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
```

```
pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB())
])
```

```
model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
```

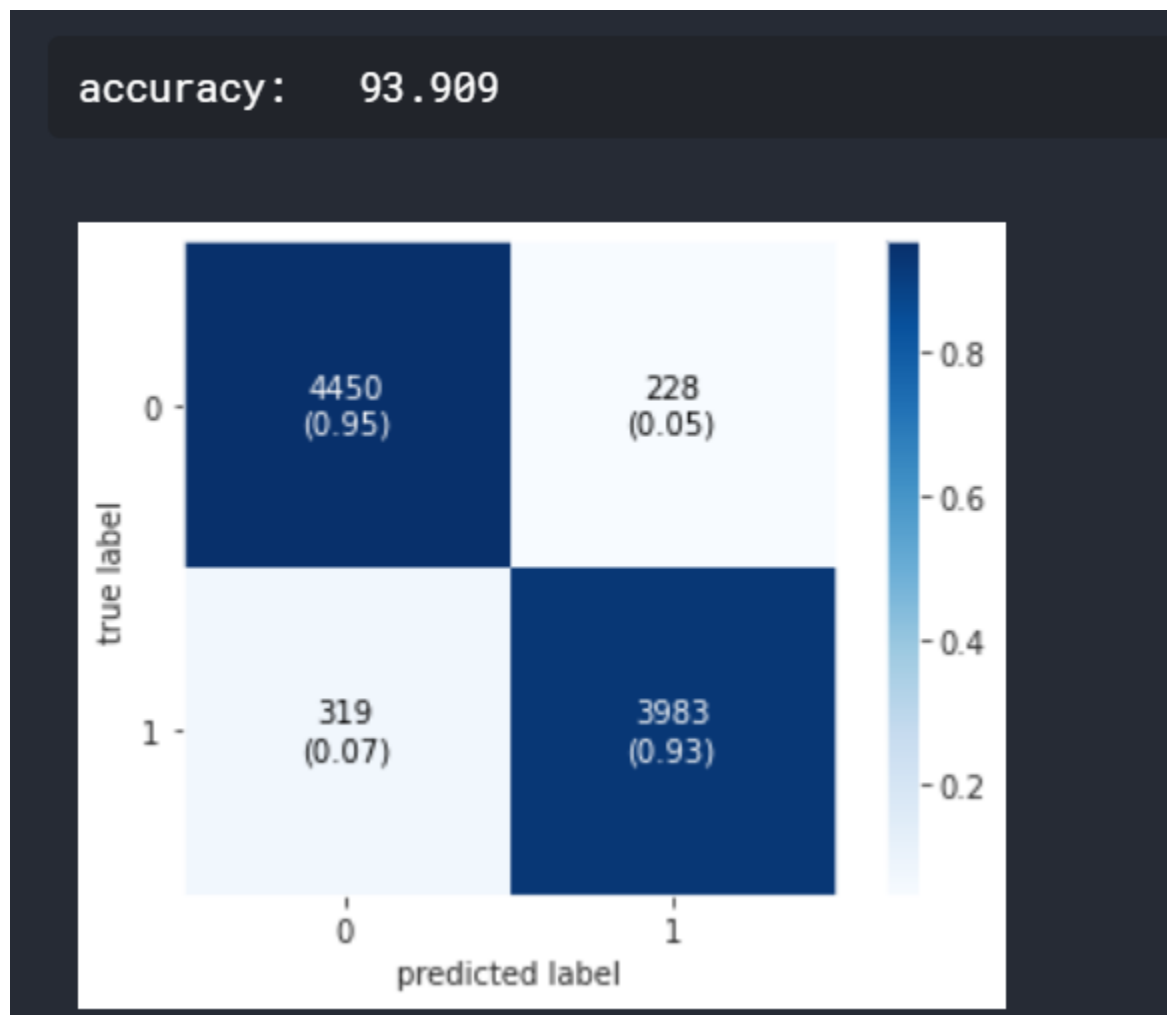
```
score = metrics.accuracy_score(y_test, prediction)
```

```
print("accuracy: %0.3f" % (score*100))  
cm = metrics.confusion_matrix(y_test, prediction, labels=[0,1])
```

```
fig, ax = plot_confusion_matrix(conf_mat=confusion_matrix(y_test,  
prediction),
```

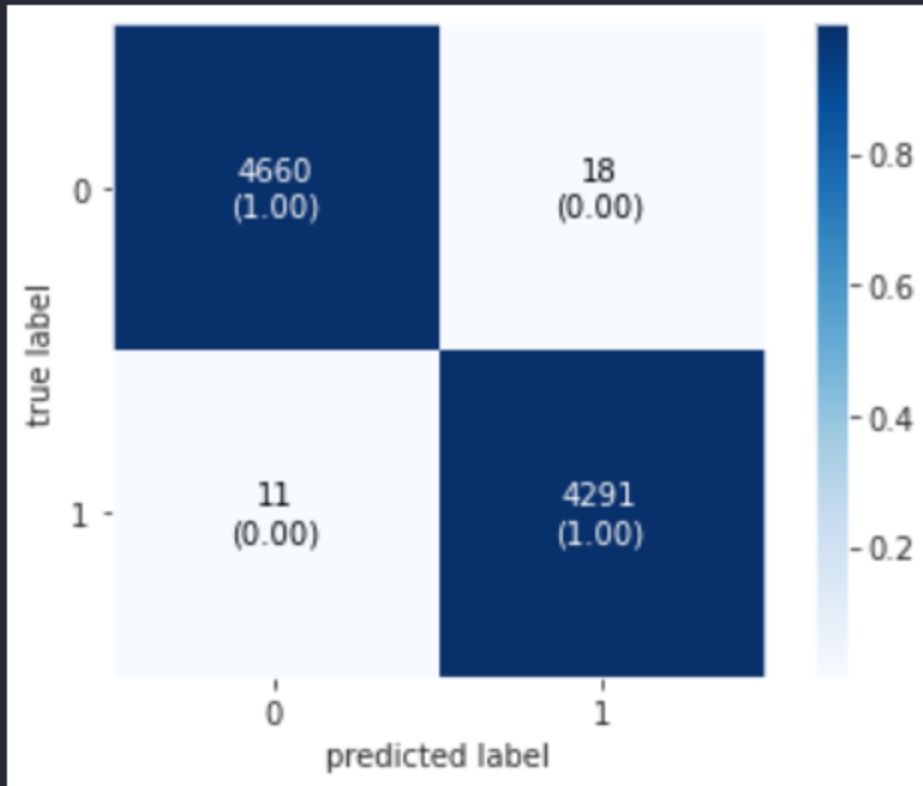
```
    show_absolute=True,  
    show_normed=True,  
    colorbar=True)
```

```
plt.show()
```



Confusion Matrix

accuracy: 99.677



SVM

```
#Passive Aggressive Classifier
from sklearn.linear_model import PassiveAggressiveClassifier
pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', PassiveAggressiveClassifier())
])

model = pipe.fit(x_train, y_train)
```

```
prediction = model.predict(x_test)
```

```
score = metrics.accuracy_score(y_test, prediction)
```

```
print("accuracy:  %0.3f" % (score*100))
```

```
cm = metrics.confusion_matrix(y_test, prediction, labels=[0,1])
```

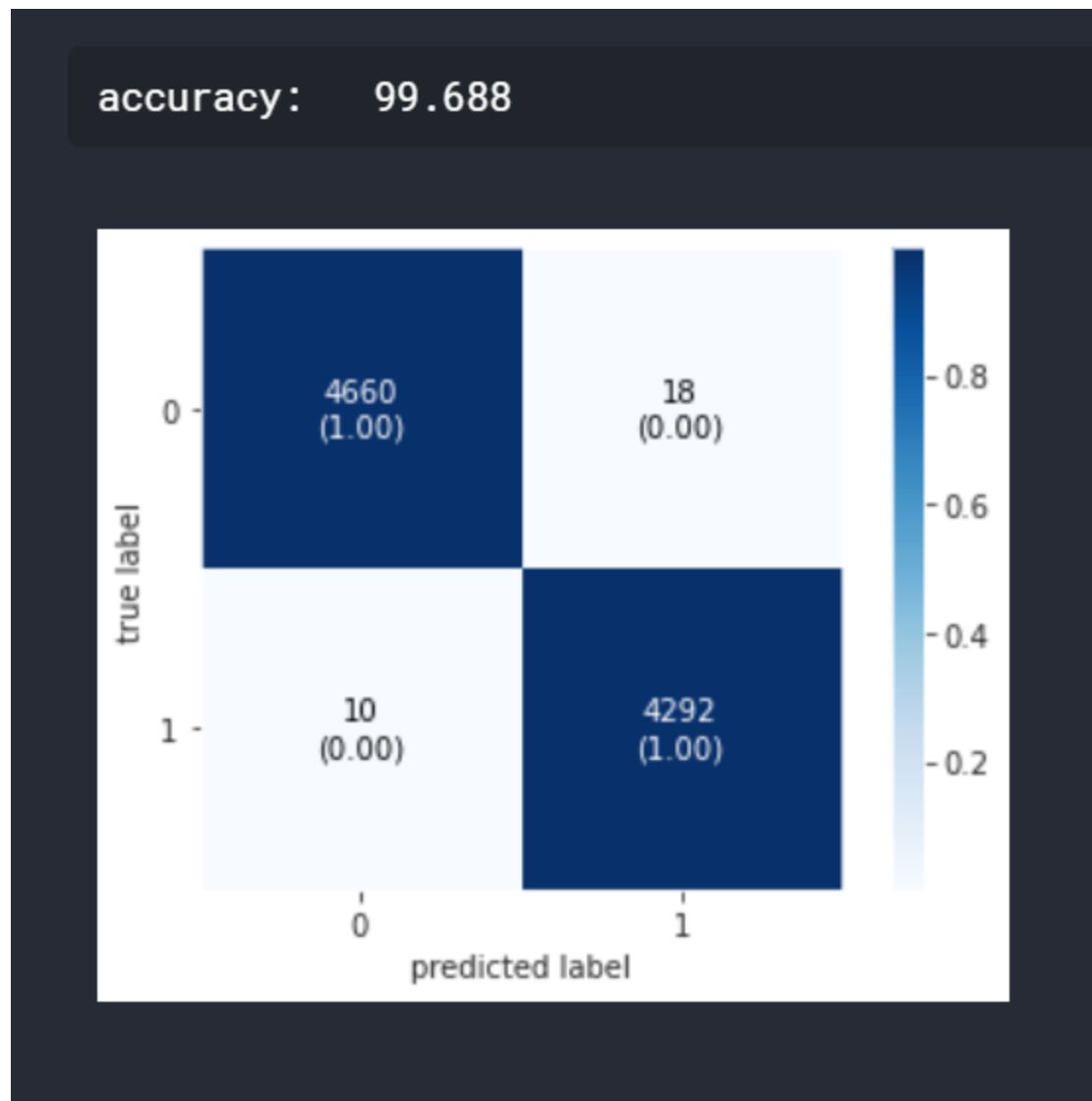
```
fig, ax = plot_confusion_matrix(conf_mat=confusion_matrix(y_test,  
prediction),
```

```
                                show_absolute=True,
```

```
                                show_normed=True,
```

```
                                colorbar=True)
```

```
plt.show()
```



Confusion Matrix

It is clear that multinomial naive bayes is not performing well as compared to other models. SVM and passive aggressive classifier have almost similar performance.

Conclusion

We have classified our news data using three classification models. We have analysed the performance of the models using accuracy and confusion matrix. But this is only a beginning point for the problem. There are advanced techniques like BERT, GloVe and ELMo which are popularly used in the field of NLP. If you are interested in NLP, you can work forward with these techniques.