Northeastern University

INFO 6105 Final Project

InShorts: Classifying & Summarizing Daily News

Team 4 – Monday Batch

Group Members

Anuja Naik

Ritesh Pendurkar

Rohit Gulati

Shubham Mahajan

Eshanee Thakur

Contents

Introduction

Background

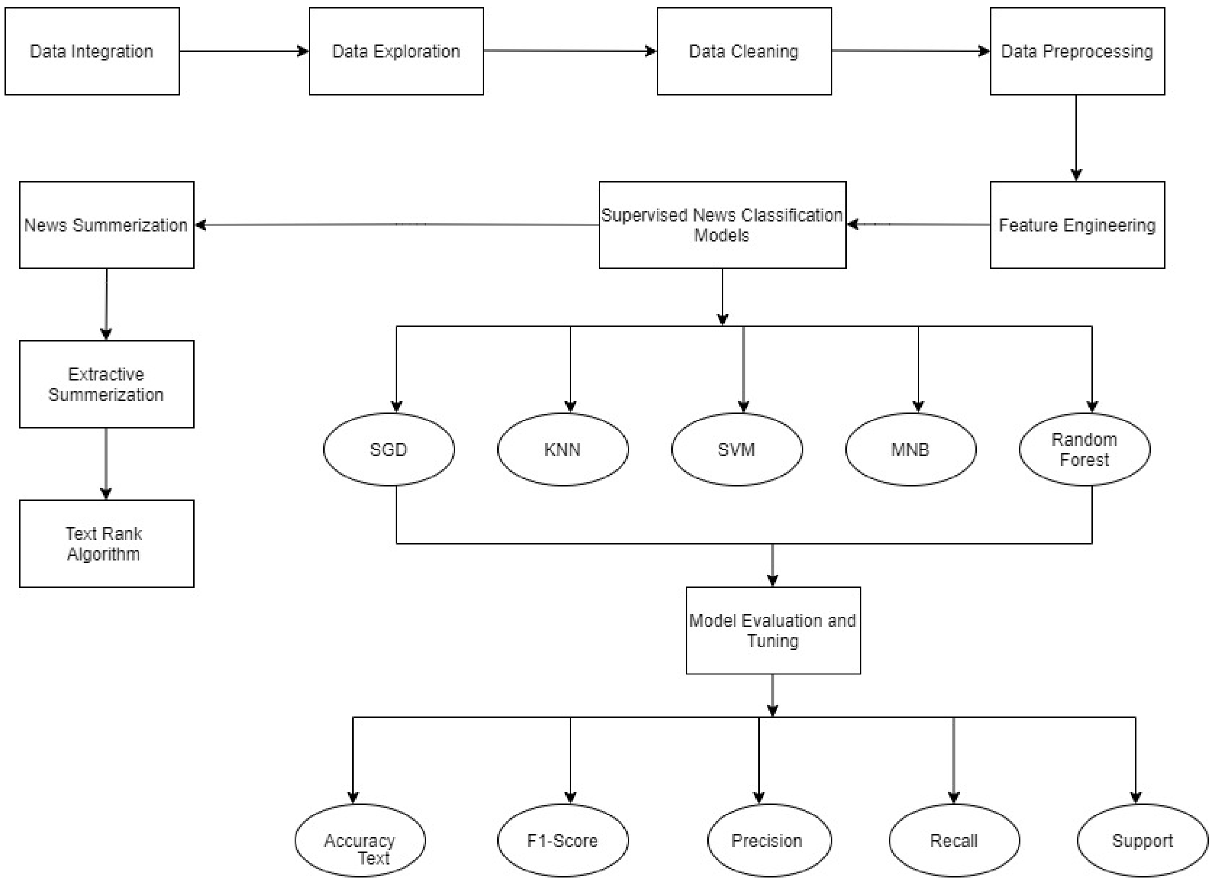
There exists a large amount of information being stored in the electronic format. With such data it has become a necessity of such means that could interpret and analyse such data and extract such facts that could help in decision-making. News information was not easily and quickly available until the beginning of last decade. But now news is easily accessible via content providers such as online news services. With the increase in the number of news it has got difficult for users to access news of his interest which makes it a necessity to categories news so that it could be easily accessed. Categorization refers to grouping that allows easier navigation among articles. Internet news needs to be divided into categories. This will help users to access the news of their interest in real-time without wasting any time. Additionally, with so little time at our disposal, it is not feasible to go through the entire news description. A short description of the entire news article would help users not just in saving time but also cover many more news articles in a short time frame. Summarization refers to providing a precis or a brief summary of the large text, without losing the integrity and the theme of the article.

Objective

When it comes to news it is much difficult to classify them as news articles are continuously appearing and thereby, they need to be processed every time. That news could be never-seen-before and could fall in a new category. We have implemented InShorts, a news classifier and a summarizing tool using Machine Learning approach. InShorts is an automated news classification model used to identify and categorize topics of untracked news. Our system implements five machine learning models. Here, machine learns to categorize on the news articles fed through the dataset. Keywords extracted from the attributes such as headlines are later fed as inputs to the classifier which predicts the appropriate category for every new news article. Using the Natural Language processing technique and TextRank algorithm the system extracts most important sentences from the news description and forms a short summary of the article. Hence, our framework saves users' time by providing the users with categorized news articles along with a summarized version of each article.

Methodology and Approach

There are different steps involved in news classification. Classification is a difficult activity as it requires pre-processing steps to convert the textual data into structured form from the un-structured form. Text classification process involves following main steps for classification of news article. These steps are data collection, pre-processing, feature selection, classification techniques application, and evaluating performance measures.

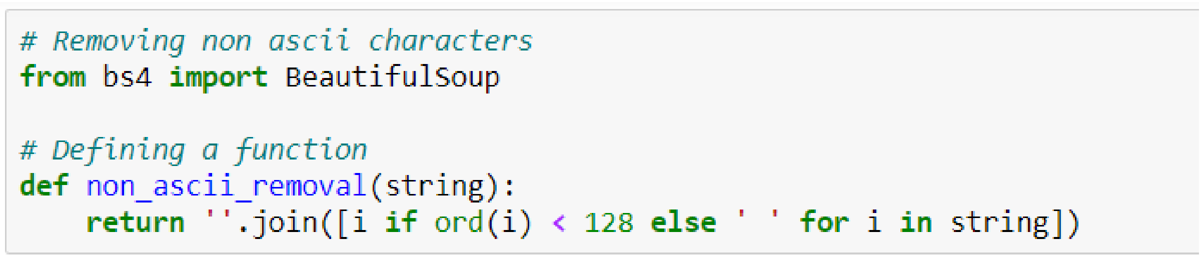


News Article Data Cleaning

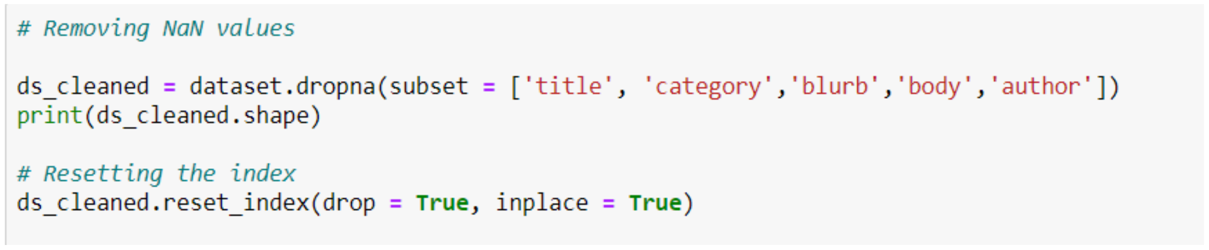
Before creating any feature from the raw text, we must perform a cleaning process to ensure no distortions are introduced to the model. We have followed these steps:

- Special character and non ascii character cleaning: special characters such as "\n" double quotes must be removed from the text along with non ascii characters since we aren't expecting any predicting power from them.
- Upcase/downcase: we would expect, for example, "Book" and "book" to be the same word and have the same predicting power. For that reason, we have downcased every word.

- Punctuation signs: characters such as “?”, “!”, “;” have been removed.
- Stop words: words such as “what” or “the” won’t have any predicting power since they will presumably be common to all the documents. For this reason, they may represent noise that can be eliminated. We have downloaded a list of English stop words from the nltk package and then deleted them from the corpus.



```
# Removing non ascii characters
from bs4 import BeautifulSoup

# Defining a function
def non_ascii_removal(string):
    return ''.join([i if ord(i) < 128 else ' ' for i in string])
```



```
# Removing NaN values

ds_cleaned = dataset.dropna(subset = ['title', 'category','blurb','body','author'])
print(ds_cleaned.shape)

# Resetting the index
ds_cleaned.reset_index(drop = True, inplace = True)
```

News Pre-processing

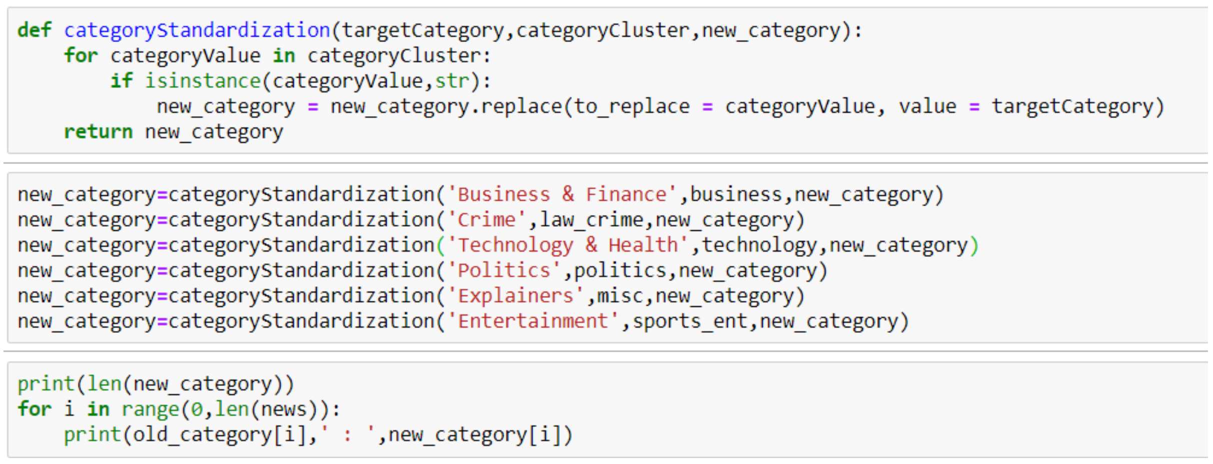
Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. We have converted the raw data into a clean data set. In other words, as the dataset contains the data from different sources, the data is in raw format which is not feasible for the analysis. For achieving better results from the applied model in Machine Learning the format of the data must be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore, to execute random forest algorithm null values must be managed from the original raw data set. Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen. Hence, we have performed data pre-processing on the raw dataset to convert it into a clean and useable dataset.

Feature Engineering

When there exist a large number of features and each of the features is a well-known descriptive word for each class, a lot of time may be required in classification and it may be possible that expected accuracy may not be achieved and to overcome these issues, a process named as feature selection is adopted in which only those relevant and highly effective features are chosen, which may prove more noticeable for better news classification. Feature engineering is the process of transforming data into features to act as inputs for machine learning models such that good quality features help in improving the model performance. When dealing with text data, there are several ways of obtaining features that represent the data.

For feature engineering we created clusters for categories. Initially there were around 170 categories which could not be considered as core categories. These 170 categories could be grouped into some generic categories. So, we manually created 5 clusters and added all the related categories in those 5 categories. These are mainly -

- Entertainment
- Health and Technology
- Politics
- Crime
- Business and Finance



```
def categoryStandardization(targetCategory,categoryCluster,new_category):
    for categoryValue in categoryCluster:
        if isinstance(categoryValue,str):
            new_category = new_category.replace(to_replace = categoryValue, value = targetCategory)
    return new_category
```

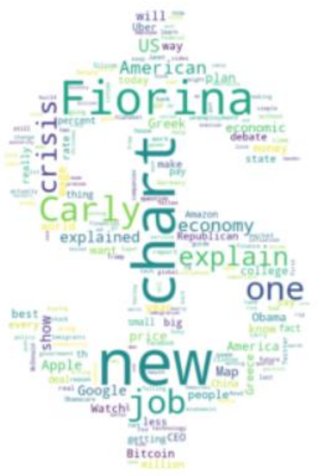
```
new_category=categoryStandardization('Business & Finance',business,new_category)
new_category=categoryStandardization('Crime',law_crime,new_category)
new_category=categoryStandardization('Technology & Health',technology,new_category)
new_category=categoryStandardization('Politics',politics,new_category)
new_category=categoryStandardization('Explainers',misc,new_category)
new_category=categoryStandardization('Entertainment',sports_ent,new_category)
```

```
print(len(new_category))
for i in range(0,len(news)):
    print(old_category[i],' : ',new_category[i])
```

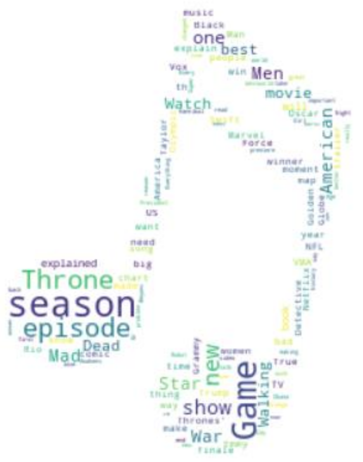
Data Visualization

Below are the word clouds generated for every category.

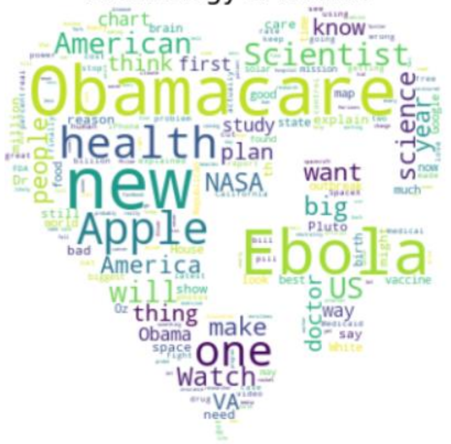
Business & Finance



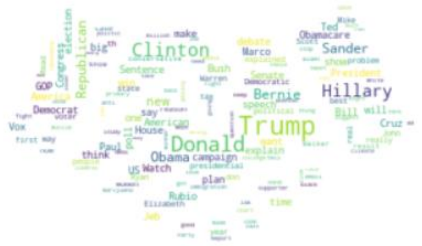
Entertainment



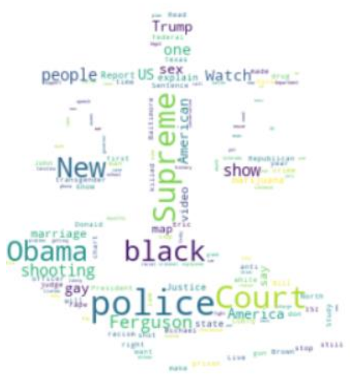
Technology & Health



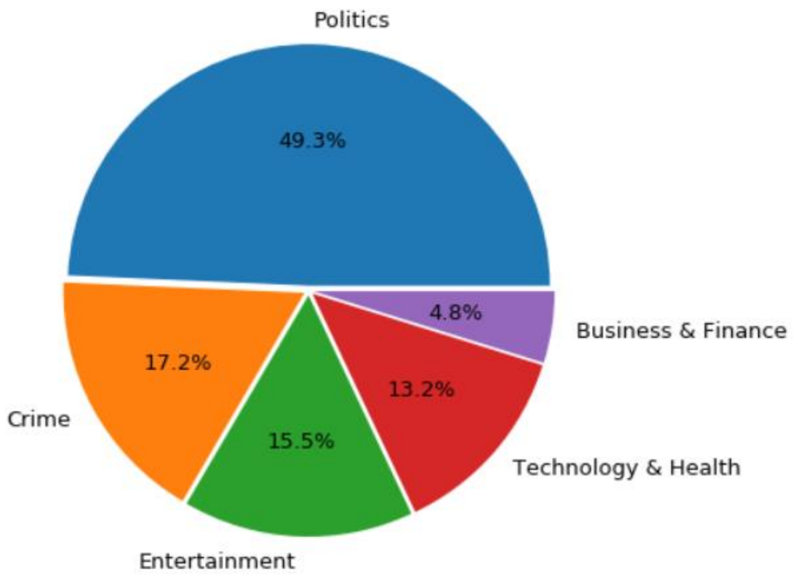
Politics

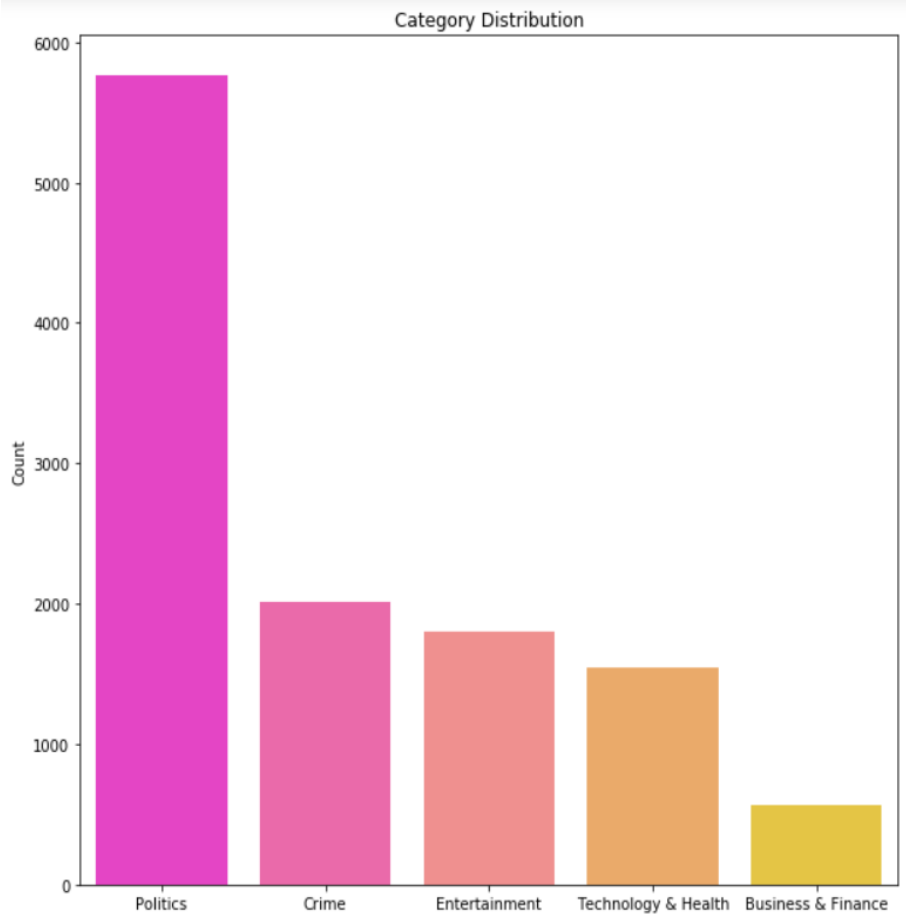


Crime



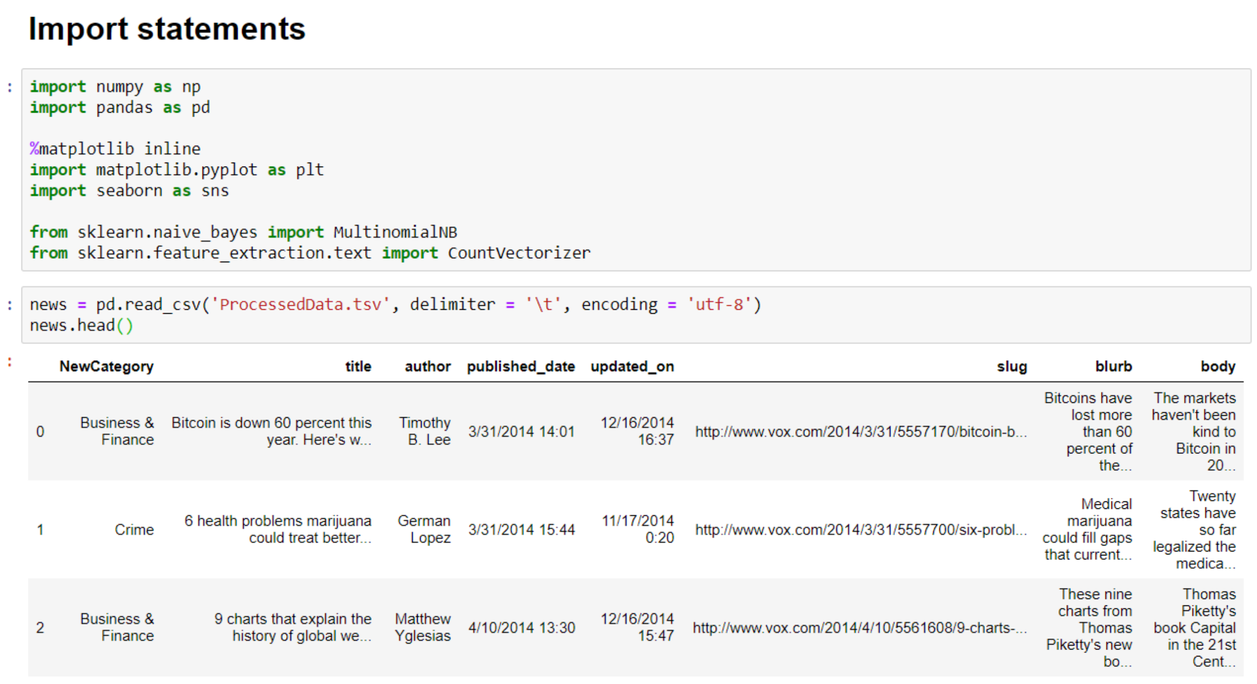
The dataset is categorized into five categories as depicted in the pie chart.





Data Loading

The very first step is loading the tsv file into a dataframe. We have imported all the necessary libraries as shown in the below screenshot. You can see the first three rows of the dataset are printed, as well as the target variable for the whole dataset.



Import statements

```
import numpy as np
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
```

```
news = pd.read_csv('ProcessedData.tsv', delimiter = '\t', encoding = 'utf-8')
news.head()
```

	NewCategory	title	author	published_date	updated_on	slug	blurb	body
0	Business & Finance	Bitcoin is down 60 percent this year. Here's w...	Timothy B. Lee	3/31/2014 14:01	12/16/2014 16:37	http://www.vox.com/2014/3/31/5557170/bitcoin-b...	Bitcoins have lost more than 60 percent of the...	The markets haven't been kind to Bitcoin in 20...
1	Crime	6 health problems marijuana could treat better...	German Lopez	3/31/2014 15:44	11/17/2014 0:20	http://www.vox.com/2014/3/31/5557700/six-probl...	Medical marijuana could fill gaps that current...	Twenty states have so far legalized the medica...
2	Business & Finance	9 charts that explain the history of global we...	Matthew Yglesias	4/10/2014 13:30	12/16/2014 15:47	http://www.vox.com/2014/4/10/5561608/9-charts-...	These nine charts from Thomas Piketty's new bo...	Thomas Piketty's book Capital in the 21st Cent...

Machine Learning Algorithms

After feature selection the next phase is the classification phase which is an important phase in which the aim is to classify the unseen news to their respective categories. The most common news classification methods are Multinomial Naive Bayes, Random Forest, Support Vector Machines, K-Nearest Neighbours and Stochastic Gradient Descent. In order to achieve the most optimal results we have used hyperparameter technique in all the above models.

We have used the following libraries in our system for the analysis:

- Numpy
- Pandas
- Sklearn
- Matplotlib
- Seaborn

NumPy

NumPy is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. In our project we have implemented NumPy to convert confusion matrix to array in order to plot confusion matrix for each model.

Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It is built on the NumPy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables. In our project we have implemented pandas to import a tsv file as a dataframe, concatenate multiple dataframe into a single dataframe and for other technicalities while generating the model.

SciKit-Learn

Scikit-learn library is used to implement various algorithms like support vector machine, random forests, Gaussian Naive Bayes, Scalers, and kNN. It also supports Python numerical and scientific libraries like NumPy and SciPy. In our project, we have used this library for model training and for calculating the model metrics like accuracy, score, etc

MatPlotLib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits. In our project, we use MatPlotLib to display a Bar Chart to visualize the news category distribution.

Seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures. Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. We have implemented seaborn for mapping bar charts to map categories against the count of news articles derived from the dataset.

Let’s now go through each model and understand the workings for the same.

Multinomial Naïve Bayes (MNB) Classifier

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. Naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes Theorem with strong independence assumptions. This algorithm computes the posterior probability of the news belonging to different categories and it assigns news article to the category with the highest posterior probability. Bayesian theory works as a framework for making decision under uncertainty - a probabilistic approach to inference and is particularly suited when the dimensionality of the inputs data is high.

Bayes theorized that the probability of future events could be calculated by determining their earlier frequency. Bayes theorem states that:

P(Y=y_i |X=x_k)=(P(Y=y_i)|P(X=x_k |Y=y_i))/P(X=x_k)

Where:

P(Y=y_i) - Prior probability of hypothesis Y - Prior

P(X=x_k) - Prior probability of training data X-Evidence

P(X=x_k |Y=y_i)- Probability of X given Y- Likelihood

P(Y=y_i |X=x_k)- Probability of Y given X- Posterior probability.

Feature Extraction and Vectorization

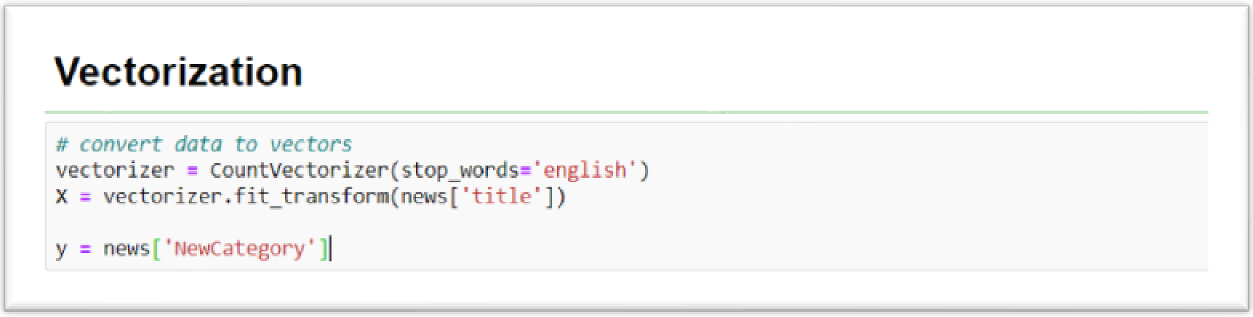
Vectorization is the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the Bag of Words or "Bag of n-grams" representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document. We convert a collection of text documents to a matrix of token counts.

Features and samples are defined as follows:

- Each individual token occurrence frequency (normalized or not) is treated as a feature.
- The vector of all the token frequencies for a given document is considered a multivariate sample

3 Steps for Vectorization are:

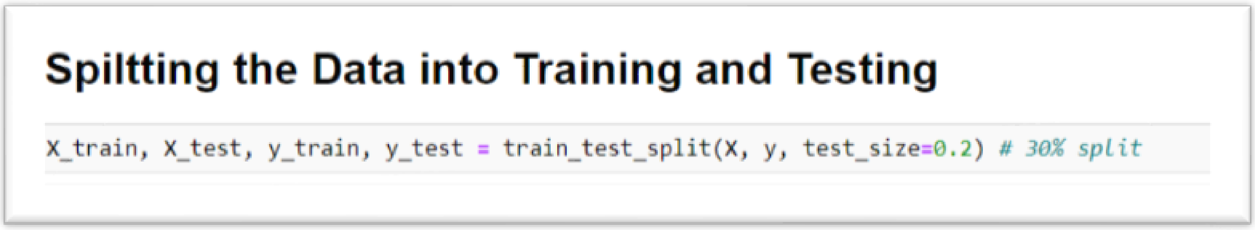
- Import
- Instantiate
- Fit and Transform


Vectorization

```
# convert data to vectors
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(news['title'])

y = news['NewCategory']
```

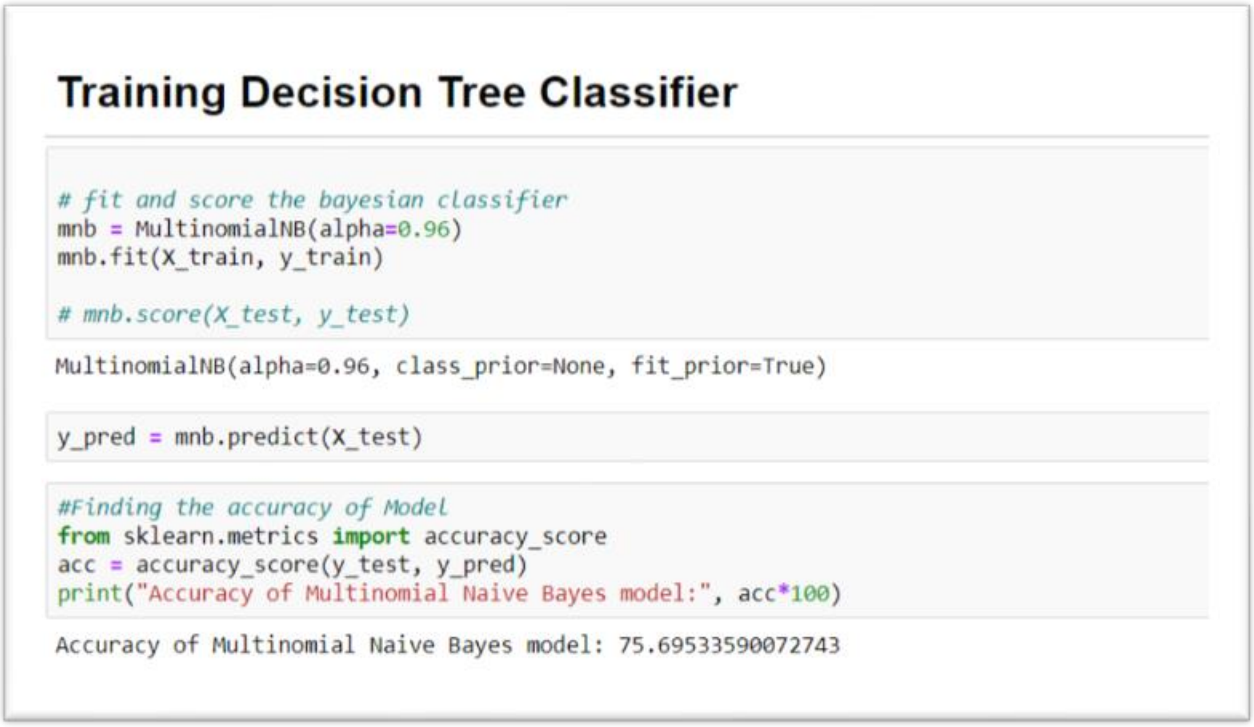
Splitting the data and Training the model

First, we have separated the columns into dependent and independent variables (or features and label). Then we have split those variables into train and test set.


Spiltting the Data into Training and Testing

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # 30% split
```

After splitting, we have generated a Multinomial Naïve Bayes model on the training set and performed prediction on test set features.


Training Decision Tree Classifier

```
# fit and score the bayesian classifier
mnb = MultinomialNB(alpha=0.96)
mnb.fit(X_train, y_train)

# mnb.score(X_test, y_test)
```

```
MultinomialNB(alpha=0.96, class_prior=None, fit_prior=True)
```

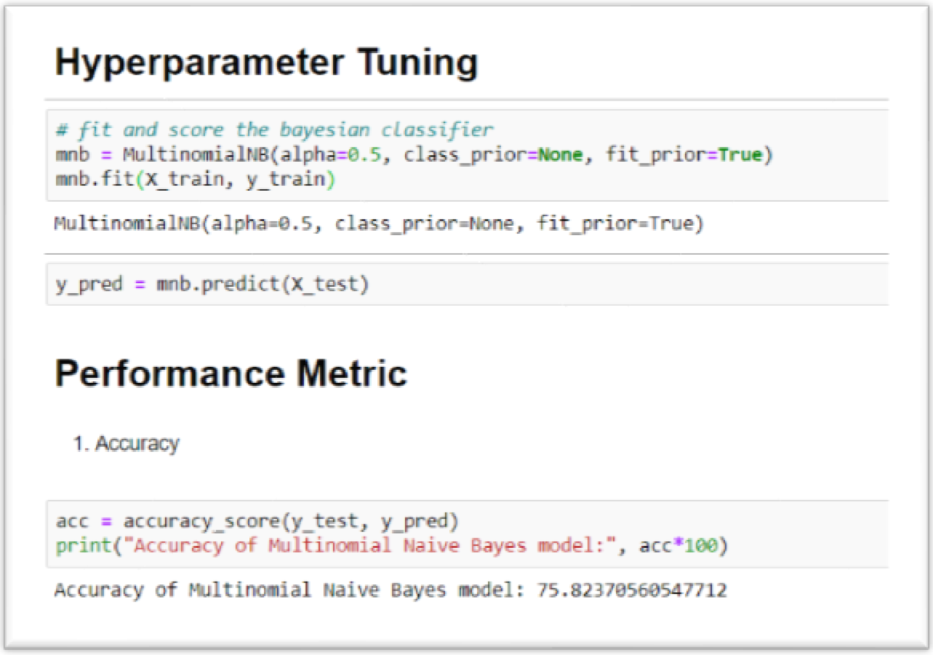
```
y_pred = mnb.predict(X_test)
```

```
#Finding the accuracy of Model
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("Accuracy of Multinomial Naive Bayes model:", acc*100)
```

```
Accuracy of Multinomial Naive Bayes model: 75.69533590072743
```

In order to enhance the performance, we would perform hyperparameter tuning. There are a few parameters we implicitly assumed when we did our training. In Multinomial Naive Bayes, the alpha parameter is what is known as a hyperparameter; i.e. a parameter that controls the form of the model itself. In most cases, the best way to determine optimal values for hyperparameters is through a grid search over possible parameter values, using cross validation to evaluate the performance of the model on your data at each value.

In our model, we changed the value of alpha parameter from 0.96 to 0.5. This modification led to increase in the accuracy of the model to 75.83%



Hyperparameter Tuning

```
# fit and score the bayesian classifier
mnb = MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True)
mnb.fit(X_train, y_train)
```

```
MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True)
```

```
y_pred = mnb.predict(X_test)
```

Performance Metric

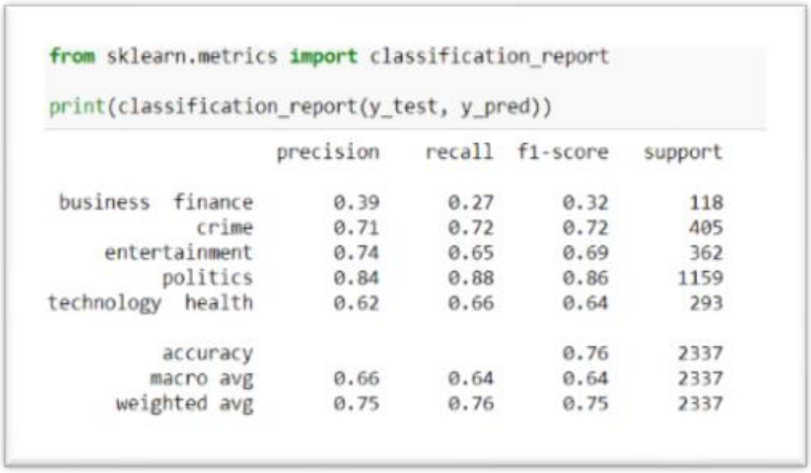
1. Accuracy

```
acc = accuracy_score(y_test, y_pred)
print("Accuracy of Multinomial Naive Bayes model:", acc*100)
```

```
Accuracy of Multinomial Naive Bayes model: 75.82370560547712
```

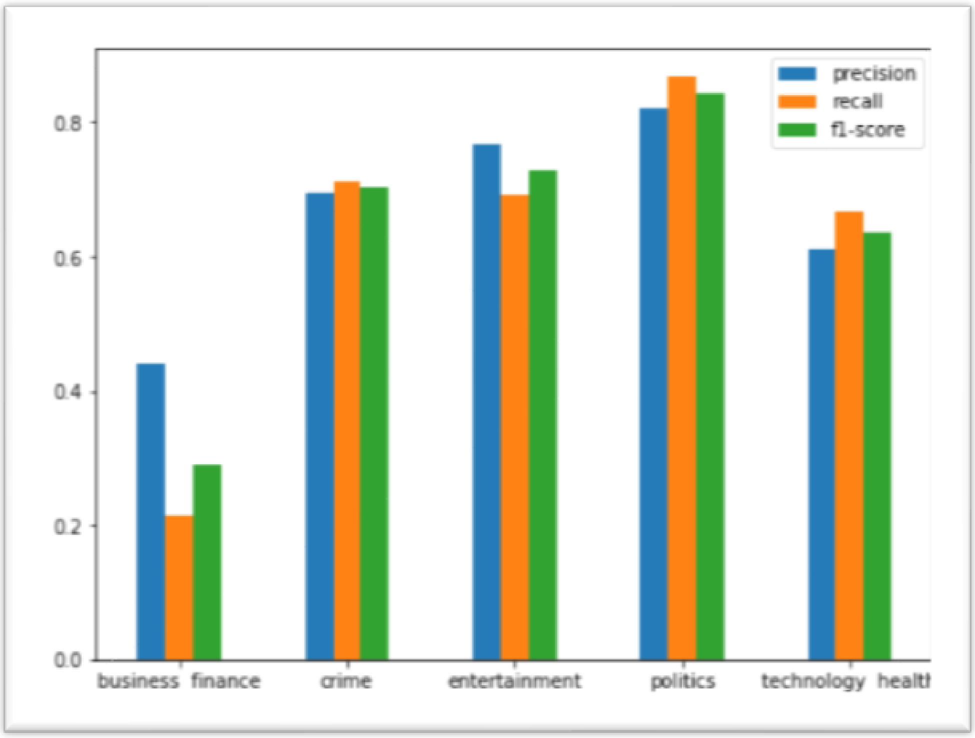
Evaluating the model

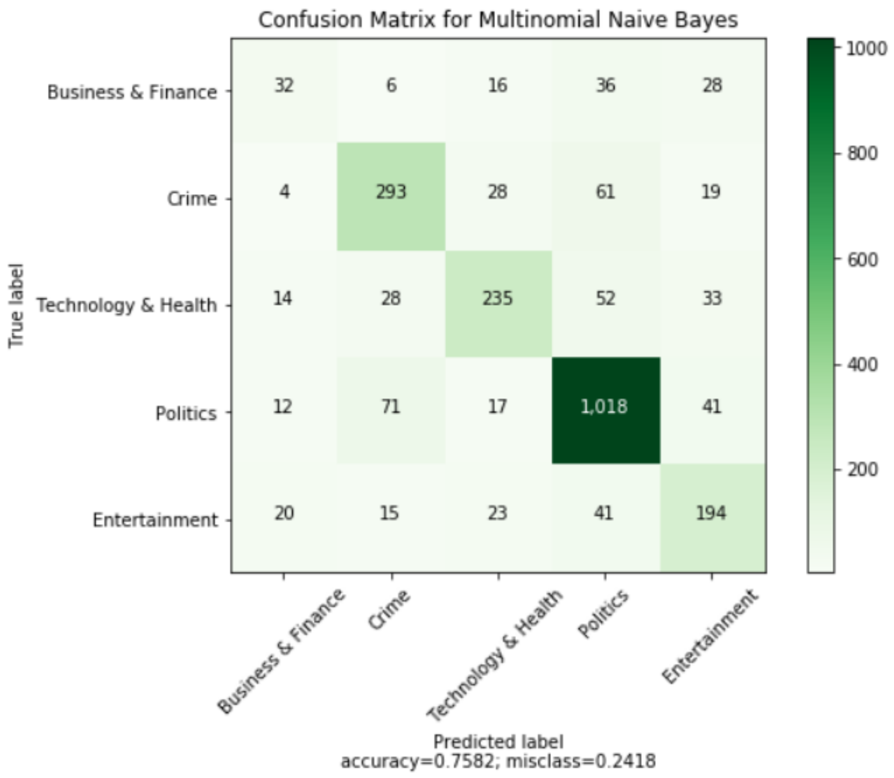
We have evaluated the model using the classification report and confusion matrix. Classification report comprises of categories mapped against the model metrics such as F1 score, Recall, support and Precision.



```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
                    precision    recall  f1-score   support

 business  finance       0.39      0.27      0.32       118
             crime       0.71      0.72      0.72       405
     entertainment       0.74      0.65      0.69       362
          politics       0.84      0.88      0.86      1159
technology  health       0.62      0.66      0.64       293

          accuracy                           0.76      2337
         macro avg       0.66      0.64      0.64      2337
      weighted avg       0.75      0.76      0.75      2337
```





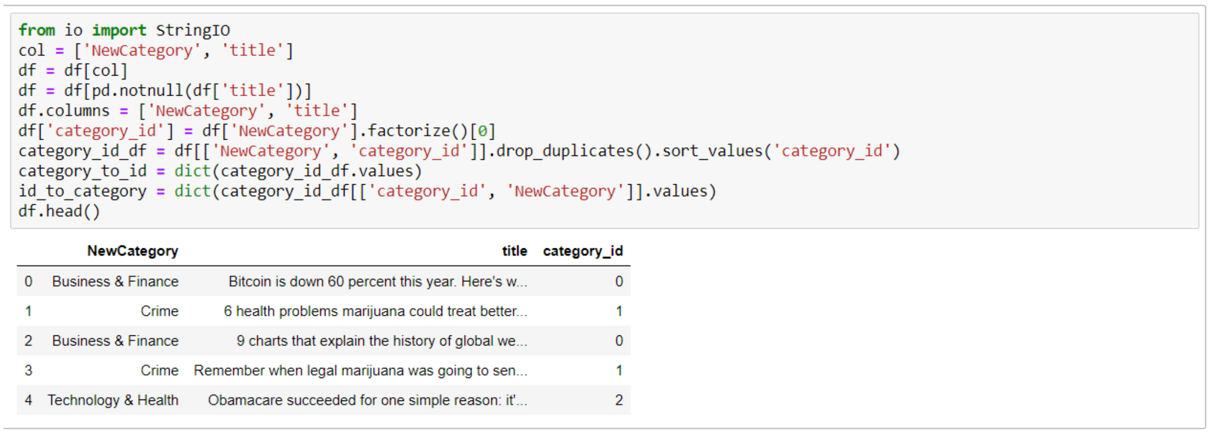
	Business & Finance	Crime	Technology & Health	Politics	Entertainment
Business & Finance	32	6	16	36	28
Crime	4	293	28	61	19
Technology & Health	14	28	235	52	33
Politics	12	71	17	1,018	41
Entertainment	20	15	23	41	194

Random Forest Classifier

Random Forest is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. Technically speaking, a random forest is a meta estimator that fits the number of decision tree classifiers on various sub samples on the data set and use averaging to improve the predictor accuracy and control over fitting. As an ensemble method we thought random forest would perform better than decision trees. While using random forest we will also take advantage of decision tree test experience.

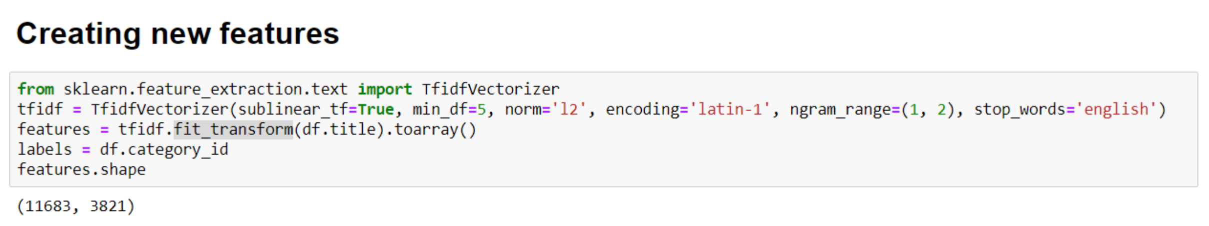
Feature Extraction and Vectorization

Unlike other models, Random Forest takes in a numeric value in order to perform feature engineering. In our project, the original dataset has categories as text value. In order to achieve feature engineering, we have factorized the Category column and converted them into category_ids. Later, the usual vectorization process has been implemented to follow bag of words approach.



```
from io import StringIO
col = ['NewCategory', 'title']
df = df[col]
df = df[pd.notnull(df['title'])]
df.columns = ['NewCategory', 'title']
df['category_id'] = df['NewCategory'].factorize()[0]
category_id_df = df[['NewCategory', 'category_id']].drop_duplicates().sort_values('category_id')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category_id', 'NewCategory']].values)
df.head()
```

	NewCategory	title	category_id
0	Business & Finance	Bitcoin is down 60 percent this year. Here's w...	0
1	Crime	6 health problems marijuana could treat better...	1
2	Business & Finance	9 charts that explain the history of global we...	0
3	Crime	Remember when legal marijuana was going to sen...	1
4	Technology & Health	Obamacare succeeded for one simple reason: it'...	2



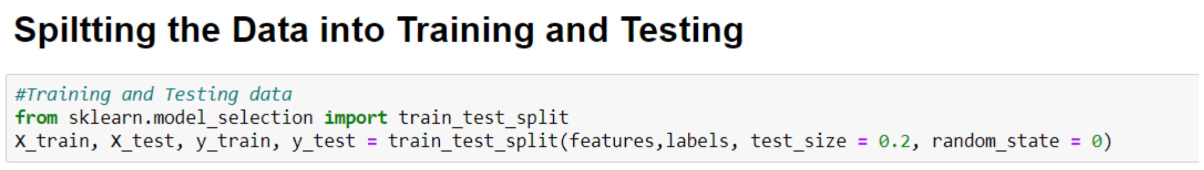
Creating new features

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='latin-1', ngram_range=(1, 2), stop_words='english')
features = tfidf.fit_transform(df.title).toarray()
labels = df.category_id
features.shape
```

```
(11683, 3821)
```

Splitting the data and Training the model

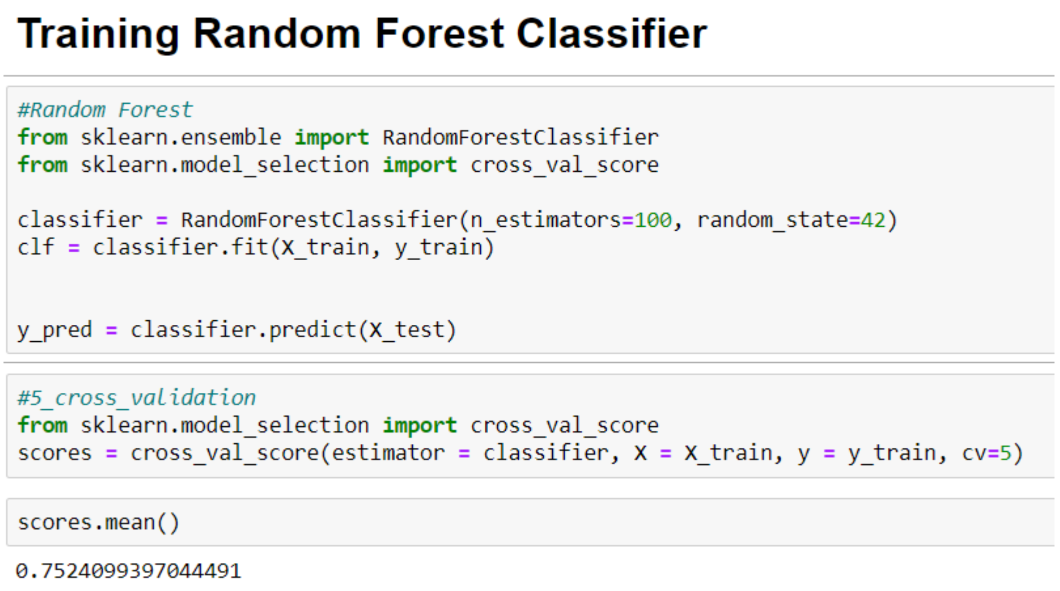
First, we have separated the columns into dependent and independent variables (or features and label). Then we have split those variables into train and test set.



Spilting the Data into Training and Testing

```
#Training and Testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features,labels, test_size = 0.2, random_state = 0)
```

After splitting, we have generated Random Forest model on the training set and performed prediction on test set features.



Training Random Forest Classifier

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

classifier = RandomForestClassifier(n_estimators=100, random_state=42)
clf = classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)
```

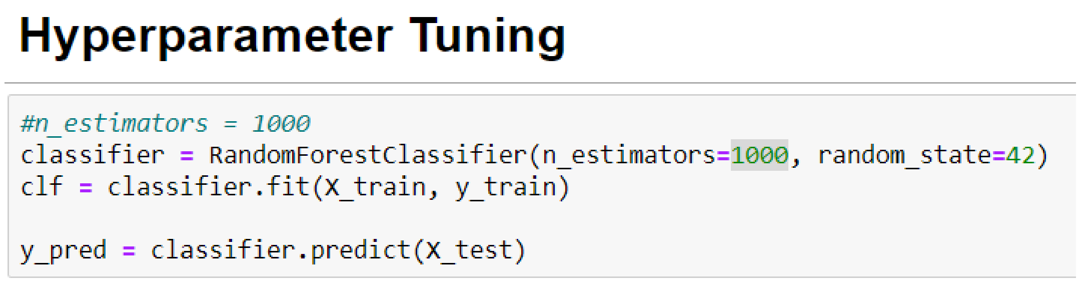
```
#5_cross_validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv=5)
```

```
scores.mean()
```

```
0.7524099397044491
```

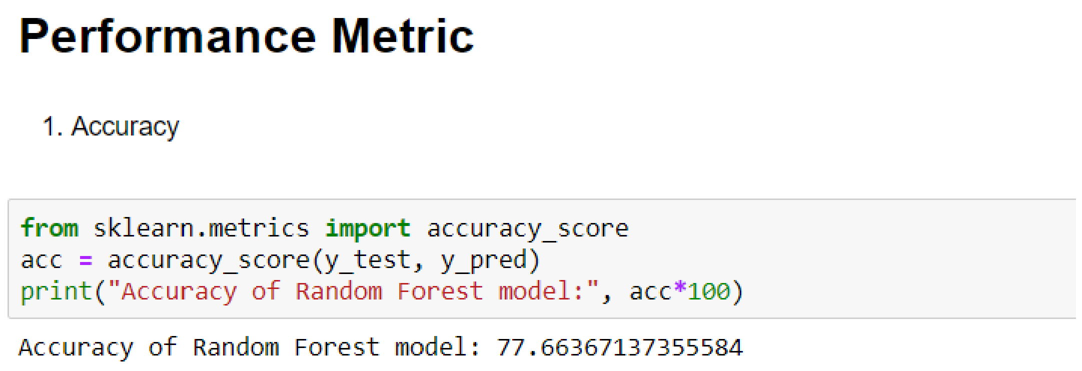
In order to enhance the performance, we would perform hyperparameter tuning. There are a few parameters we implicitly assumed when we did our training. In Random Forest model, n_estimators are the parameter on which hyperparameter tuning is performed. N_estimator denotes the number of trees in the forest. In most cases, the best way to determine optimal values for hyperparameters is through a grid search over possible parameter values, using cross validation to evaluate the performance of the model on your data at each value.

In our model, we changed the value of n_estimator parameter from 100 to 1000. This modification led to increase in the accuracy of the model to 77.66%



Hyperparameter Tuning

```
#n_estimators = 1000
classifier = RandomForestClassifier(n_estimators=1000, random_state=42)
clf = classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```



Performance Metric

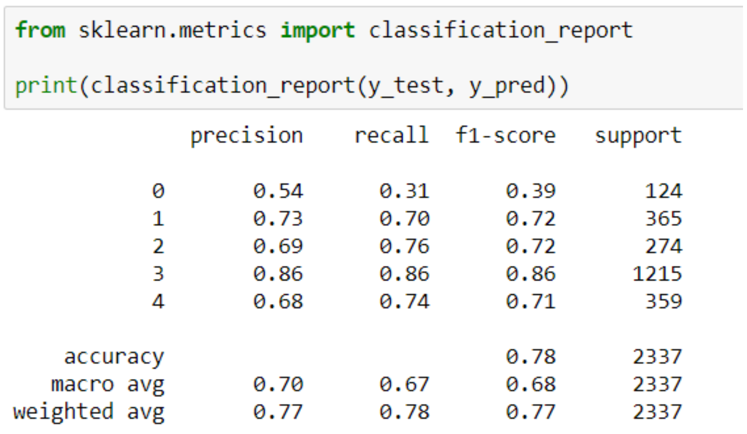
1. Accuracy

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("Accuracy of Random Forest model:", acc*100)
```

```
Accuracy of Random Forest model: 77.66367137355584
```

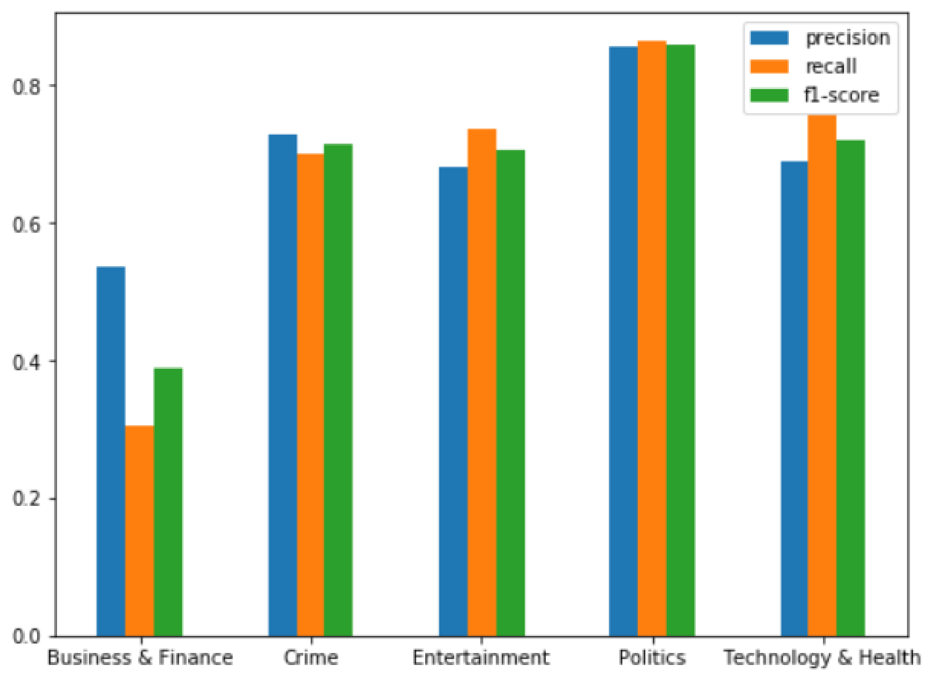
Evaluating the model

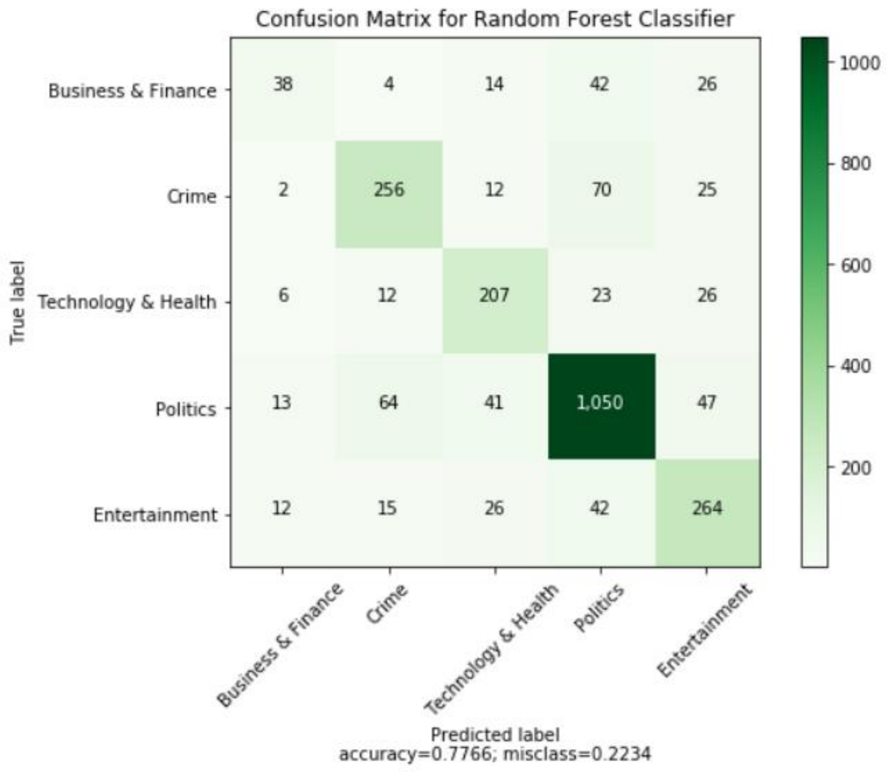
We have evaluated the model using the classification report and confusion matrix. Classification report comprises of categories mapped against the model metrics such as F1 score, Recall, support and Precision.



```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.54      0.31      0.39       124
           1       0.73      0.70      0.72       365
           2       0.69      0.76      0.72       274
           3       0.86      0.86      0.86      1215
           4       0.68      0.74      0.71       359

    accuracy                           0.78      2337
   macro avg       0.70      0.67      0.68      2337
weighted avg       0.77      0.78      0.77      2337
```


precision
recall
f1-score
0.8
0.6
0.4
0.2
0.0
Business & Finance
Crime
Entertainment
Politics
Technology & Health


Confusion Matrix for Random Forest Classifier
True label
Business & Finance
Crime
Technology & Health
Politics
Entertainment
38 4 14 42 26
2 256 12 70 25
6 12 207 23 26
13 64 41 1,050 47
12 15 26 42 264
1000
800
600
400
200
Predicted label
accuracy=0.7766; misclass=0.2234

Support Vector Machine (SVM) Classifier

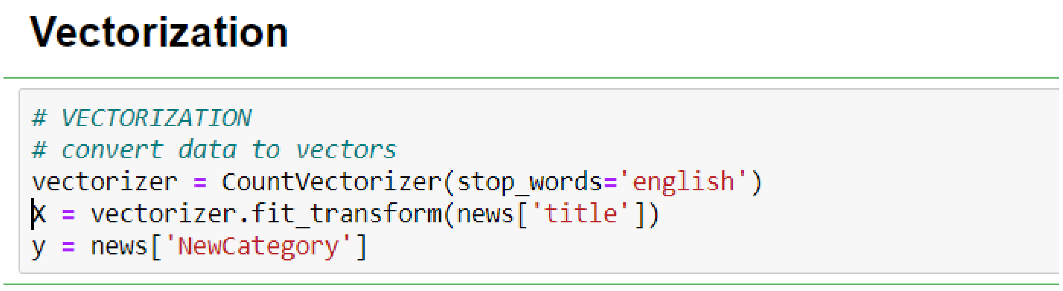
Support Vector Machines is a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane (MMH) that best divides the dataset into classes. Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier. A hyperplane is a decision plane which separates between a set of objects having different class memberships. A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

Kernel trick helps you to build a more accurate classifier.

- Linear Kernel - A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.
- Polynomial Kernel - A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.
- Radial Basis Function Kernel - The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

Feature Extraction and Vectorization

The usual vectorization process has been implemented to follow bag of words approach.



Vectorization

```
# VECTORIZATION
# convert data to vectors
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(news['title'])
y = news['NewCategory']
```

Splitting the data and Training the model

First, we have separated the columns into dependent and independent variables (or features and label). Then we have split those variables into train and test set.

```
# Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # 20% split
```

After splitting, we have generated Support Vector Machine model on the training set and performed prediction on test set features.

```
# Importing SVM = training without hyperparametering
clf_svm = svm.LinearSVC()
clf_svm.fit(X_train, y_train)
predicted = clf_svm.predict(X_test)
```

```
# from sklearn.metrics import accuracy_score
print("Accuracy of SVM model:", accuracy_score(y_test, predicted)*100)
```

```
Accuracy of SVM model: 74.79674796747967
```

In order to enhance the performance, we would perform hyperparameter tuning. There are a few parameters we implicitly assumed when we did our training. In SVM model, we have used grid search over possible parameter values to determine optimal values for hyperparameters.

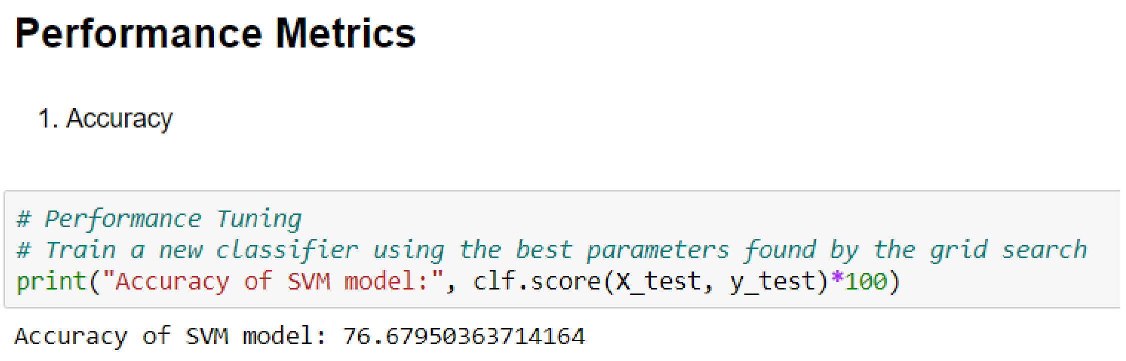
In our model, we used varied value for C, gamma and kernel. Using grid search, we determined that the best fit for our model would be: C = 1000, gamma = 0.0001 and kernel = rbf. This modification led to increase in the accuracy of the model to 76.68%

Hyperparameter Tuning

```
# HyperParameter tuning = setting parameters for enhancing the training model
parameter_candidates = [
  {'C': [1, 10, 100, 1000, 10000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000,10000], 'gamma': [0.1,0.01,0.001, 0.0001,0.00001], 'kernel': ['rbf']}
]
```

```
# Create a classifier object with the classifier and parameter candidates
# clf = GridSearchCV(estimator=svm.SVC(), param_grid = parameter_candidates, n_jobs = -1)
clf = GridSearchCV(estimator=svm.SVC(), param_grid = parameter_candidates, refit = True)
```

```
# Train the classifier on data1's feature and target data
clf.fit(X_train, y_train)
```



Performance Metrics

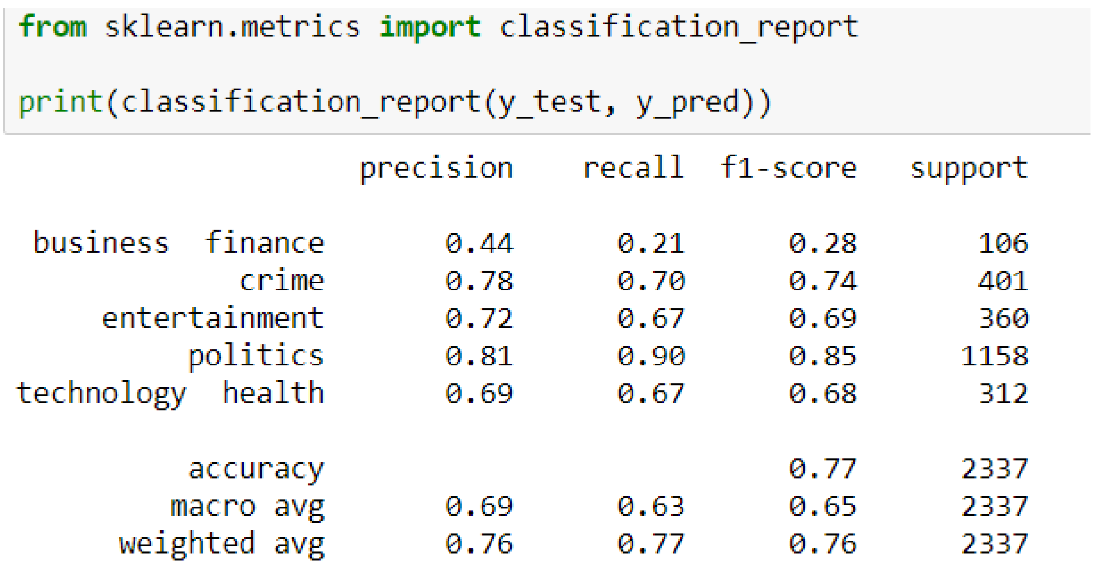
1. Accuracy

```
# Performance Tuning
# Train a new classifier using the best parameters found by the grid search
print("Accuracy of SVM model:", clf.score(X_test, y_test)*100)
```

```
Accuracy of SVM model: 76.67950363714164
```

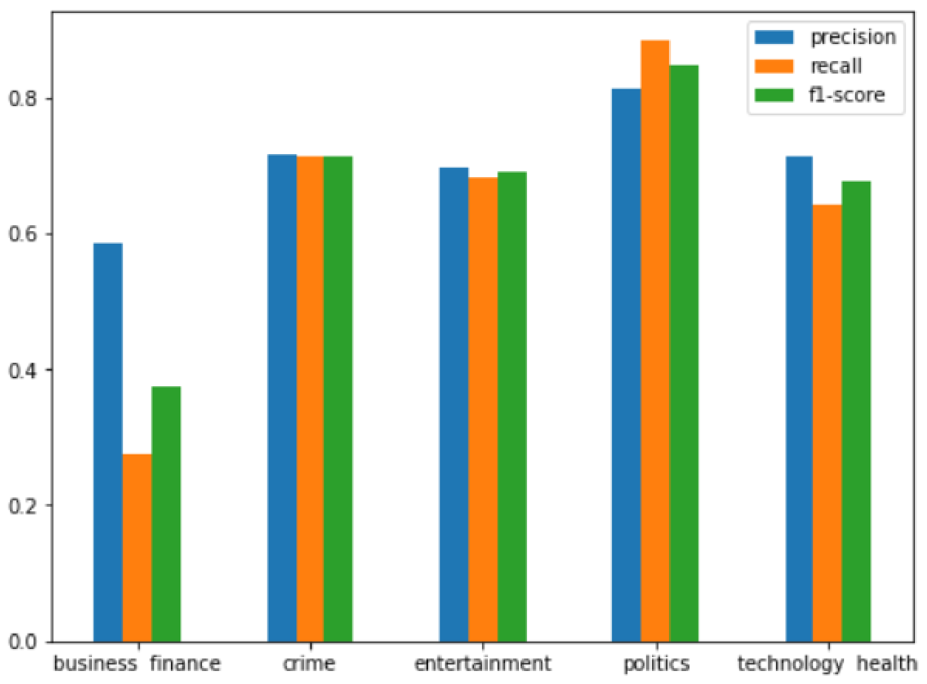
Evaluating the model

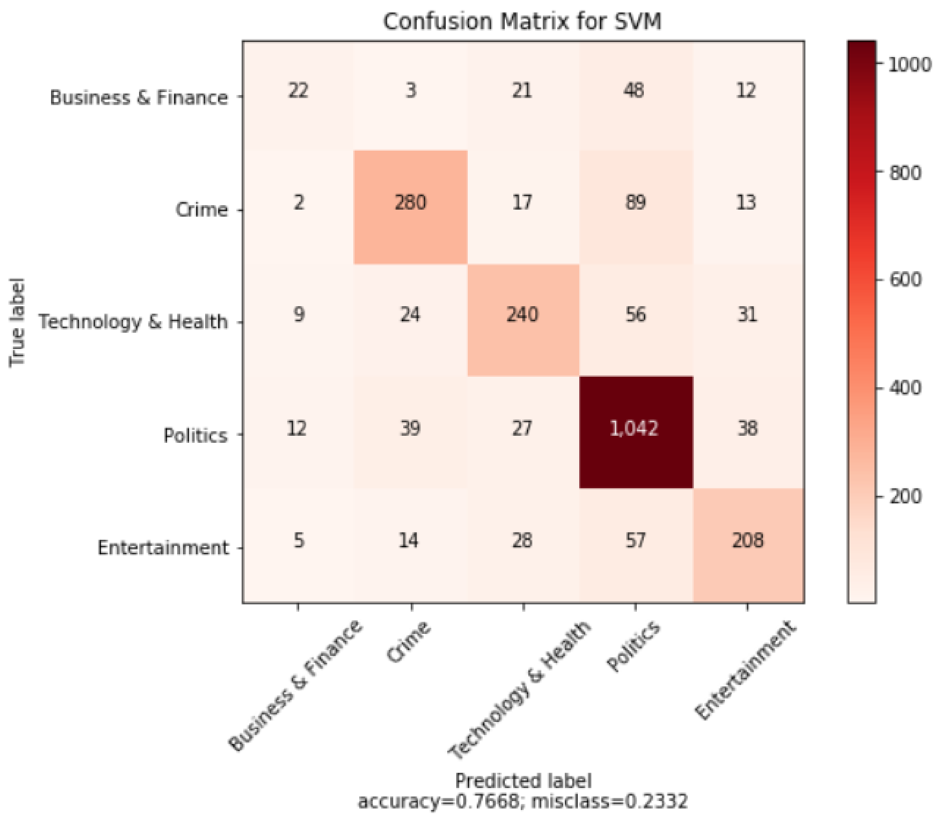
We have evaluated the model using the classification report and confusion matrix. Classification report comprises of categories mapped against the model metrics such as F1 score, Recall, support and Precision.



```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
                    precision    recall  f1-score   support

  business  finance      0.44      0.21      0.28       106
              crime      0.78      0.70      0.74       401
      entertainment      0.72      0.67      0.69       360
           politics      0.81      0.90      0.85      1158
 technology  health      0.69      0.67      0.68       312

           accuracy                          0.77      2337
          macro avg      0.69      0.63      0.65      2337
       weighted avg      0.76      0.77      0.76      2337
```


precision
recall
f1-score
0.0
0.2
0.4
0.6
0.8
business finance
crime
entertainment
politics
technology health

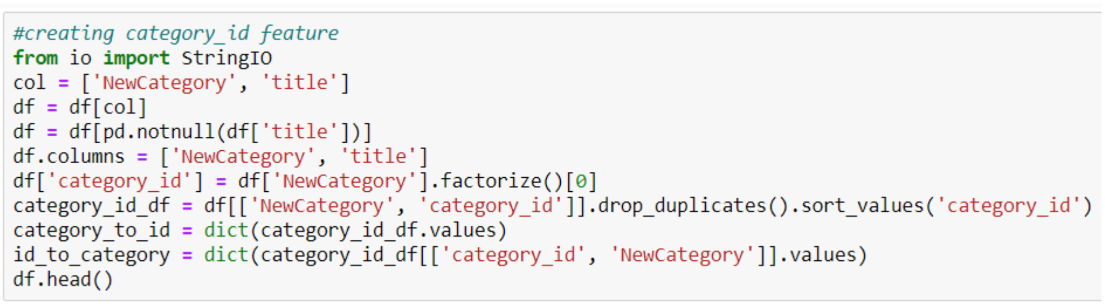

Confusion Matrix for SVM
True label
Business & Finance
Crime
Technology & Health
Politics
Entertainment
22 3 21 48 12
2 280 17 89 13
9 24 240 56 31
12 39 27 1,042 38
5 14 28 57 208
1000
800
600
400
200
Predicted label
accuracy=0.7668; misclass=0.2332

K-Nearest Neighbours (KNN) Classifier

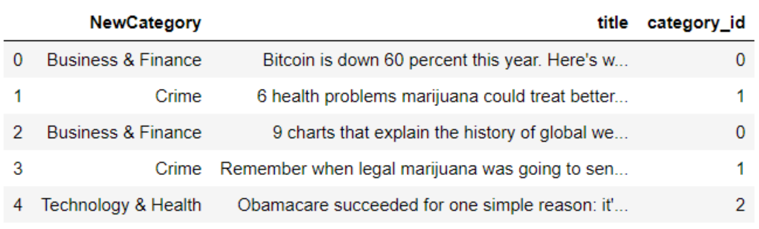
In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification. The input consists of the k closest training examples in the feature space. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Feature Extraction and Vectorization

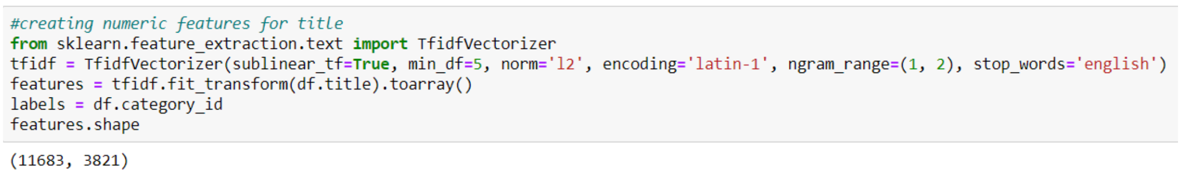
Unlike other models, Random Forest takes in a numeric value in order to perform feature engineering. In our project, the original dataset has categories as text value. In order to achieve feature engineering, we have factorized the Category column and converted them into category_ids. Later, the usual vectorization process has been implemented to follow bag of words approach.



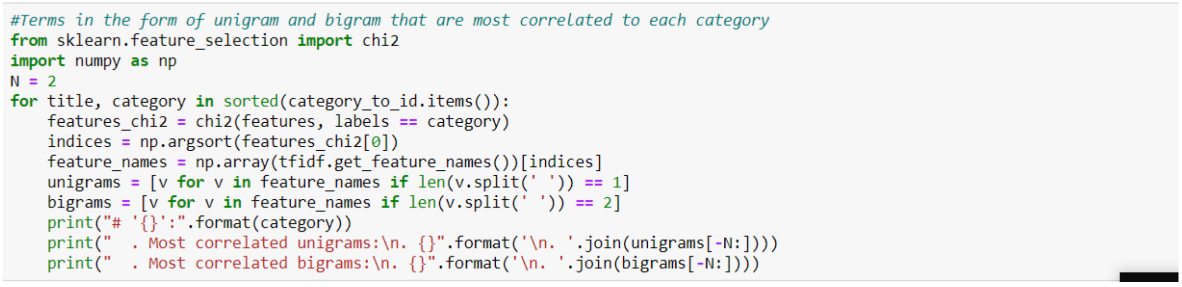
```
#creating category_id feature
from io import StringIO
col = ['NewCategory', 'title']
df = df[col]
df = df[pd.notnull(df['title'])]
df.columns = ['NewCategory', 'title']
df['category_id'] = df['NewCategory'].factorize()[0]
category_id_df = df[['NewCategory', 'category_id']].drop_duplicates().sort_values('category_id')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category_id', 'NewCategory']].values)
df.head()
```



	NewCategory	title	category_id
0	Business & Finance	Bitcoin is down 60 percent this year. Here's w...	0
1	Crime	6 health problems marijuana could treat better...	1
2	Business & Finance	9 charts that explain the history of global we...	0
3	Crime	Remember when legal marijuana was going to sen...	1
4	Technology & Health	Obamacare succeeded for one simple reason: it'...	2



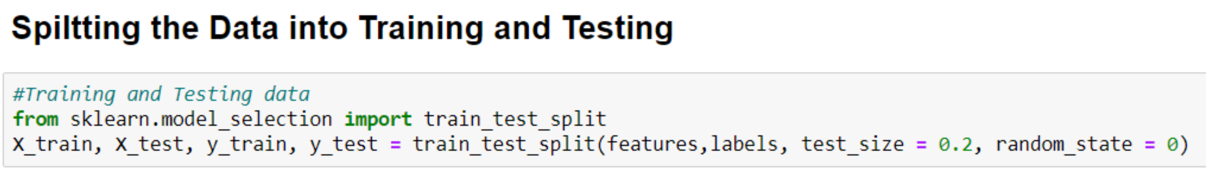
```
#creating numeric features for title
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='latin-1', ngram_range=(1, 2), stop_words='english')
features = tfidf.fit_transform(df.title).toarray()
labels = df.category_id
features.shape

(11683, 3821)
```



```
#Terms in the form of unigram and bigram that are most correlated to each category
from sklearn.feature_selection import chi2
import numpy as np
N = 2
for title, category in sorted(category_to_id.items()):
    features_chi2 = chi2(features, labels == category)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf.get_feature_names())[indices]
    unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
    bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
    print("# '{}':".format(category))
    print("  . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-N:])))
    print("  . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-N:])))
```

Splitting the data and Training the model

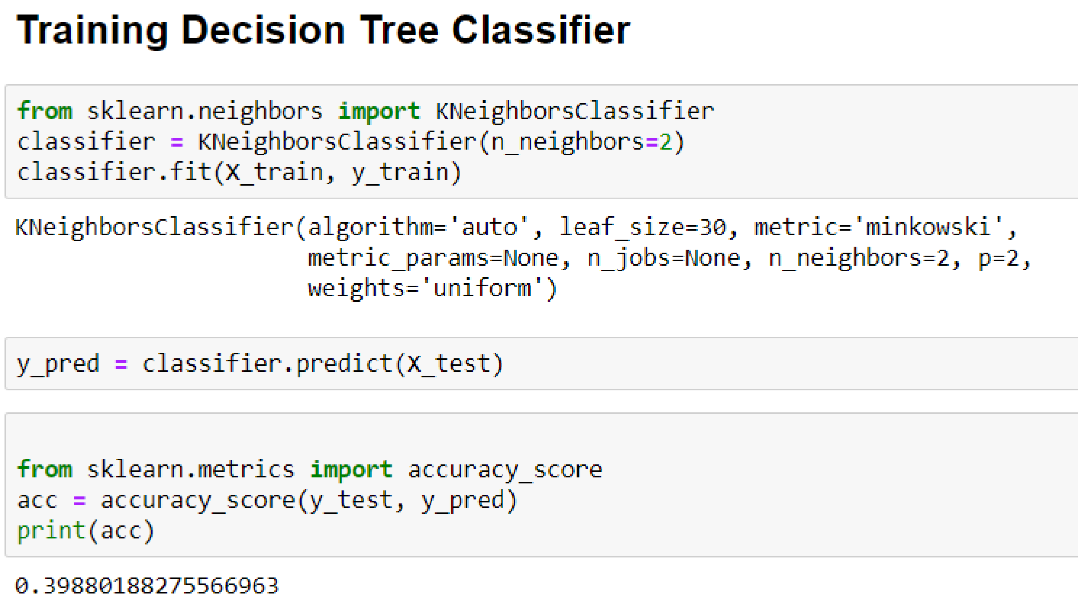
First, we have separated the columns into dependent and independent variables (or features and label). Then we have split those variables into train and test set.



Spiltting the Data into Training and Testing

```
#Training and Testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features,labels, test_size = 0.2, random_state = 0)
```

After splitting, we have generated KNN model on the training set and performed prediction on test set features.



Training Decision Tree Classifier

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=2, p=2,
                     weights='uniform')
```

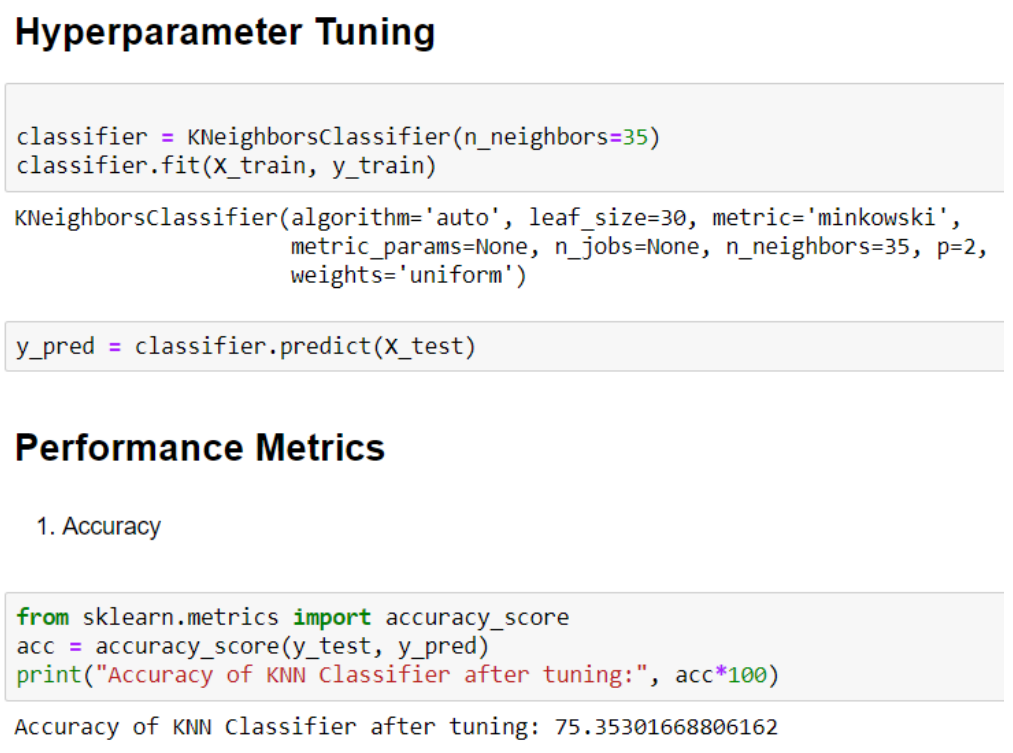
```
y_pred = classifier.predict(X_test)
```

```

from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print(acc)
```

```
0.39880188275566963
```

In order to enhance the performance, we would perform hyperparameter tuning. There are a few parameters we implicitly assumed when we did our training. In Random Forest model, n_neighbors is the parameter on which hyperparameter tuning is performed. N_neighbors denotes the number of neighbors to be taken into consideration. In most cases, the best way to determine optimal values for hyperparameters is through a grid search over possible parameter values, using cross validation to evaluate the performance of the model on your data at each value.

In our model, we changed the value of n_ neighbors parameter from 2 to 35. This modification led to increase in the accuracy of the model to 75.35%



Hyperparameter Tuning

```
classifier = KNeighborsClassifier(n_neighbors=35)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=35, p=2,
                     weights='uniform')
```

```
y_pred = classifier.predict(X_test)
```

Performance Metrics

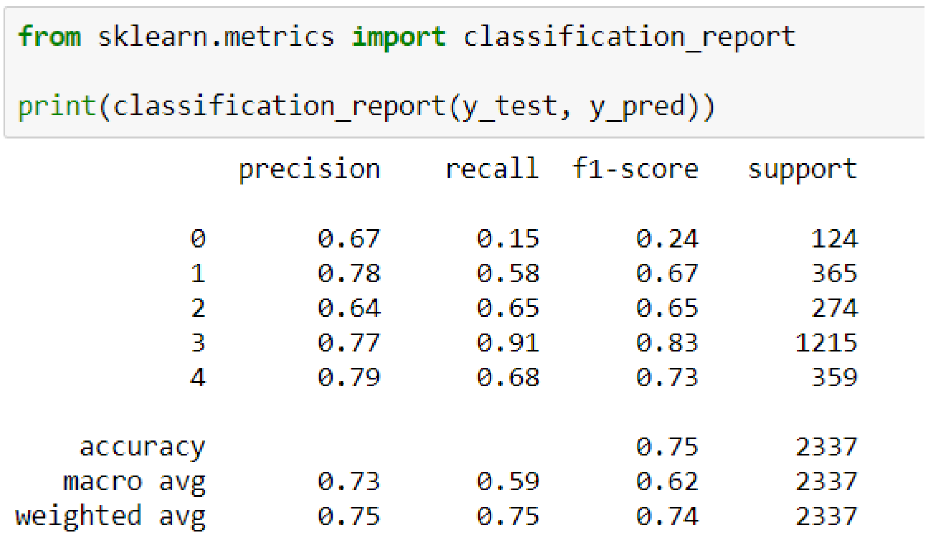
1. Accuracy

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("Accuracy of KNN Classifier after tuning:", acc*100)
```

```
Accuracy of KNN Classifier after tuning: 75.35301668806162
```

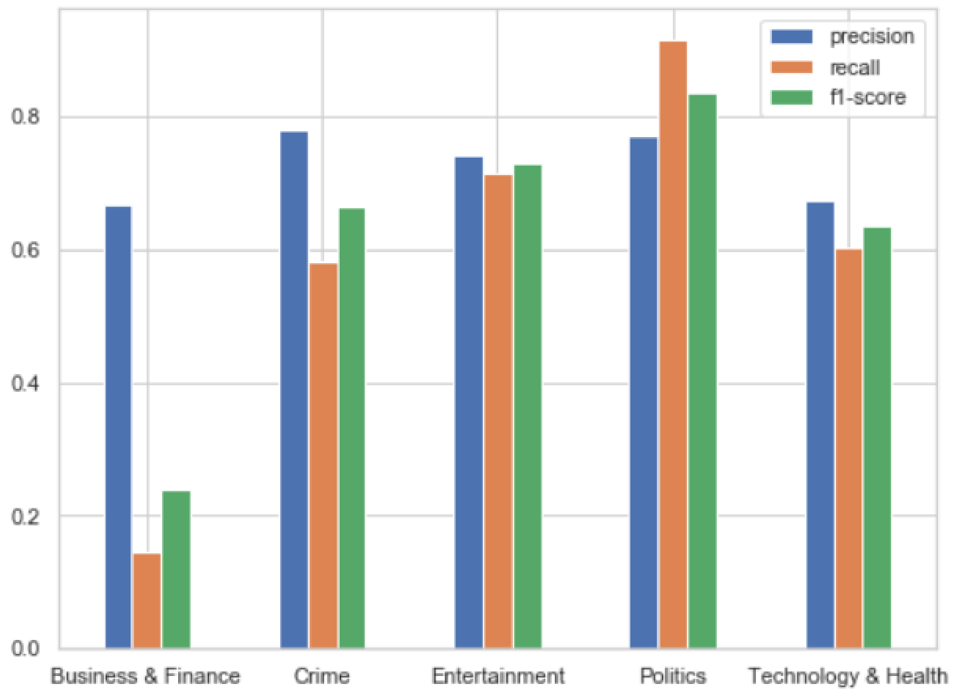
Evaluating the model

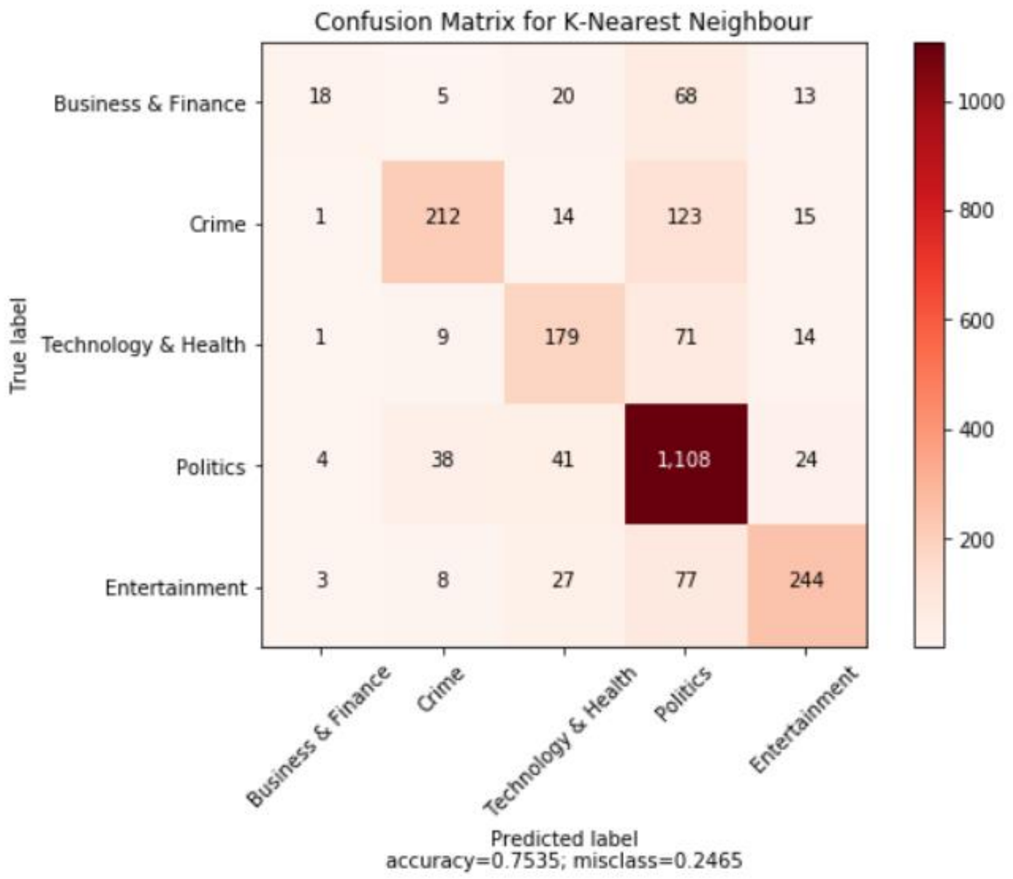
We have evaluated the model using the classification report and confusion matrix. Classification report comprises of categories mapped against the model metrics such as F1 score, Recall, support and Precision.



```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.15      0.24       124
           1       0.78      0.58      0.67       365
           2       0.64      0.65      0.65       274
           3       0.77      0.91      0.83      1215
           4       0.79      0.68      0.73       359

    accuracy                           0.75      2337
   macro avg       0.73      0.59      0.62      2337
weighted avg       0.75      0.75      0.74      2337
```





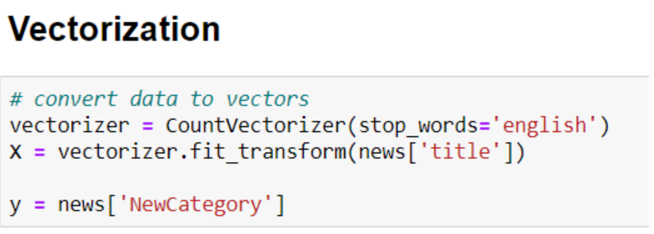
	Business & Finance	Crime	Technology & Health	Politics	Entertainment
Business & Finance	18	5	20	68	13
Crime	1	212	14	123	15
Technology & Health	1	9	179	71	14
Politics	4	38	41	1,108	24
Entertainment	3	8	27	77	244

Stochastic Gradient Descent (SGD) Classifier

In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, the batch is taken to be the whole dataset. Multiple iterations take place until the minima is reached. The data sample is randomly shuffled and selected for performing the iterations.

Feature Extraction and Vectorization

The usual vectorization process has been implemented to follow bag of words approach.

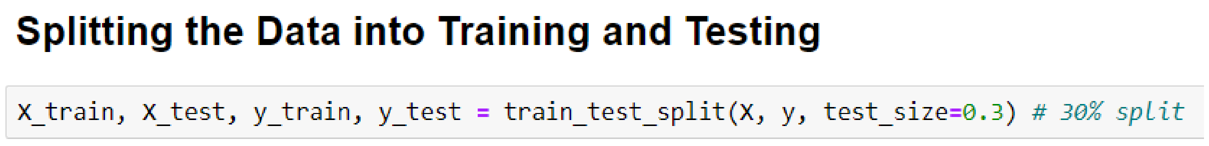


Vectorization

```
# convert data to vectors
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(news['title'])

y = news['NewCategory']
```

Splitting the data and Training the model

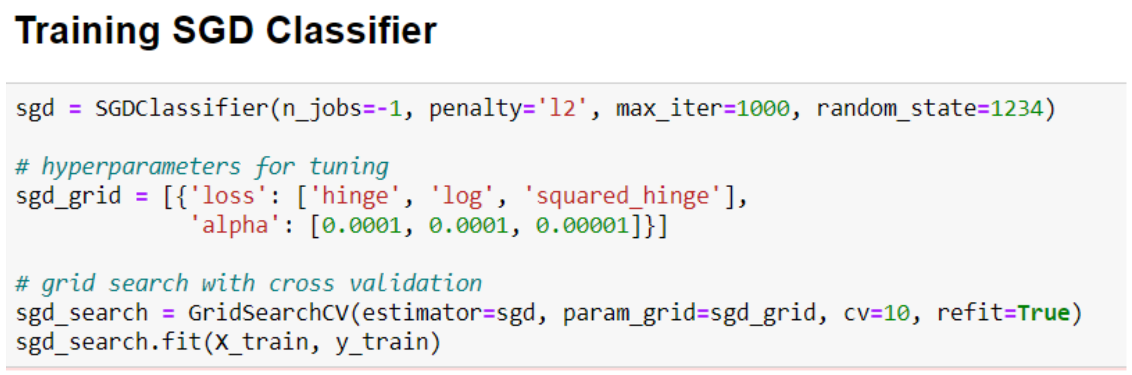
First, we have separated the columns into dependent and independent variables (or features and label). Then we have split those variables into train and test set.



Splitting the Data into Training and Testing

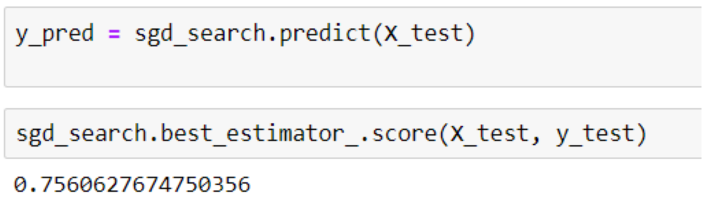
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 30% split
```

After splitting, we have generated SGD model on the training set and performed prediction on test set features.



Training SGD Classifier

```
sgd = SGDClassifier(n_jobs=-1, penalty='l2', max_iter=1000, random_state=1234)

# hyperparameters for tuning
sgd_grid = [{'loss': ['hinge', 'log', 'squared_hinge'],
             'alpha': [0.0001, 0.0001, 0.00001]}]

# grid search with cross validation
sgd_search = GridSearchCV(estimator=sgd, param_grid=sgd_grid, cv=10, refit=True)
sgd_search.fit(X_train, y_train)
```



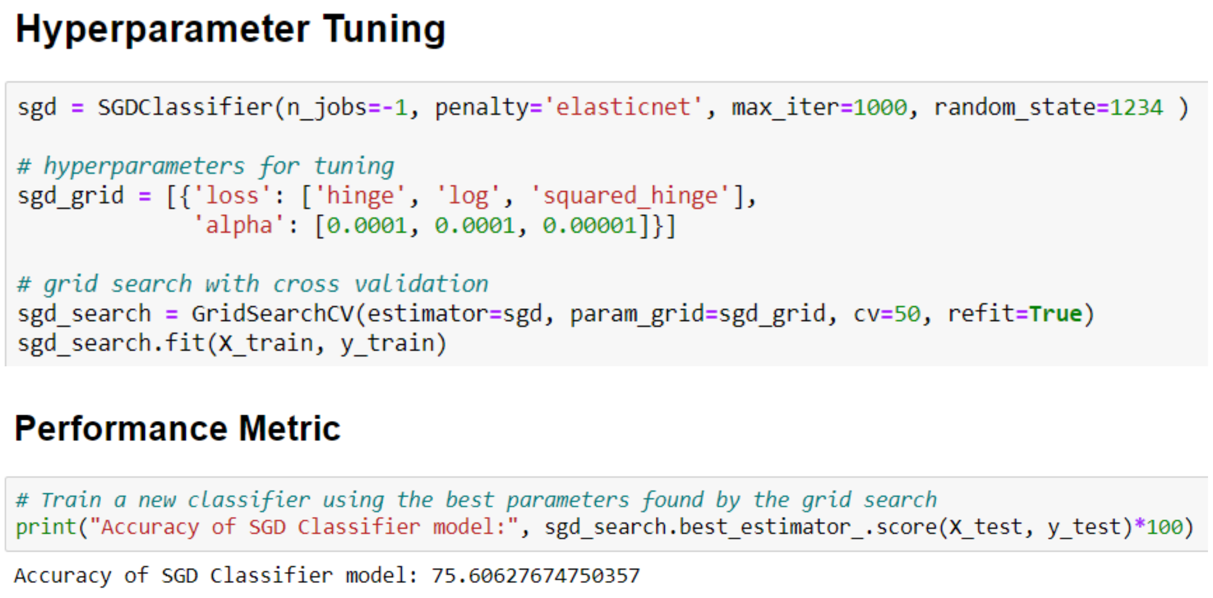
```
y_pred = sgd_search.predict(X_test)
```

```
sgd_search.best_estimator_.score(X_test, y_test)
```

```
0.7560627674750356
```

In order to enhance the performance, we would perform hyperparameter tuning. There are a few parameters we implicitly assumed when we did our training. In SGD model, we have used grid search over possible parameter values to determine optimal values for hyperparameters.

In our model, we used varied value for loss and alpha. Using grid search, we determined that the best fit for our model. Additionally, we have changed the penalty parameter from l2 to elasticnet. This modification led to increase in the accuracy of the model to 75.61%



Hyperparameter Tuning

```
sgd = SGDClassifier(n_jobs=-1, penalty='elasticnet', max_iter=1000, random_state=1234 )

# hyperparameters for tuning
sgd_grid = [{'loss': ['hinge', 'log', 'squared_hinge'],
             'alpha': [0.0001, 0.0001, 0.00001]}]

# grid search with cross validation
sgd_search = GridSearchCV(estimator=sgd, param_grid=sgd_grid, cv=50, refit=True)
sgd_search.fit(X_train, y_train)
```

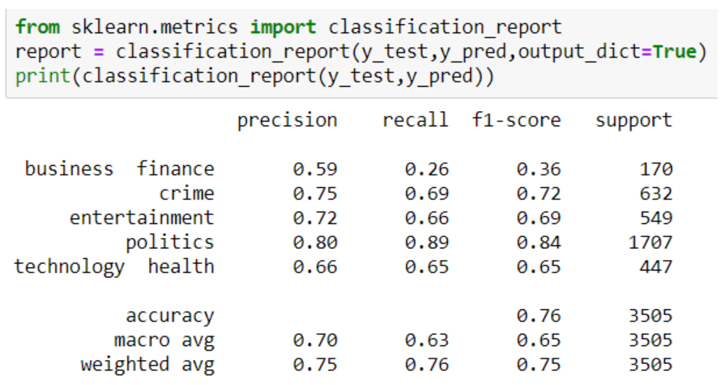
Performance Metric

```
# Train a new classifier using the best parameters found by the grid search
print("Accuracy of SGD Classifier model:", sgd_search.best_estimator_.score(X_test, y_test)*100)
```

```
Accuracy of SGD Classifier model: 75.60627674750357
```

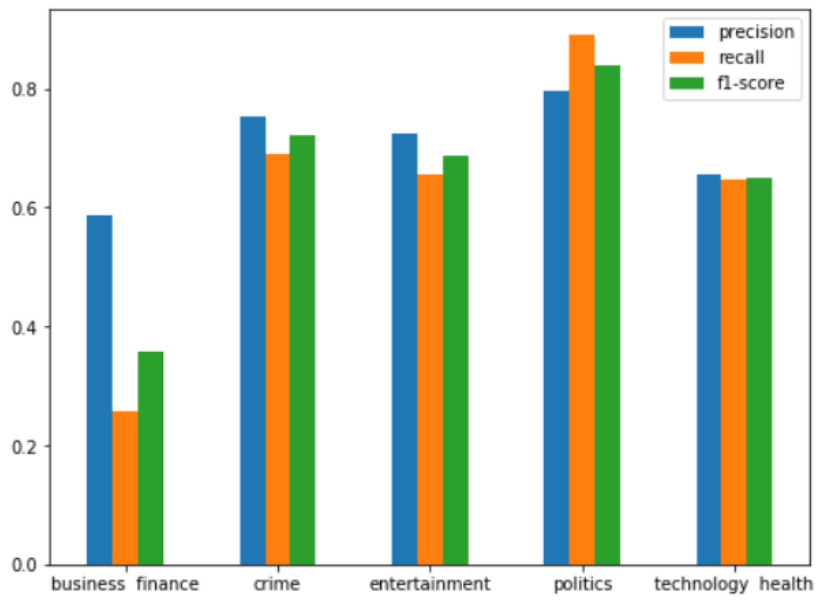
Evaluating the model

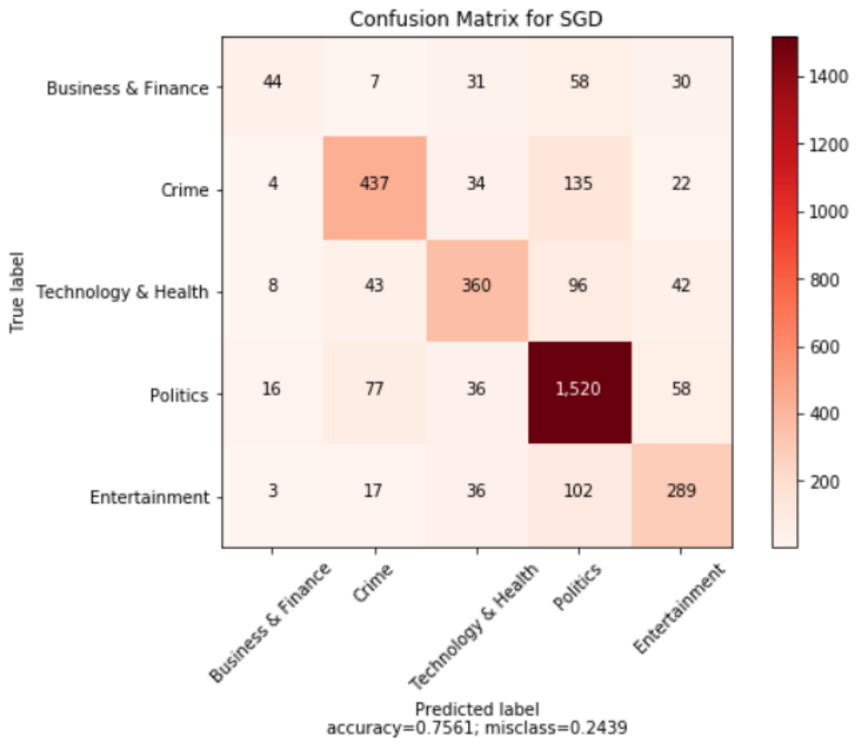
We have evaluated the model using the classification report and confusion matrix. Classification report comprises of categories mapped against the model metrics such as F1 score, Recall, support and Precision.



```
from sklearn.metrics import classification_report
report = classification_report(y_test,y_pred,output_dict=True)
print(classification_report(y_test,y_pred))
```

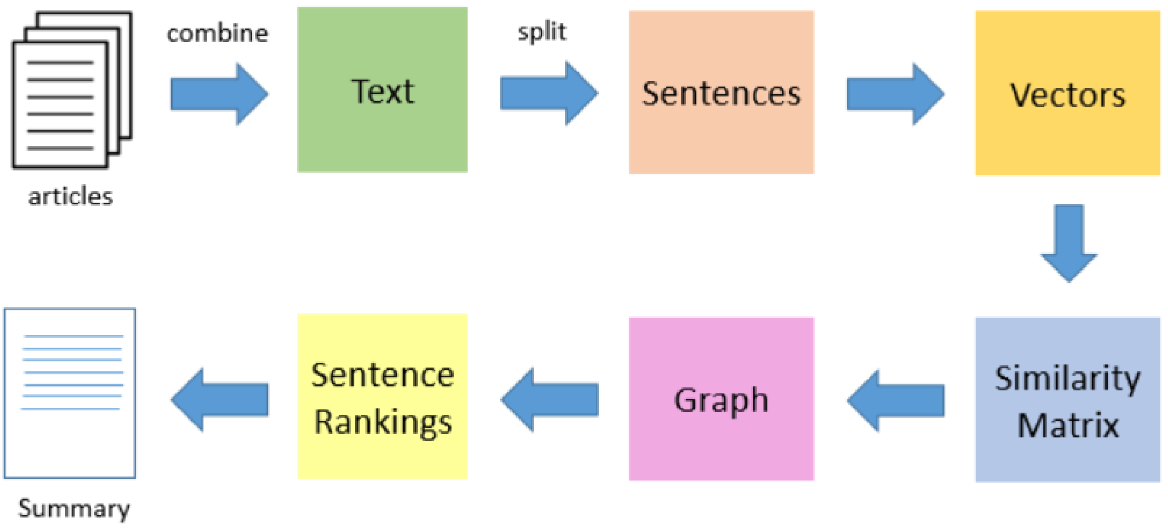
```
                    precision    recall  f1-score   support

 business  finance       0.59      0.26      0.36       170
             crime       0.75      0.69      0.72       632
     entertainment       0.72      0.66      0.69       549
          politics       0.80      0.89      0.84      1707
technology  health       0.66      0.65      0.65       447

          accuracy                           0.76      3505
         macro avg       0.70      0.63      0.65      3505
      weighted avg       0.75      0.76      0.75      3505
```





News Summarization using Natural Language Processing (NLP)

Text Summarization is one of those applications of Natural Language Processing (NLP) which is bound to have a huge impact on our lives.



How text summarization works

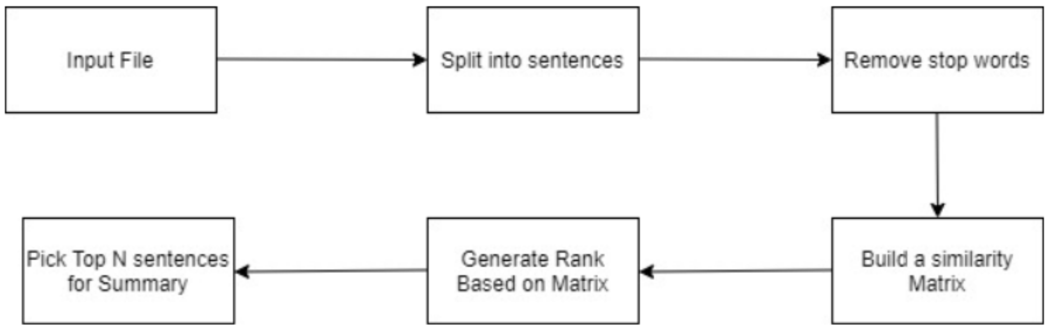
In general, there are two types of summarization, abstractive and extractive summarization. In our project we will be focusing on the extractive text summarization technique.

Extractive Summarization - Extractive methods attempt to summarize articles by selecting a subset of words that retain the most important points.

This approach weights the important part of sentences and uses the same to form the summary. Different algorithm and techniques are used to define weights for the sentences and further rank them based on importance and similarity among each other.

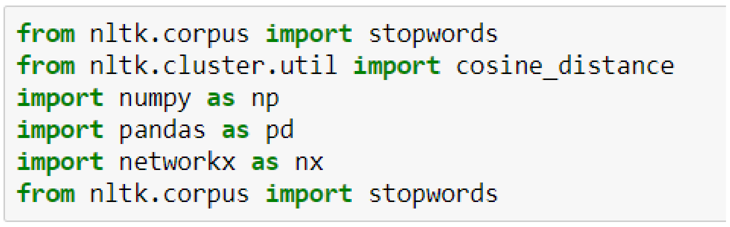
Input document → sentences similarity → weight sentences → select sentences with higher rank.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Since we will be representing our sentences as the bunch of vectors, we can use it to find the similarity among sentences. Its measures cosine of the angle between vectors. Angle will be 0 if sentences are similar.


Input File
Split into sentences
Remove stop words
Build a similarity Matrix
Generate Rank Based on Matrix
Pick Top N sentences for Summary

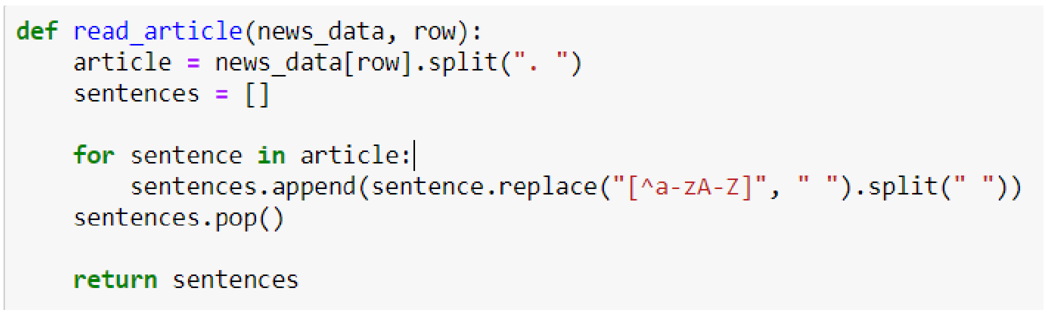
Implementation

Importing all the necessary libraries.



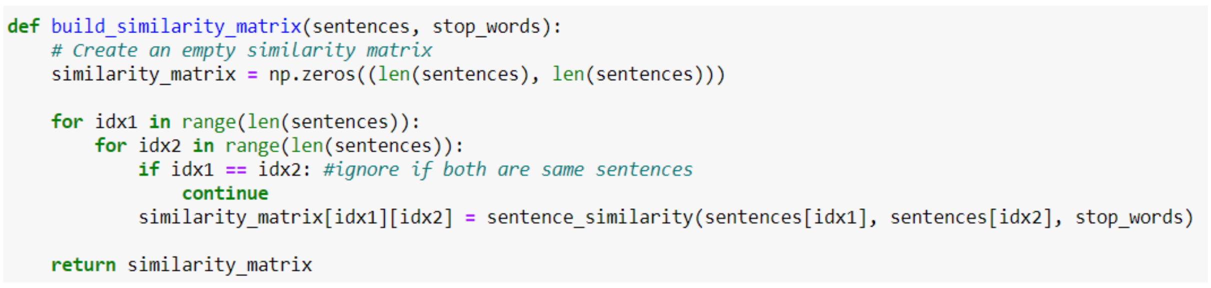
```
from nltk.corpus import stopwords
from nltk.cluster.util import cosine_distance
import numpy as np
import pandas as pd
import networkx as nx
from nltk.corpus import stopwords
```

Remove stop words, non ascii characters and generate clean sentences.



```
def read_article(news_data, row):
    article = news_data[row].split(". ")
    sentences = []

    for sentence in article:
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
    sentences.pop()

    return sentences
```

After the above step, we generate a similarity matrix to calculate similarity between the sentences.



```
def build_similarity_matrix(sentences, stop_words):
    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences), len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2: #ignore if both are same sentences
                continue
            similarity_matrix[idx1][idx2] = sentence_similarity(sentences[idx1], sentences[idx2], stop_words)

    return similarity_matrix
```

```
def sentence_similarity(sent1, sent2, stopwords=None):
    if stopwords is None:
        stopwords = []

    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]

    all_words = list(set(sent1 + sent2))

    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)

    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1

    # build the vector for the second sentence
    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1

    return 1 - cosine_distance(vector1, vector2)
```

After the similarity matrix, we generate the summary of the article based on the news body description.

```
def generate_summary(file_name, top_n=5):
    stop_words = stopwords.words('english')
    summarize_text = []
```

Specifications of Dataset

The data has been gathered from one of the widely used news site – Vox. Dataset contains news articles published on the Vox news site which cover variety of domains. The link of the dataset can be found below:

https://data.world/elenadata/vox-articles/workspace/file?filename=dsjVoxArticles.tsv

The dataset size is just over 200 MB, containing 22,994 rows and 8 columns. There are around 170 categories which could not be considered as core categories. These 170 categories could be grouped into some generic categories. So, we have manually created 5 clusters and added all the related categories in those 5 categories. These are mainly -

- Entertainment
- Health and Technology
- Politics
- Crime
- Business and Finance

Dataset contains 8 columns namely,

- **Title**: This column contains the headlines of the news article. This attribute has been used by the system during feature engineering.
- **Author**: This column contains the names of the author of that news article.
- **Category**: Every news is categorized into various news domains based on the article.
- **Published Date**: Contains the date on which the news article was published.
- **Updated Date**: This column displays the latest date on which the news article was recently updated from the time it published.
- **Slug**: This column contains the URL of the news from where this news has been retrieved.
- **Blurb**: This column contains a short description of the news article. We are ignoring this column from the dataset as our system provides a short summary from the news body.
- **Body**: This displays the news description of the article. This column is being used by the system to form a short summary of the article using NLP technique.

Results and Analysis

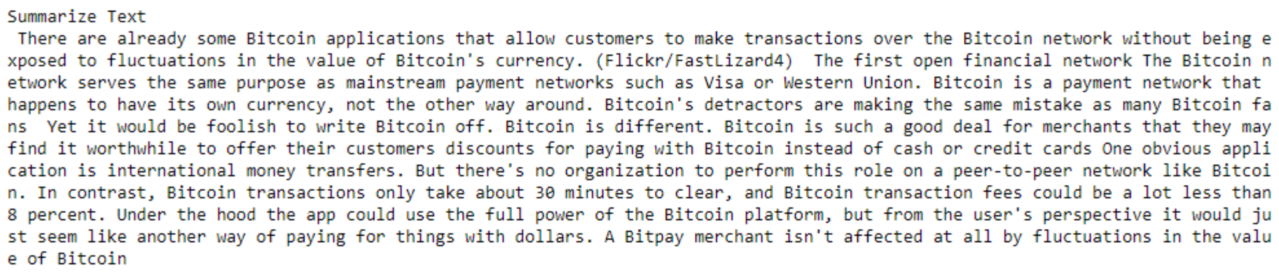
In our project we have implemented the five machine learning algorithms to classify the news articles based on some pre-defined categories. Based on our model performance and analysis, the Random Forest classifier model displayed the best accuracy amongst the other models. Random Forest classifier could classify the news articles into categories with 77.66% accuracy. Below table displays a consolidated view of accuracy and other parameters for all the models.

MODELS	ACCURACY	PRECISION	RECALL	F1-SCORE	SUPPORT
Random Forest Classifier	77.66%	0.77	0.78	0.77	2337
Support Vector Machine Classifier	76.68%	0.76	0.77	0.76	2337
Multinomial Naïve Bayes Classifier	75.82%	0.75	0.76	0.75	2337
Stochastic Gradient Descent Classifier	75.61%	0.75	0.76	0.75	3505
K-Nearest Neighbor Classifier	75.35%	0.8	0.79	0.79	3360

In our project we have implemented the text rank algorithm which a subset of Extractive and unsupervised text summarization technique. Our news articles contained more than 100 lines of text in the description which was converted into a 10-line summary without losing the integrity and the context of the article with the help of this NLP technique.

Below screenshot displays the actual news description and its corresponding summarized text.

Summarized Text:



```
Summarize Text
 There are already some Bitcoin applications that allow customers to make transactions over the Bitcoin network without being e
xposed to fluctuations in the value of Bitcoin's currency. (Flickr/FastLizard4)  The first open financial network The Bitcoin n
etwork serves the same purpose as mainstream payment networks such as Visa or Western Union. Bitcoin is a payment network that
happens to have its own currency, not the other way around. Bitcoin's detractors are making the same mistake as many Bitcoin fa
ns  Yet it would be foolish to write Bitcoin off. Bitcoin is different. Bitcoin is such a good deal for merchants that they may
find it worthwhile to offer their customers discounts for paying with Bitcoin instead of cash or credit cards One obvious appli
cation is international money transfers. But there's no organization to perform this role on a peer-to-peer network like Bitcoi
n. In contrast, Bitcoin transactions only take about 30 minutes to clear, and Bitcoin transaction fees could be a lot less than
8 percent. Under the hood the app could use the full power of the Bitcoin platform, but from the user's perspective it would ju
st seem like another way of paying for things with dollars. A Bitpay merchant isn't affected at all by fluctuations in the valu
e of Bitcoin
```

Original Text:

The markets haven't been kind to Bitcoin in 2014. The currency reached a high of nearly $1,000 in January before falling to around $350 this month, a plunge of more than 60 percent. It would be easy to write Bitcoin off as a fad whose novelty has worn off. After all, dollars seem superior in almost every respect. They're accepted everywhere, they're convenient to use, and they have a stable value. Bitcoin is an inferior currency on all three counts. Bitcoin's detractors are making the same mistake as many Bitcoin fans Yet it would be foolish to write Bitcoin off. The currency has had months-long slumps in the past, only to bounce back. More importantly, it's a mistake to think about Bitcoin as a new kind of currency. What makes Bitcoin potentially revolutionary is that it's the world's first completely open financial network. History suggests that open platforms like Bitcoin often become fertile soil for innovation. Think about the internet. It didn't seem like a very practical technology in the 1980s. But it was an open platform that anyone could build on, and in the long run it proved to be really useful. The internet succeeded because Silicon Valley have created applications that harness the internet's power while shielding users from its complexity. You don't have to be an expert on the internet's TCP/IP protocols to check Facebook on your iPhone. Bitcoin applications can work the same way. There are already some Bitcoin applications that allow customers to make transactions over the Bitcoin network without being exposed to fluctuations in the value of Bitcoin's currency. That basic model should work for a wide variety of Bitcoin-based services, allowing the Bitcoin payment network to reach a mainstream audience.   This is the very first node on the ARPANET, the predecessor to the Internet that launched in 1969. (Flickr/FastLizard4) The first open financial network The Bitcoin network serves the same purpose as mainstream payment networks such as Visa or Western Union. But there's an important difference. The Visa and Western Union networks are owned and operated by for-profit companies. If you want to build a business based on one of those networks, you have to get permission from the owner. And that's not always easy. To use the Visa network, for example, you have to comply with hundreds of pages of regulations. The Visa network also has high fees, and there are some things Visa won't let you do on its network at all. Bitcoin is different. Because no one owns or controls the network, there are no limits on how people can use it. Some people have used that freedom to do illegal things like buying drugs or gambling online. But it also means there's a low barrier to entry for building new Bitcoin-based financial services. There's an obvious parallel to the internet. Before the internet became mainstream, the leading online services were commercial networks like Compuserve and Prodigy. The companies that ran the network decided what services would be available on them. In contrast, the internet was designed for anyone to create new services. Tim Berners-Lee didn't need to ask anyone's permission to create the world wide web. He simply wrote the first web browser and web server and posted them online for others to download. Soon thousands of people were using the software and the web was born. Finding Bitcoin's killer app So what will people do with Bitcoin? It's hard to predict tomorrow's innovations, but we can get some idea of Bitcoin's potential by thinking about weaknesses of the conventional financial system. Bitcoin is such a good deal for merchants that they may find it worthwhile to offer their customers discounts for paying with Bitcoin instead of cash or credit cards One obvious application is international money transfers. Companies like Western Union and Moneygram can charge as much as 8 percent to transfer cash from one country to another, and transfers can take as long as 3 days to complete. In contrast, Bitcoin transactions only take about 30 minutes to clear, and Bitcoin transaction fees could be a lot less than 8 percent. People have been building Bitcoin ATMs to let people convert between bitcoins and their local currency. The first Bitcoin ATM was launched a little over a year ago. Today, there are 329 of them. If these devices continue to proliferate, they could become a useful alternative to conventional money-transfer services. Currently, each machine charges a transaction fee of around 3 percent, so the total cost of transferring money from one Bitcoin ATM to another is around 6 percent. That's comparable to the fees charged by incumbent money transfer services, and competition is likely to push down Bitcoin ATM fees over time. A more ambitious application for Bitcoin would be as an alternative to credit cards for daily purchases. Startups such Bitpay have already figured out how to make Bitcoin attractive to merchants as a way of accepting payments. Credit card networks charge merchants around 3 percent to process transactions. Bitpay charges 1 percent or less to accept Bitcoin payments on behalf of merchants. Bitpay merchants don't have to worry about the headache of disputed payments known as "chargebacks." Of course, for Bitcoin to take off as an alternative to credit cards, consumers will have to start using them regularly. And that's going to be a hard sell, especially if consumers are exposed to the risk of Bitcoin's volatility. But a Bitcoin-based payment app could also have some advantages. One is security. The current credit card network essentially works on the honor system, allowing any merchant to charge a credit card and relying on after-the-fact adjudication to police fraud. Bitcoin could allow companies to experiment with alternative approaches that build in security at the front end, for example by asking users to confirm a transaction on their smartphones before it's approved. That could cut fraud, reducing the hassle of disputing fraudulent payments and allowing lower fees. Moreover, Bitcoin is such a good deal for merchants that they may find it worthwhile to offer their customers discounts for paying with Bitcoin instead of cash or credit cards. That might entice bargain-hunting consumers who aren't otherwise interested in trying a new payment technology. 401(K) 2013 Using Bitcoin-the-network without Bitcoin-the-currency The biggest stumbling block for many Bitcoin services is Bitcoin's volatility. The current generation of Bitcoin "wallet" apps, which store bitcoins on behalf of users, expose consumers to fluctuations in Bitcoin's value. Ordinary consumers are unlikely to ever be comfortable with a payment system where their wealth can shrink by 10 percent or more in a single day. Fortunately, it's possible to design Bitcoin-based financial services that don't expose users to fluctuations in Bitcoin's value. Bitpay is a good example. Bitpay merchants set prices in conventional currencies such as the dollar, converting to the equivalent number of Bitcoins at the time of sale. Once the sale is made, Bitpay immediately converts it to an equivalent number of dollars and deposits the cash in the merchant's conventional bank account. This means that from the merchant's perspective, Bitpay is just another way of accepting dollars. A Bitpay merchant isn't affected at all by fluctuations in the value of Bitcoin. The same principle could apply to any other Bitcoin-based service. A consumer-friendly payments app could store a user's cash in dollars, converting them to bitcoins at the time of payment. Under the hood the app could use the full power of the Bitcoin platform, but from the user's perspective it would just seem like another way of paying for things with dollars. You're probably wondering: if Bitcoin's value is as a payment network, why not just build an open payment network based on a conventional currency like the dollar? An open, dollar-based payment network would be a great idea. The problem is that no one has figured out how to build one. Dollars in the Paypal network are worth a dollar because the Paypal company has promised to honor withdrawal requests. But there's no organization to perform this role on a peer-to-peer network like Bitcoin. Suppose there were a network called Dollarcoin that worked exactly like Bitcoin except a company called Dollarcoin Inc. promised to convert dollarcoins into dollars. Then the value of one dollarcoin would always equal one dollar. But as the manager of the Dollarcoin network, the Dollarcoin company would face pressure to comply with a variety of laws regarding fraud, money laundering, and so forth. To keep the costs of complying with those requirements under control, it would be forced to regulate who could use the network and how. (Indeed, that's exactly what happened to Paypal 15 years ago.) Over time, the Dollarcoin network could become as restrictive as conventional financial networks. Bitcoin's openness depends on the fact that no one owns the network. And with no owner, there's no one to guarantee that bitcoins have a predictable value. Many of Bitcoin's early adopters were acolytes of Ron Paul's brand of hard-money libertarianism. They were attracted to the promise of a currency whose supply was outside of state control, and as a consequence, Bitcoin has gained a reputation as the second coming of the gold standard. That, in turn, has made mainstream economists who are hostile to Ron Paul and the gold standard hostile to Bitcoin. But in reality, the case for Bitcoin simply doesn't have much to do with its unorthodox monetary policy. Bitcoin is a payment network that happens to have its own currency, not the other way around. It's worth taking seriously whether or not you agree with Ron Paul's views on the Federal Reserve.

Conclusion & Future Scope

An implementation of news classification and summarization is bestowed in this project. The purpose of this project was to gain insights into the information retrieval field in terms of how classification and summarization functionality works. Our system, InShorts, caters to two sets of users – a news reader and a news analyst. We have categorized the news article dataset with high accuracy into categories such as Entertainment, Politics, Crime, Health & Technology and Business & Finance and created relevant visualizations for the same. We have implemented five supervised machine learning classification algorithms to determine which of the models were able to classify the news article with the highest accuracy rate. We performed data cleaning and data pre-processing to convert raw dataset into a meaningful and useable dataset. In addition, stop words filtering using frequency-based stop words removal approach is also implemented. In continuation from the above step, we performed feature engineering and model training. We optimized the results of the testing dataset by implementing hyperparameter tuning. We concluded that Random Forest Classifier model displayed the highest rate of accuracy amongst all the classifier models. The second part of our project was text summarization. We achieved the summarization of news articles by implementing NLP techniques. We could successfully summarize a page long news article into a 10-line summary without losing the context of the article.

In future these algorithms can be tested on larger dataset. Moreover, these algorithms can be improved so that efficiency of categorization could be enhanced. A combination of algorithm can be used in order to achieve clustering in a faster way. Additionally, for feature engineering, we could implement other techniques like doc2vec for better results. We could also use a predictive model to predict the categories of the news articles based on the title of the news and compare the results with this existing model. For the summarization, we could use abstractive text summarization technique which would give better and more realistic results.

References and resources

[1] Classification of News Dataset http://cs229.stanford.edu/proj2018/report/183.pdf
[2] http://www.iosrjournals.org/iosr-jce/papers/Vol18-issue1/Version-3/D018132226.pdf
[3] Li, Yanjun, Soon M. Chung, and John D. Holt. "Text document clustering based on frequent word meaning sequences.”, Data & Knowledge Engineering.
[4] Luo, Congnan, Yanjun Li, and Soon M. Chung. "Text document clustering based on neighbors.", Data & Knowledge Engineering.
[5] Zheng, Hai-Tao, Bo-Yeong Kang, and Hong-Gee Kim. "Exploiting noun phrases and semantic relationships for text document clustering ,179(13) ,2009,2249-2269.
[6] Scikit-learn 0.20.0 documentation https://scikit-learn.org/0.20/_downloads/scikit-learndocs.pdf
[7] Scikit-learn Text Feature Extraction https://scikitlearn.org/stable/modules/ feature_extraction.html#text-feature-extraction
[8] Scikit-learn Support Vector Machine Classifier https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[9] Scikit-learn Random Forest Classifier https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[10] Scikit-learn K-Nearest Neighbors Classifier https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[11] Scikit-learn Stochastic Gradient Descent Classifier https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[12] Scikit-learn Multinomial Naïve Bayes Classifier https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[13] Introduction to Machine Learning with Python A Guide for Data Scientists by Andreas C.Müller, Sarah Guido (z-lib.org).pdf
[14] http://sifaka.cs.uiuc.edu/~wang296/Course/IR_Fall/docs/Projects/Samples/6.pdf
[15] https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank/python/