DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY, JALANDHAR(144011), PUNJAB

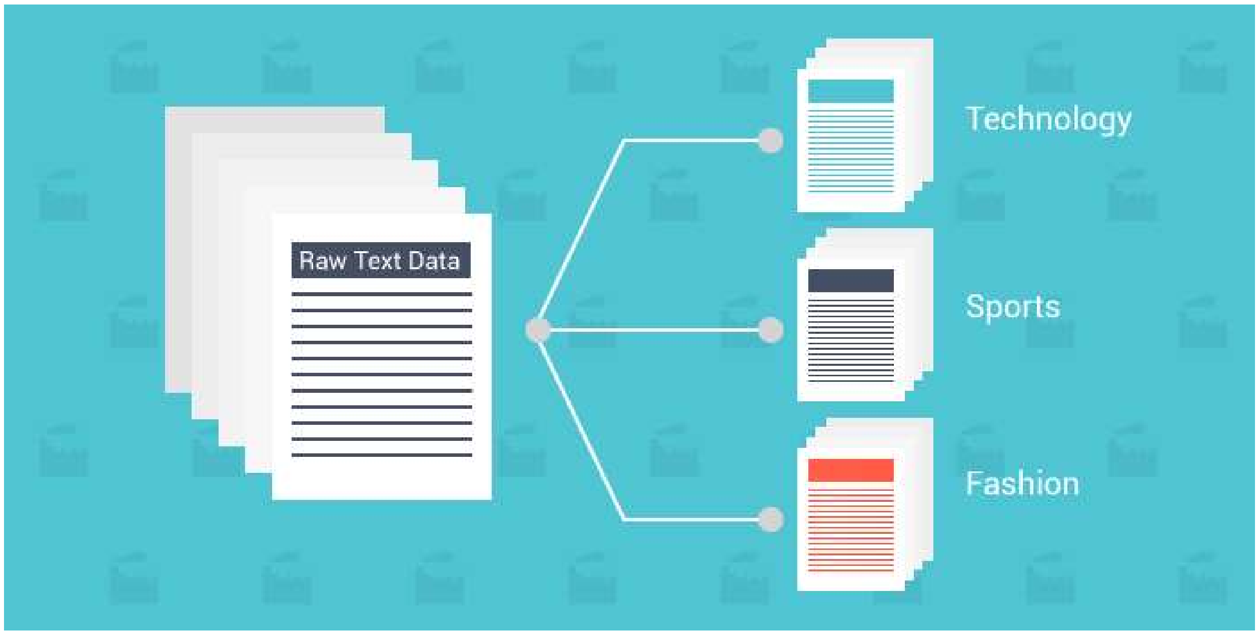
MACHINE LEARNING (CSPC-204)
MINI PROJECT

Submitted By:
Shreya Das (20103134)
Simranjeet Kaur (20103139)
Smriti Chaurasia (20103141)
Branch- CSE
Section-B Group-2

Submitted to:
Dr. Jagdeep Kaur

MACHINE LEARNING CSPC-204

MACHINE LEARNING: MINI PROJECT
TEXT CLASSIFICATION ON YOUTUBE DATA



Introduction

In today's age there are so many videos available over the internet on various platforms. Youtube is one such platform which is most popular among the users. Because of the abundance of content it can be time-consuming to navigate through it. Classifying the content and products into categories help users to easily search and navigate within website or application. We use text analysis here.

Text classification or Text Categorization is the activity of labeling natural language texts with relevant categories from a predefined set. In laymen terms, text classification is a process of extracting generic tags from unstructured text. These generic tags come from a set of pre-defined categories.

Dataset Information

Our dataset is Video Classification Dataset. It contains 4 attributes: Video Id, Title, Description and Category.

Video Id - It is a random string that uniquely determines the video.

Title- It corresponds to the title given by the uploader/creator.

Description- Brief explanation of the content of the video by the creator.

Category- The classes in which the videos will be classified.

Problem Statement

We aim refers to analyse the contents of raw text and decide which category it belongs to. There are many text classification techniques for example,S Support Vector Machines, Adaboost, and LSTM but we will be working with the Naive Bayes Technique. Our task is to classify a video into different classes based on its title and description using the Naive Bayes technique and analyzing its performance.

The classes chosen are:

- Travel
- Science and Technology
- Food
- Art and Music
- History
- Manufacturing

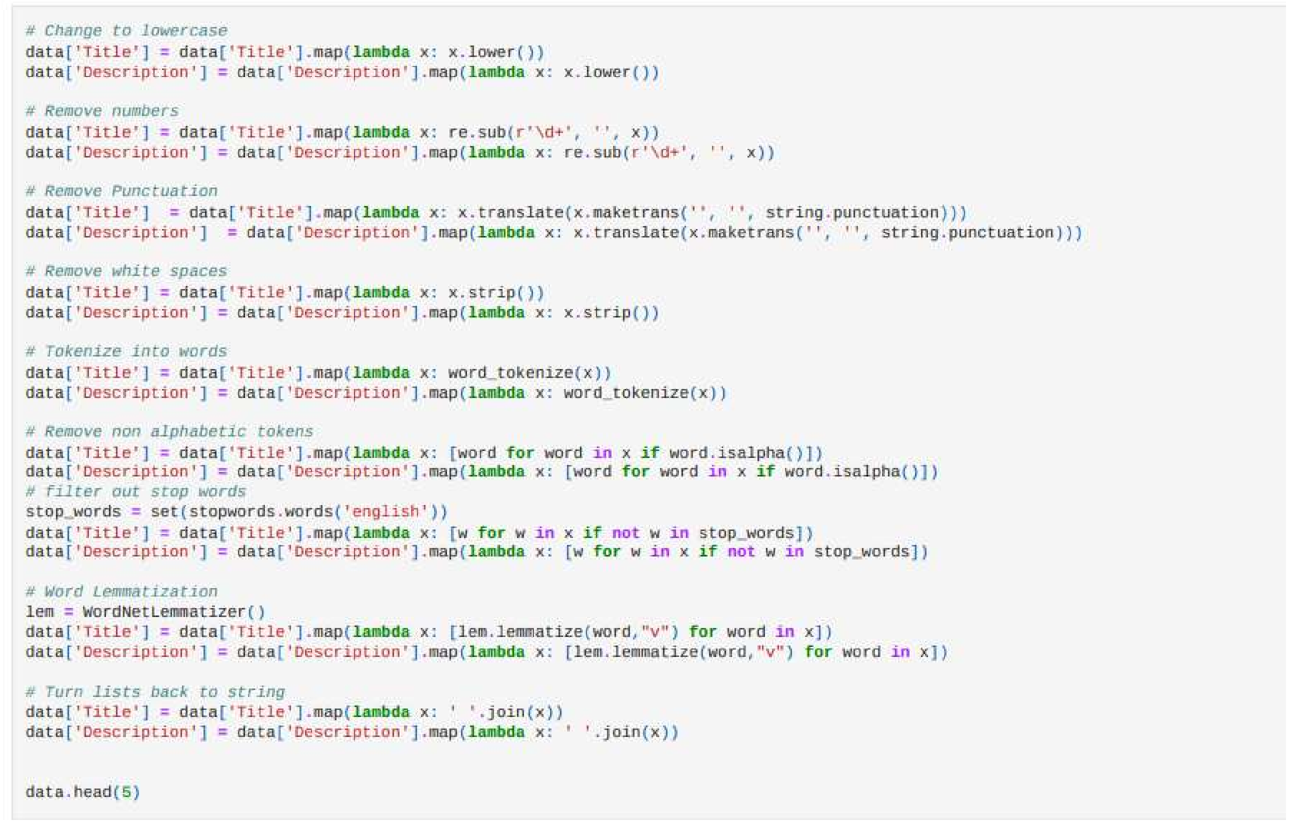
Algorithms Used

Data Cleaning and Pre-processing

The first step of my data pre-processing process is to handle the missing data. Fortunately, there exist only 334 missing values out of 9999 total samples, so it would not affect model performance during training.

The 'Video Id' column is not really useful for our predictive analysis, and thus it would not be chosen as part of the final training set, so we do not have any pre-processing steps for it. There are 2 columns of importance here, namely — **Title** and **Description**, but they are unprocessed raw texts. Therefore, we will filter out the noisiness.

Code snippet:



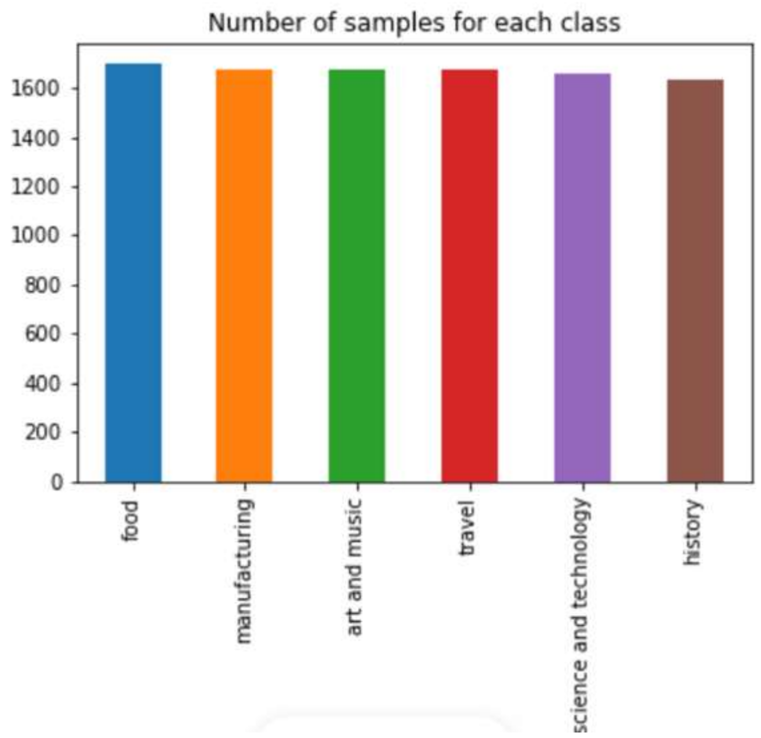
```
# Change to lowercase
data['Title'] = data['Title'].map(lambda x: x.lower())
data['Description'] = data['Description'].map(lambda x: x.lower())

# Remove numbers
data['Title'] = data['Title'].map(lambda x: re.sub(r'\d+', '', x))
data['Description'] = data['Description'].map(lambda x: re.sub(r'\d+', '', x))

# Remove Punctuation
data['Title']  = data['Title'].map(lambda x: x.translate(x.maketrans('', '', string.punctuation)))
data['Description']  = data['Description'].map(lambda x: x.translate(x.maketrans('', '', string.punctuation)))

# Remove white spaces
data['Title'] = data['Title'].map(lambda x: x.strip())
data['Description'] = data['Description'].map(lambda x: x.strip())

# Tokenize into words
data['Title'] = data['Title'].map(lambda x: word_tokenize(x))
data['Description'] = data['Description'].map(lambda x: word_tokenize(x))

# Remove non alphabetic tokens
data['Title'] = data['Title'].map(lambda x: [word for word in x if word.isalpha()])
data['Description'] = data['Description'].map(lambda x: [word for word in x if word.isalpha()])
# filter out stop words
stop_words = set(stopwords.words('english'))
data['Title'] = data['Title'].map(lambda x: [w for w in x if not w in stop_words])
data['Description'] = data['Description'].map(lambda x: [w for w in x if not w in stop_words])

# Word Lemmatization
lem = WordNetLemmatizer()
data['Title'] = data['Title'].map(lambda x: [lem.lemmatize(word,"v") for word in x])
data['Description'] = data['Description'].map(lambda x: [lem.lemmatize(word,"v") for word in x])

# Turn lists back to string
data['Title'] = data['Title'].map(lambda x: ' '.join(x))
data['Description'] = data['Description'].map(lambda x: ' '.join(x))


data.head(5)
```

We cannot feed our text data as it is to our machine learning models, no matter how clean it is. thus , we need to convert them into numerical based features. Since, the output variable('Category') is categorical, we need to encode each class as a number using **Label Encoding.**

The main piece of information for each sample is the raw text data. To extract data from the text as features and represent them in a numerical format, a very common approach is to **vectorize them**. The Scikit-learn library contains the 'TF-IDFVectorizer' for this very purpose. **TF-IDF**(Term Frequency-Inverse Document Frequency) calculates the frequency of each word inside and across multiple documents to identify the importance of each word.

Data Analysis and Feature Exploration

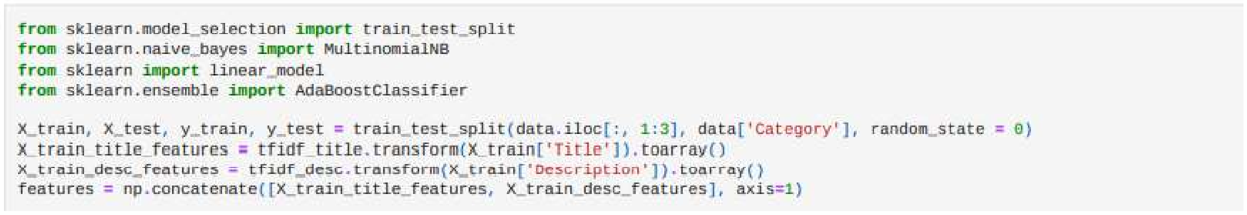
The distribution of classes so as to check for an imbalanced number of samples.



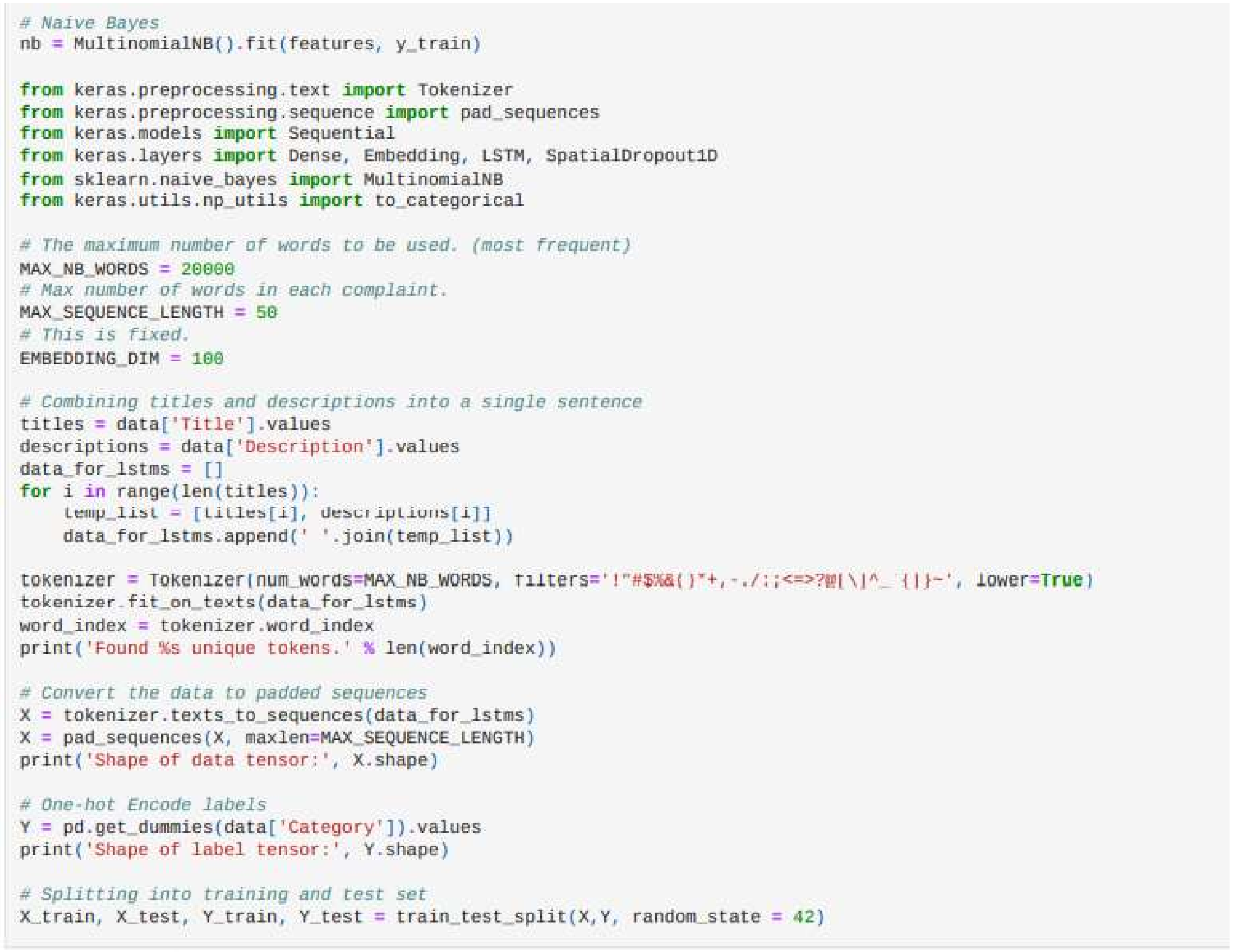
Modeling and Training (Using Naive Bayes)

The dataset is split into Train and Test sets. Features for Title and Description are computed independently and then concatenated to construct a final feature matrix.

Code snippet:



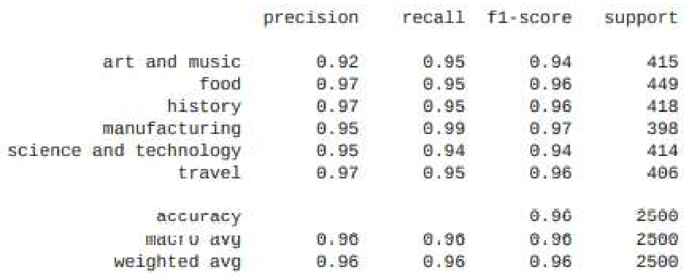
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn import linear_model
from sklearn.ensemble import AdaBoostClassifier

X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, 1:3], data['Category'], random_state = 0)
X_train_title_features = tfidf_title.transform(X_train['Title']).toarray()
X_train_desc_features = tfidf_desc.transform(X_train['Description']).toarray()
features = np.concatenate([X_train_title_features, X_train_desc_features], axis=1)
```



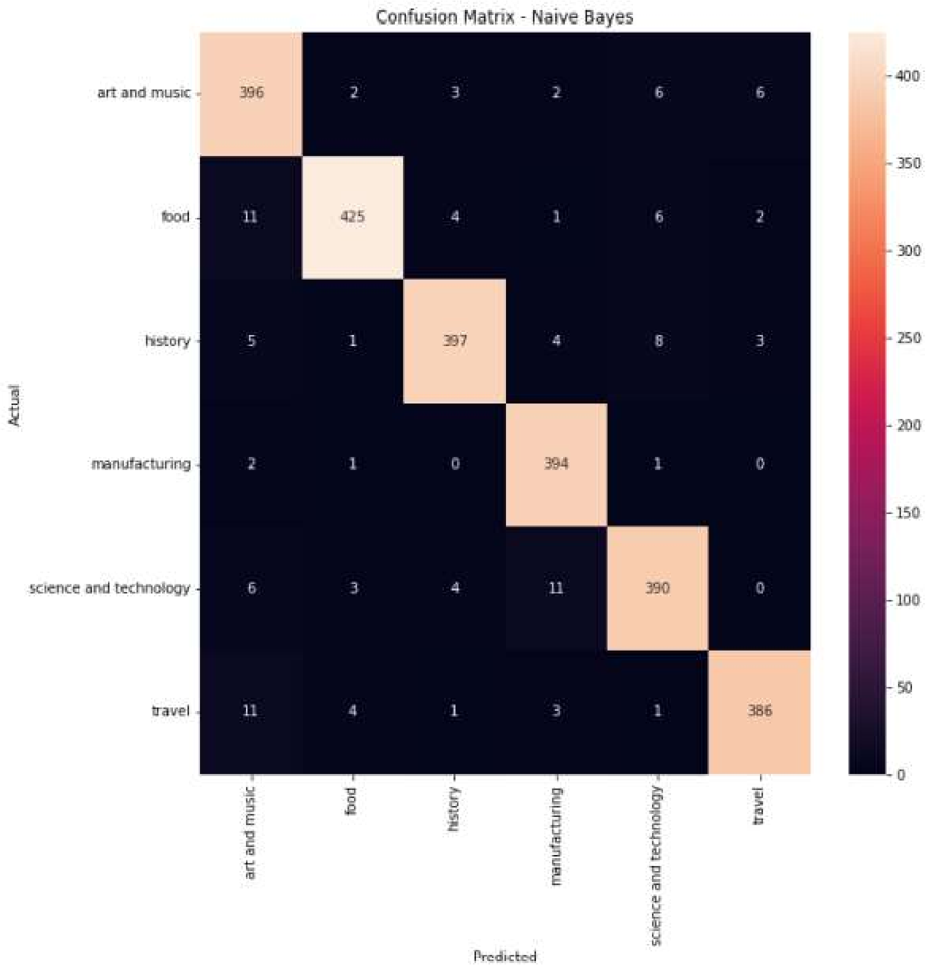
```
# Naive Bayes
nb = MultinomialNB().fit(features, y_train)

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.naive_bayes import MultinomialNB
from keras.utils.np_utils import to_categorical

# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 20000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 50
# This is fixed.
EMBEDDING_DIM = 100

# Combining titles and descriptions into a single sentence
titles = data['Title'].values
descriptions = data['Description'].values
data_for_lstms = []
for i in range(len(titles)):
    temp_list = [titles[i], descriptions[i]]
    data_for_lstms.append(' '.join(temp_list))

tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(data_for_lstms)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

# Convert the data to padded sequences
X = tokenizer.texts_to_sequences(data_for_lstms)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)

# One-hot Encode labels
Y = pd.get_dummies(data['Category']).values
print('Shape of label tensor:', Y.shape)

# Splitting into training and test set
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state = 42)
```

Results

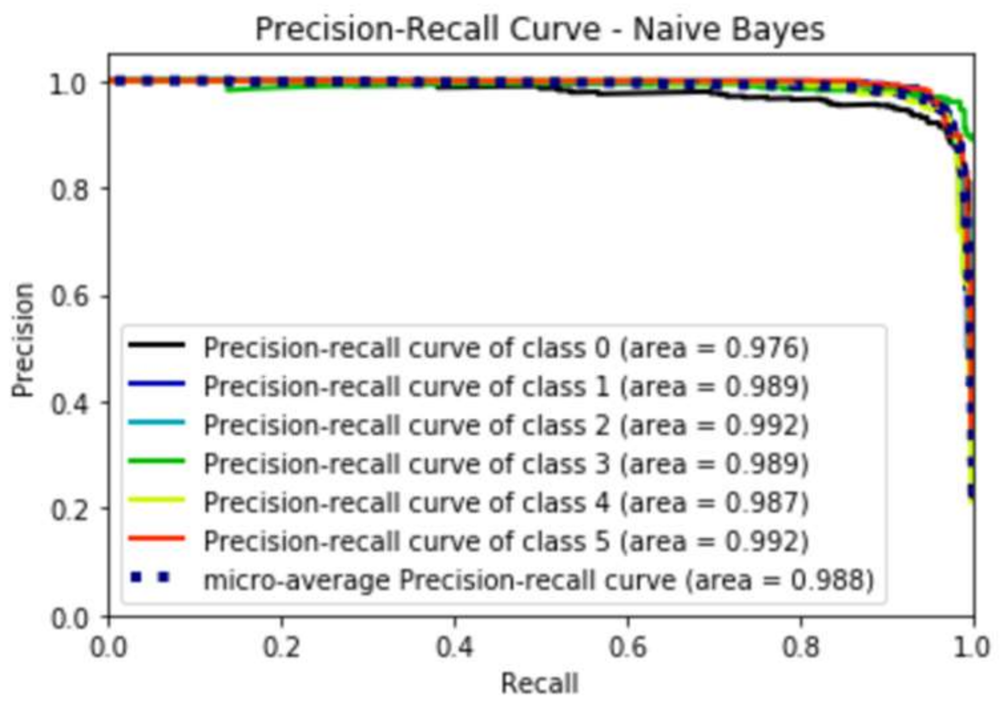
Performance Evaluation(Using Confusion matrix)



```
                        precision    recall  f1-score   support

         art and music       0.92      0.95      0.94       415
                  food       0.97      0.95      0.96       449
               history       0.97      0.95      0.96       418
         manufacturing       0.95      0.99      0.97       398
science and technology       0.95      0.94      0.94       414
                travel       0.97      0.95      0.96       406

              accuracy                           0.96      2500
             macro avg       0.96      0.96      0.96      2500
          weighted avg       0.96      0.96      0.96      2500
```



Conclusion



Naive Bayes Classifier considers the features as independent, it does not take into account any interactions between different features. There are other techniques which work better than Naive Bayes for text classification here.

References

- Simplilearn
- Analyticsvidhya
- datacamp

THANK YOU!!