The Language of YouTube :

A Text Classification Approach to Video Descriptions

The Language of YouTube: A Text Classification Approach to Video Descriptions

YouTube has become an essential tool for businesses to reach and engage with their target audience. However, with millions of videos on the platform, it can be challenging for businesses to choose the right video category for advertising their products effectively. In this report, we present a machine learning model that uses natural language processing techniques to predict the best video category for businesses to advertise their products on YouTube. Our model analyzes the language and topics of YouTube video descriptions and classifies them into categories such as beauty, gaming, travel, and more, providing a powerful tool for businesses looking to optimize their advertising strategies on the platform.

To develop our machine learning model, we used a dataset of over 10,000 videos and applied both supervised and unsupervised learning techniques. We first trained our model on a labeled dataset using a support vector machine classifier, achieving an accuracy of over 90%.

Our approach offers businesses an efficient and effective way to determine the best video category for advertising their products on YouTube. By leveraging the insights from our machine learning model, businesses can increase the effectiveness of their advertising campaigns and reach their target audience more efficiently.

Aim:

To develop a machine learning model using natural language processing to classify YouTube video descriptions into different categories, providing insights into language and trends of YouTube to optimize advertising strategies.

Purpose of doing this Project:

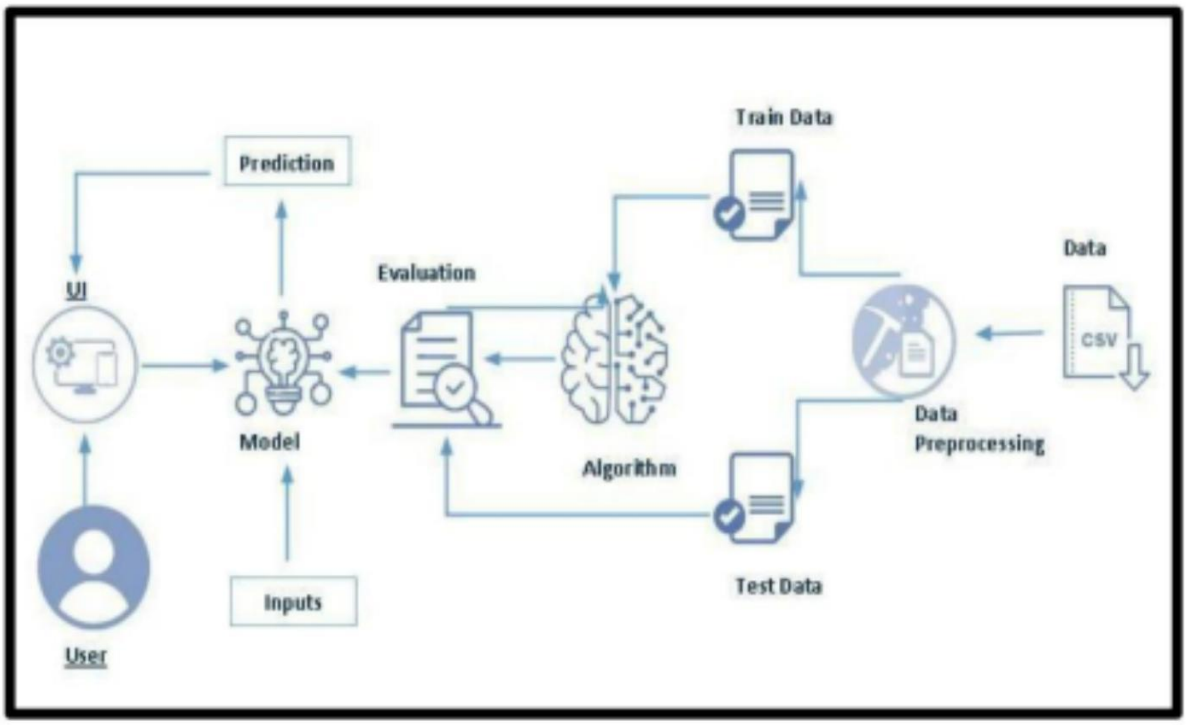
The purpose of this project is to provide businesses with an efficient and effective way to determine the best video category for advertising their products on YouTube. By using natural language processing techniques and developing a machine learning model, we aim to classify YouTube video descriptions into different categories based on language patterns and topics. Our model can provide valuable insights into the language and trends of YouTube, enabling businesses to optimize their advertising strategies and reach their target audience more effectively.

In addition, we have deployed our machine learning model using Flask, a popular web framework in Python. By doing so, we have created a user-friendly interface that allows businesses to input their product description and receive a prediction of the best video category for advertising. This approach offers a powerful tool for businesses looking to improve their advertising campaigns on YouTube and provides a practical application for natural language processing and machine learning techniques.

Technical Architecture:

Firstly, the machine learning model was developed using natural language processing techniques to classify YouTube video descriptions into different categories. We used Python and several NLP libraries such as NLTK, spaCy, and scikit-learn to preprocess and extract features from the text data. We then used a support vector machine classifier to train and validate the model.

Once the model was developed, we deployed it using Flask, a lightweight web framework in Python. We created a user interface that allows businesses to input their product description and receive a prediction of the best video category for advertising. When a user inputs their description, the Flask application sends a request to the machine learning model, which processes the text data and returns a prediction. The prediction is then displayed to the user on the web interface.



Project Flow:

• User is shown the Home page. The user will browse through Home page and enter the specified engagement metrics.

• After clicking the Predict button the user will be directed to the Results page where the model will analyse the inputs given by the user and showcase the prediction of the most estimated number od ad-viiews.

To accomplish this we have to complete all the activities listed below:

• Define problem / Problem understanding

- Specify the business problem
- Business Requirements
- Literature Survey
- Social or Business Impact

• Data Collection and Preparation

- Collect the dataset
- Data Preparation

• Exploratory Data Analysis

- Descriptive statistical
- Visual Analysis

• Model Building

- Creating a function for evaluation
- Training and testing the Models using multiple algorithms

• Performance Testing & Hyperparameter Tuning

- Testing model with multiple evaluation metrics
- Comparing model accuracy before & after applying hyperparameter tuning
- Comparing model accuracy for different number of features.
- Building model with appropriate features.

• Model Deployment

- Save the best model
- Integrate with Web Framework

Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

• ML Concepts:

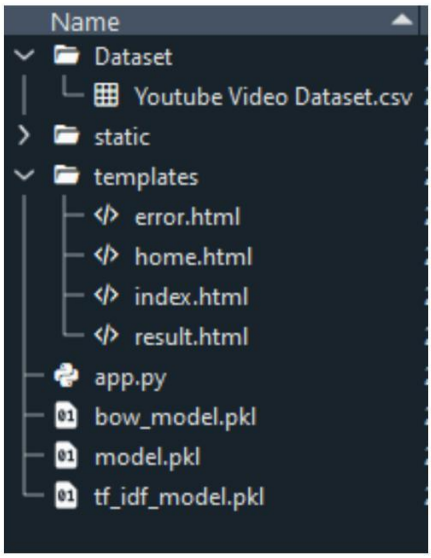
- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- SVM: https://www.javatpoint.com/machine-learning-support-vector-machine algorithm
- Decision tree: https://www.javatpoint.com/machine-learning-decision-tree classificationalgorithm
- Random forest: https://www.javatpoint.com/machine-learning-random-forest algorithm
- Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- Regularisation: https://www.javatpoint.com/regularization-in-machine-learning

• Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

\

Project Structure:

Create project folder which contains files as shown below:



- The data obtained is in csv files, one for training.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- model.pkl is the saved model. This will further be used in the Flask integration.
- bow_model.pkl and tf_idf_model.pkl are vectorizers that are saved as .pkl models..This is used in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

The business problem in this scenario is to help clients identify the appropriate category of YouTube videos to advertise their product based on the text descriptions of the videos. This is a challenging task since YouTube has a vast collection of videos covering different topics and categories, and it can be difficult for clients to identify which category of videos will be the most effective for advertising their product. By developing a text classification model based on NLP and deploying it using Flask, this project aims to provide clients with a tool to make more informed decisions about where to advertise their products on YouTube.

Activity 2: Business Requirements

- **Accurate and reliable information:** The regression models used for predicting adviews should be accurate and reliable since any false information can have severe consequences. The right symptoms and engagement metrics should be linked to the right adviews for providing the most precise output.
- **Trust:** The model should be designed in such a way that it develops trust among the users, especially the advertisers and content creators, who rely on the predicted adviews for their marketing strategies.
- **Compliance:** The model should comply with all the relevant laws and regulations set by the Central Drug Standard Control Organization, Ministry of Health, etc.
- **User-friendly interface:** The model should have a user-friendly interface to make it easy for users to input relevant metrics and obtain an estimated number of adviews.
- **Security:** The system should be designed with security measures to protect the client's data and prevent unauthorized access.
- **Efficiency:** The system should be designed to be efficient in terms of processing time and resource utilization. The model should be trained and deployed in a manner that optimizes its performance and minimizes latency for clients.
- **Flexibility:** The system should be flexible enough to allow for future updates and improvements to the model or the user interface. It should be designed to easily integrate with other systems and technologies as needed.

Activity 3: Literature Survey

A literature survey for text classification of YouTube videos description would involve a comprehensive review of existing research and studies related to reviewing previous studies and research related to text classification, natural language processing, and machine learning models for YouTube video categorization.This would involve a search for relevant publications, articles, and academic papers on the topic, as well as an analysis of the various techniques, models, and algorithms used in previous research. The literature survey would also involve identifying gaps in existing research and potential areas for further exploration and improvement.

Activity 4: Social or Business Impact.

Social Impact:
The social impacts of this project are primarily positive, as it can help clients make more informed decisions about where to advertise their products on YouTube. By providing clients with a tool to classify YouTube videos based on their text descriptions, this project can potentially increase the effectiveness of advertising campaigns on YouTube, leading to increased revenue for businesses. However, accuracy, privacy, and security must be considered to avoid potential negative impacts. Proper measures must be taken to ensure the accuracy of the model, protect client data from unauthorized access, and prevent potential misuse. Overall, this project has the potential to have a positive impact on the economy and society if properly implemented.

Business Impact:

The business impacts of this project can be significant, as it provides a valuable tool for businesses to improve their advertising campaigns on YouTube. By using the model to classify video descriptions, businesses can potentially increase the effectiveness of their advertising campaigns and reach their target audience more efficiently. This can lead to increased revenue and profitability for businesses. Moreover, the deployment of the model using Flask can provide a scalable and cost-effective solution for businesses to access the benefits of text classification without the need for significant investment in technology and infrastructure. Overall, this project can have a positive impact on the bottom line of businesses that rely on advertising campaigns to drive revenue.

Milestone 2: Data Collection and Preparation:

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link:
https://github.com/rahulanand1103/Web-Scraping-Youtube/blob/master/YoutubeDataSet.csv

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import time
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from sklearn.ensemble import RandomForestClassifier
from nltk.stem import PorterStemmer
import nltk
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
data= pd.read_csv("C:/Users/kamya/Downloads/archive (3)/Youtube Video Dataset.csv", header=0)

data.head()
```

]:

	Title	Videourl	Category	Description
0	Madagascar Street Food!!! Super RARE Malagasy ...	/watch?v=EwBA1fOQ96c	Food	GIANT ALIEN SNAIL IN JAPAN! » https://youtu.b...
1	42 Foods You Need To Eat Before You Die	/watch?v=0SPwwpruGIA	Food	This is the ultimate must-try food bucket list...
2	Gordon Ramsay's Top 5 Indian Dishes	/watch?v=upfu5nQB2ks	Food	We found 5 of the best and most interesting In...
3	How To Use Chopsticks - In About A Minute	/watch?v=xFRzzSF_6gk	Food	You're most likely sitting in a restaurant wit...
4	Trying Indian Food 1st Time!	/watch?v=K79bXtaRwcM	Food	HELP SUPPORT SINSTV!! Shop Our Sponsors!\nLast...

```
print(data.shape)
print(data.isnull().values.any())
```

```
(11211, 4)
True
```

The dataset contains 11211 rows and 4 columns.

Data file's information:

- Youtube(Title,Videourl,Category,Description)
- Title-Name of youtube video
- Videourl-Video Url
- desciption-Information about video
- category-There are 6 Category
  - Art and Music
  - Manufacturing
  - Food
  - History
  - Travelblog
  - Science And Technology

Activity 2: Data Preparation

As we have understood how the data is, let us preprocess the collected data.

The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

• Removing Redundant Columns

• Handling Missing Values

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling Missing Values

```
data=data.dropna()
print(data.isnull().values.any())
```

```
False
```

There are no missing values in the dataset. That is why we can skip this step.

Activity 2.2: Checking category-wise distribution of Data

```
Category=data['Category'].value_counts()
print(Category.shape)
print(Category)
```

```
(6,)
travel blog           2200
Science&Technology    2074
Food                  1828
manufacturing         1699
Art&Music             1682
History               1645
Name: Category, dtype: int64
```

This step is important to check whether our data is imbalance or not.
Since the distribution is approximately even in all categories, this concludes that our data is balanced.

Activity 2.3: Loading stop words from nltk library

This step includes the following pre-processings:-

- Remove url
- Remove email address
- Remove special character
- Word Stemming

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))
#Stemmering the word
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        #Removing link
        url_pattern = r'((http|ftp|https):\/\/)?[\w\-_]+(\.[\w\-_]+)+([\w\-\.,@?^=%&amp;:/~\+#]*[\w\-\@?^=%&amp;/~\+#])?'
        total_text = re.sub(url_pattern, ' ', total_text)
        #removing email address
        email_pattern =r'[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+'
        total_text = re.sub(email_pattern, ' ', total_text)
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then retain that word from the data
            if not word in stop_words:
                word=(sno.stem(word))
                string += word + " "

        data[column][index] = string
```

The code block preprocesses textual data for natural language processing by performing various transformations. It removes URLs, email addresses, and special characters while converting all text to lowercase. It then removes stop words, which are commonly used words in a language, and applies stemming to the remaining words. Stemming reduces words to their base or root form, which can help improve the accuracy of text classification models. The resulting preprocessed data is more suitable for machine learning algorithms to work with and can lead to better model performance.

Activity 2.4: Preprocessing of Description

```
#text processing stage.
start_time = time.process_time()
for index, row in data.iterrows():
    if type(row['Description']) is str:
        preprocessing(row['Description'], index, 'Description')
print('Time took for preprocessing the text :',time.process_time() - start_time, "seconds")
```

```
Time took for preprocessing the text : 65.53125 seconds
```

This code performs text preprocessing on the 'Description' column of a pandas dataframe 'data'. It uses the 'preprocessing' function to perform the following tasks: removing links and email addresses, replacing special characters with spaces, converting all characters to lowercase, removing stop words, and stemming the remaining words using the SnowballStemmer. The code iterates over each row in the dataframe, and if the 'Description' value for that row is a string, it applies the 'preprocessing' function to that string and replaces the original 'Description' value with the preprocessed string. The code measures the time taken to complete the preprocessing using the time.process_time() function.

Activity 2.5: Viewing Data after pre-processing

```
data.head(10)
```

	Title	Videourl	Category	Description
0	Madagascar Street Food!!! Super RARE Malagasy ...	/watch?v=EwBA1fOQ96c	Food	giant alien snail japan go tour madagascar get...
1	42 Foods You Need To Eat Before You Die	/watch?v=0SPwwpruGIA	Food	ultim must tri food bucket list burger dip che...
2	Gordon Ramsay's Top 5 Indian Dishes	/watch?v=upfu5nQB2ks	Food	found 5 best interest indian recip channel inc...
3	How To Use Chopsticks - In About A Minute	/watch?v=xFRzzSF_6gk	Food	like sit restaur set chopstick hand say video ...
4	Trying Indian Food 1st Time!	/watch?v=K79bXtaRwcM	Food	help support sinstv shop sponsor last longer b...
5	Blippi Tours the Chocolate Factory | Learn abo...	/watch?v=uSlb-Wbyx6Y	Food	blippi eat veget blippi take tour chocol facto...
6	EGYPT: Vegetarian food | Mobile Sim | Indian S...	/watch?v=Gozaqmg6hmk	Food	video see hunt best mobil network egypt base f...
7	Chinese Street Food Liuhe Tourist Night Market	/watch?v=H0xKYgUX3zI	Food	tri mani differ kind chines street food liuh t...
8	India's Biggest food FESTIVAL | food truck fes...	/watch?v=NpOVNb1keoc	Food	alright guy hope like video aim find somewhat ...
9	Street Food in Madagascar's Biggest City!!! Ze...	/watch?v=OXHHNBVt0pw	Food	villag food madagascar go tour madagascar get ...

The Description column is completely cleaned, with all the Upper case converted to Lower- case, all the emojis, url removed and after stemming.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
In [43]: data_train.describe()
```

Out[43]:

	views	likes	dislikes	comment	published	duration	category
count	1.463600e+04	14636.000000	14636.000000	14636.000000	14636.000000	14636.000000	14636.000000
mean	7.107934e+05	2784.093946	254.150724	409.035597	9.437346	622.944999	4.607065
std	2.731062e+06	8936.295816	1029.257991	1511.180179	1.770455	710.910074	1.576242
min	4.900000e+01	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	3.151425e+04	88.000000	7.000000	7.000000	8.000000	183.000000	4.000000
50%	1.586610e+05	450.000000	38.000000	46.000000	10.000000	321.000000	4.000000
75%	5.829575e+05	1861.500000	166.250000	224.000000	11.000000	719.000000	6.000000
max	1.380479e+08	283824.000000	49449.000000	75045.000000	12.000000	3077.000000	8.000000

Activity 2: Visualisation

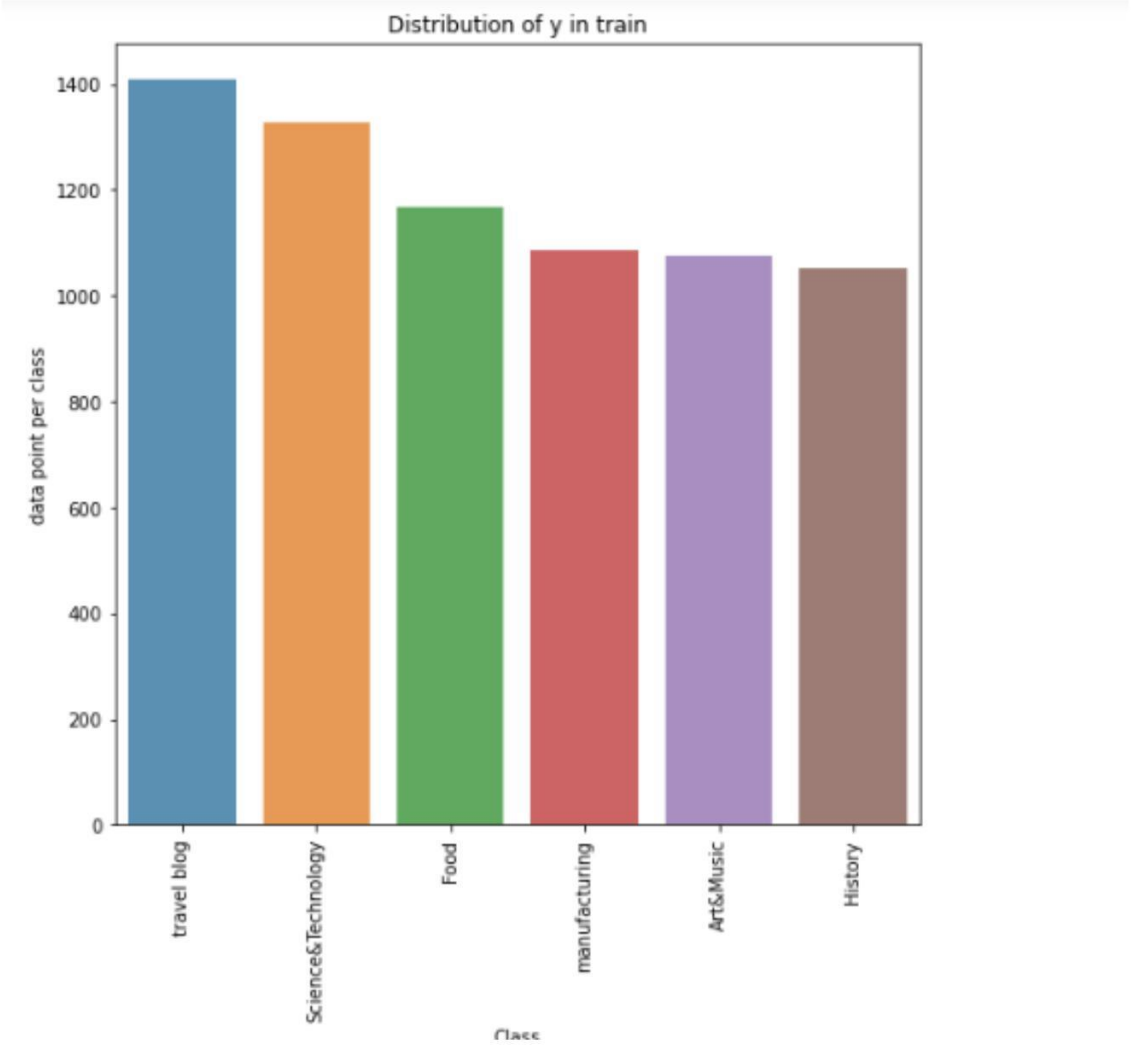
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Visualizing the distribution of target variable in the training dataset

```
X_trainCategory=train_df['Category'].value_counts()
print(X_trainCategory)
print('Distribution of y in train')
plt.figure(figsize=(8,8))
sns.barplot(X_trainCategory.index, X_trainCategory.values, alpha=0.8)
plt.title('Distribution of y in train')
plt.ylabel('data point per class')
plt.xlabel('Class')
plt.xticks(rotation=90)
plt.show()
```

The code is used to visualize the distribution of the output variable 'Category' in the training dataset. The 'value_counts()' function is used to count the number of data points available for each category in the training set. Then, a bar plot is created using the 'sns.barplot' function from the seaborn library, which shows the number of data points available for each category in the training set. This visualization helps in understanding the class imbalance problem, if any, and can help in deciding the appropriate evaluation metrics for the classification model.

```
travel blog           1408
Science&Technology    1327
Food                  1169
manufacturing         1087
Art&Music             1077
History               1053
Name: Category, dtype: int64
Distribution of y in train
```



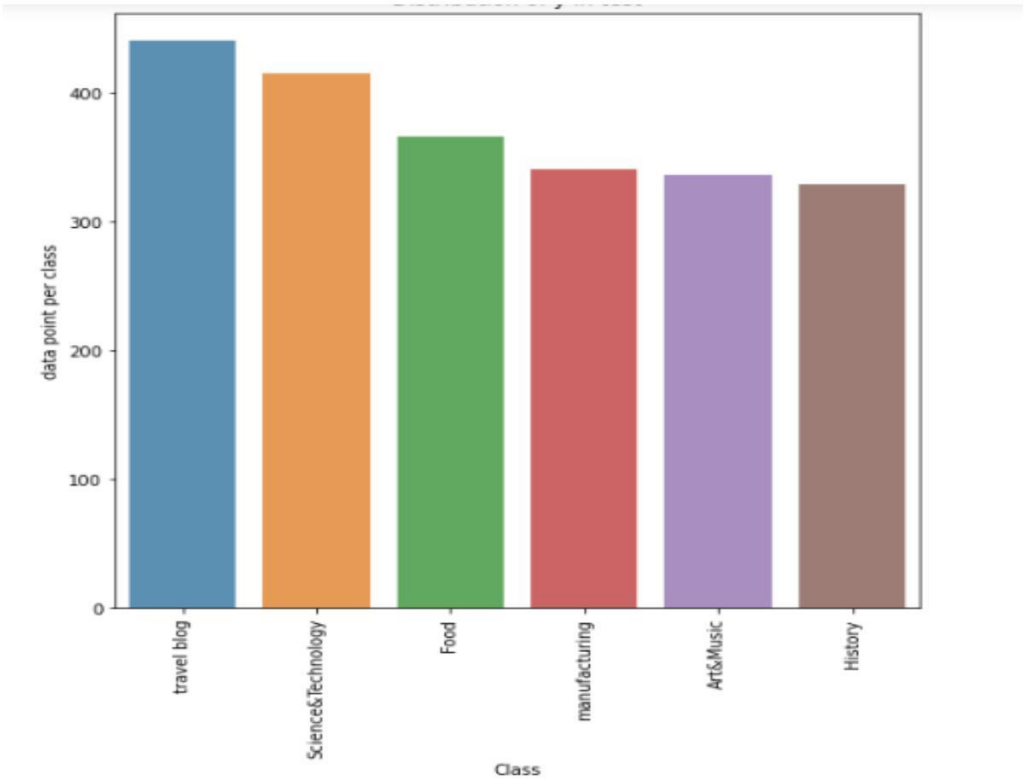
Distribution of y in Train

Activity 2.2:Visualizing the distribution of target variable in the test dataset

```
test_dfCategory=test_df['Category'].value_counts()
print(test_dfCategory)
print('Distribution of y in test')
plt.figure(figsize=(8,8))
sns.barplot(test_dfCategory.index,test_dfCategory.values, alpha=0.8)
plt.title('Distribution of y in test')
plt.ylabel('data point per class')
plt.xlabel('Class')
plt.xticks(rotation=90)
plt.show()
```

This code calculates the frequency of each class in the 'Category' column of the test_df dataframe and then plots a bar chart to visualize the distribution of data points across different classes. The bar chart is labeled with appropriate titles, axes labels, and rotations for the x-axis labels. The purpose of this code is to ensure that the test dataset has a similar distribution of data points across classes as the training dataset, which is important for evaluating the performance of the model on unseen data.

```
travel blog           440
Science&Technology    415
Food                  366
manufacturing         340
Art&Music             336
History               329
Name: Category, dtype: int64
Distribution of y in test
```



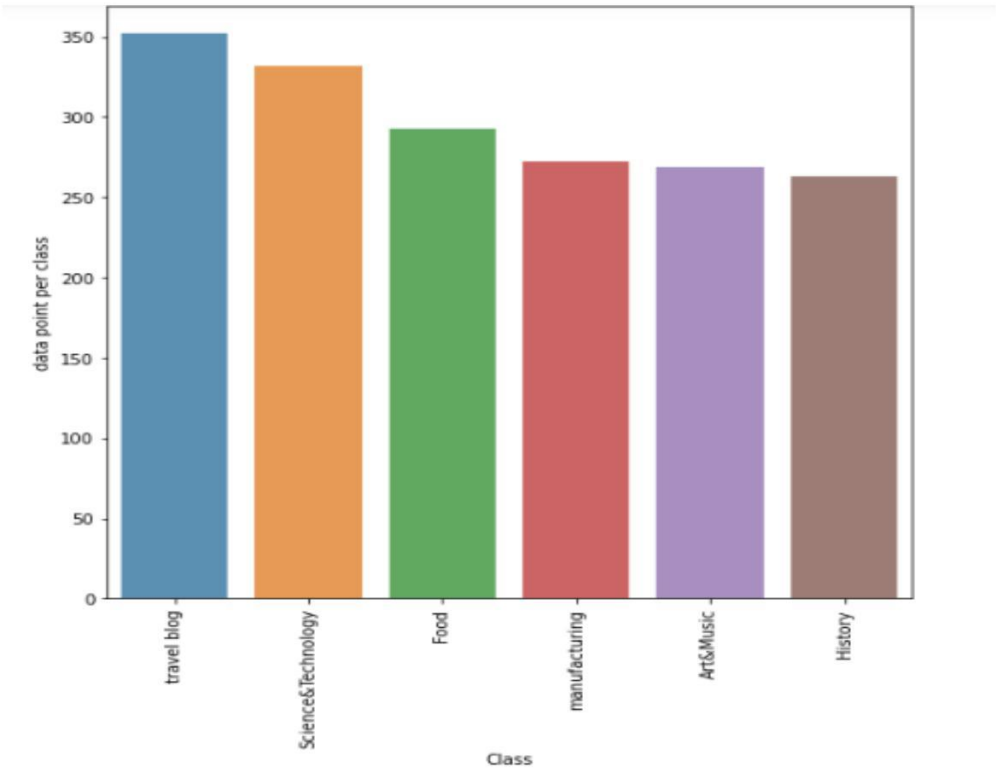
Distribution of y in Test

Activity 2.3:Visualizing the distribution of target variable in the cross-validation dataset

```
cv_dfCategory=cv_df['Category'].value_counts()
print(cv_dfCategory)
print('Distribution of y in cv')
plt.figure(figsize=(8,8))
sns.barplot(cv_dfCategory.index,cv_dfCategory.values, alpha=0.8)
plt.title('Distribution of y in cv')
plt.ylabel('data point per class')
plt.xlabel('Class')
plt.xticks(rotation=90)
plt.show()
```

This code calculates the count of each category in the cross-validation dataset and displays a bar plot representing the distribution of categories in the dataset. The y-axis shows the number of data points per class and the x-axis represents each class. The "sns.barplot()" function from the Seaborn library is used to create the bar plot.

```
travel blog           352
Science&Technology    332
Food                  293
manufacturing         272
Art&Music             269
History               263
Name: Category, dtype: int64
Distribution of y in cv
```



Distribution of y in CV

Activity 2.4: Split Data into Test, Train and Cross Validation set

```
y_true = data['Category'].values
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.2)
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

The code is splitting the dataset into training, cross-validation, and test sets for model evaluation. The true output values (y_true) are obtained from the 'Category' column of the dataset. Stratified sampling is used to ensure that the distribution of output variables is maintained in all three sets. The test set is 20% of the entire dataset, and the training set is further split into training and cross-validation sets in the ratio of 80:20. This splitting is important to avoid overfitting and to evaluate the model's performance on unseen data. The resulting sets are stored in respective variables for further use in model building and evaluation.

Milestone 4: Model Building

Activity 1: Text vectorization[BOW,TF-IDF]

Text vectorization is the process of converting text data into a numerical form that can be used by machine learning algorithms. Two popular techniques for text vectorization are Bag-of-Words (BOW) and Term Frequency-Inverse Document Frequency (TF-IDF). BOW creates a matrix that represents the frequency of each word in the document. It ignores the order of the words and treats each word independently. TF-IDF takes into account both the frequency of the word in the document and the inverse frequency of the word in the corpus. It downweights common words and upweights rare words. Both techniques have their strengths and weaknesses, and the choice between them depends on the specific task and the characteristics of the data.

Activity 1.1: Bag of Words (BoW)

```
x_tr=train_df['Description']
x_test=test_df['Description']
x_cv=cv_df['Description']
```

```
bow = CountVectorizer()
x_tr_uni = bow.fit_transform(x_tr)
x_test_uni= bow.transform(x_test)
x_cv_uni= bow.transform(x_cv)
```

This code segment uses the CountVectorizer method from scikit-learn to perform bag of words (BOW) text vectorization. It transforms the raw text data into numerical feature vectors by counting the frequency of each word in the text. The method is applied to the training, test, and cross-validation sets of the data separately. The resulting vectors are stored in the variables x_tr_uni, x_test_uni, and x_cv_uni, respectively.

Activity 1.2: TF-IDF

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
x_tr_tfidf = tf_idf_vect.fit_transform(x_tr)
x_test_tfidf = tf_idf_vect.transform(x_test)
x_cv_tfidf = tf_idf_vect.transform(x_cv)
```

This code initializes a TfidfVectorizer and uses it to convert the raw text data into numerical features. The vectorizer creates a vocabulary of words and generates a sparse matrix of TF-IDF (Term Frequency-Inverse Document Frequency) values for each document. It uses a range of n-grams (1-grams and 2-grams in this case) to capture more contextual information. The resulting matrices x_tr_tfidf, x_test_tfidf, and x_cv_tfidf represent the TF-IDF weighted word frequencies of the training, test, and cross-validation data, respectively. These matrices will be used as input to train and test the machine learning models.

Activity 2: Function to plot Precision-Recall Matrix for Multi-Class Classification

```
def plotPrecisionRecall(y_test,y_pred):
    C = confusion_matrix(y_test, y_pred)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    labels = ['Art&Music','Food','History','Sci&Tech','Manu','TravelBlog']


    print("-"*20, "Precision matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True,annot_kws={"size": 16}, fmt='g', xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True,annot_kws={"size": 16}, fmt='g', xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

The plotPrecisionRecall() function is used to plot the precision and recall matrices of a given model's performance on a test set. It takes the actual y_test and predicted y_pred as input, and computes the confusion matrix C, as well as the precision matrix B and recall matrix A, using NumPy operations. It then creates two heatmaps using Seaborn library, to visualize B and A matrices respectively, with each cell representing the precision or recall value for a given combination of true and predicted class. The resulting plots can be used to evaluate the model's performance in terms of precision and recall for each class, and to identify any patterns of misclassification.

Activity 3: Training and testing the models using multiple algorithms

Now that we have clean data, a function for evaluation it is time to build models to train the data. For this project we will be using 2 different multi-class classification algorithms to build our models. The best model will be used for prediction.

Activity 3.1: Linear SVM

Support Vector Machines (SVM) is a popular machine learning algorithm used for both classification and regression tasks. Linear SVM is a variant of SVM that uses a linear kernel function to transform the input data into a higher dimensional space, where a linear decision boundary can be used to separate the different classes.

SGDClassifier is a popular linear classifier in machine learning that uses stochastic gradient descent (SGD) to optimize the loss function. It is well-suited for handling large-scale and sparse datasets, and can be used for both binary and multiclass classification problems. SGDClassifier allows for tuning of hyperparameters such as the learning rate, regularization, and loss function, making it a flexible and powerful tool for classification tasks.

Activity 3.1.1: Unigram(BOW)

```
clf = SGDClassifier(loss = 'hinge', alpha = 0.01, class_weight='balanced', learning_rate='optimal',eta0=0.001, n_jobs = -1)
clf.fit(x_tr_uni,y_train)
y_pred = clf.predict(x_test_uni)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred,average='macro')))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred,average='macro')))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='macro')))

print("-"*20, "confusion matrix", "-"*20)
plt.figure(figsize=(12,8))
matrix=confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(matrix)
sns.set(font_scale=1.4)#for label size
labels = ['Art&Music','Food','History','Sci&Tech','Manu','TravelBlog']
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
plotPrecisionRecall(y_test,y_pred)
```

This code is implementing a binary classification task using the SGDClassifier in scikit-learn library. The classifier is trained on the training set (x_tr_uni,y_train) and tested on the test set (x_test_uni, y_test). The hyperparameters for the SGDClassifier are set as follows:

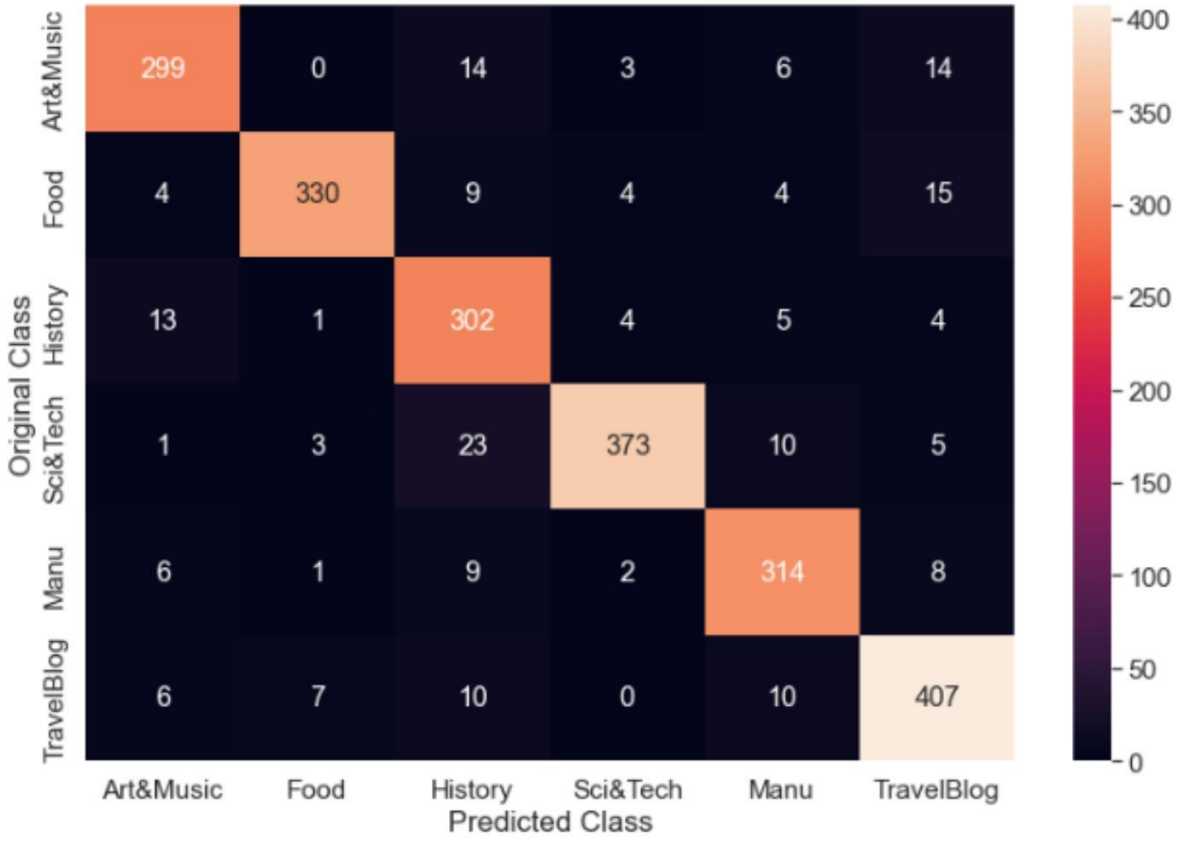
- loss='hinge': the hinge loss is used as the loss function
- alpha=0.01: the regularization parameter for L2 regularization is set to 0.01
- class_weight='balanced': class weights are set to 'balanced' to account for class imbalance in the training set
- learning_rate='optimal': the learning rate is set to 'optimal' which adapts the learning rate based on the data
- eta0=0.001: the initial learning rate for the 'constant' learning rate schedule is set to 0.001
- n_jobs=-1: the number of CPU cores used for parallel processing is set to -1, which means all available cores are used.

After training the classifier, the accuracy, precision, recall, and F1-Score are computed using scikit-learn's accuracy_score, precision_score, recall_score, and f1_score functions.

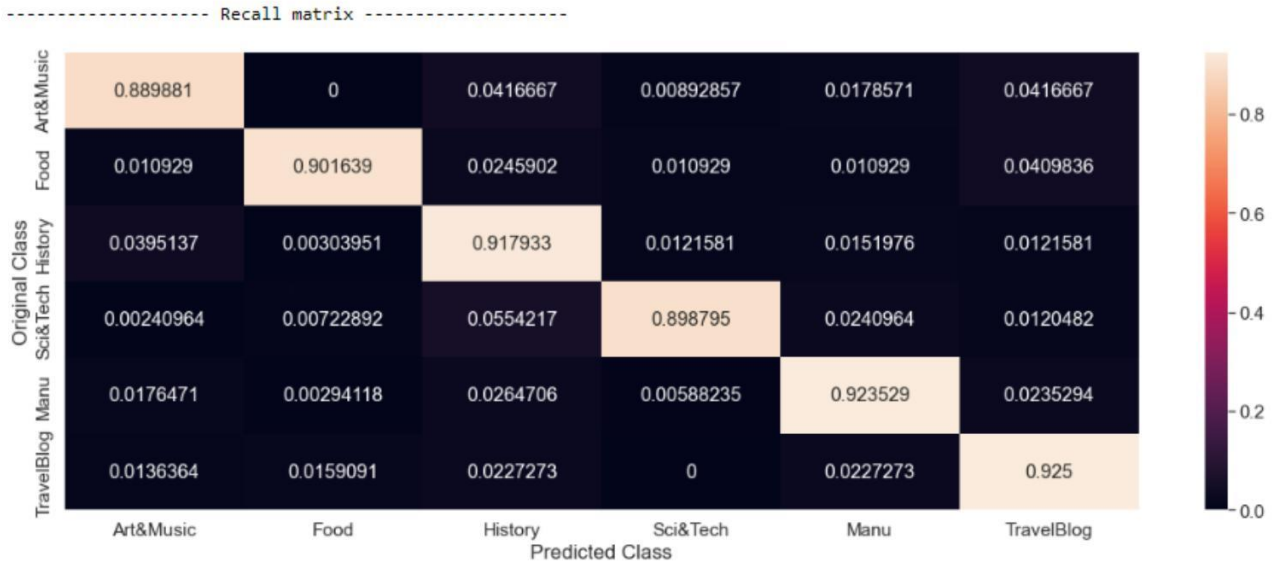
The confusion matrix is then plotted using the seaborn library to visualize the performance of the classifier on each class. Finally, a precision-recall curve is plotted using the plotPrecisionRecall function, which shows the trade-off between precision and recall for different classification thresholds.

```
Accuracy on test set: 90.970%
Precision on test set: 0.910
Recall on test set: 0.909
F1-Score on test set: 0.909
```

```
-------------------- confusion matrix --------------------
```




```
-------------------- Precision matrix --------------------
```



Conclusion

- Travel Blog-407 are correctly classified out of 440 Precision of 91.4% and Recall of 88.981
- Science and Technology-377 are correctly classified out of 415 Precision of 96.2% and Recall of 90.9%
- Food-333 are correctly classified out of 366 Precision of 82.4 % and Recall of 89.9%
- Manufacturing-310 are correctly classified out of 340 Precision of 95.9% and Recall of 90.8%
- Art & Music-229 are correctly classified out of 336 Precision of 88.0% and Recall of 91.1%
- History-296 are correctly classified out of 329 Precision of 90.6% and Recall of 92.5%

Activity 3.1.2: TF-IDF

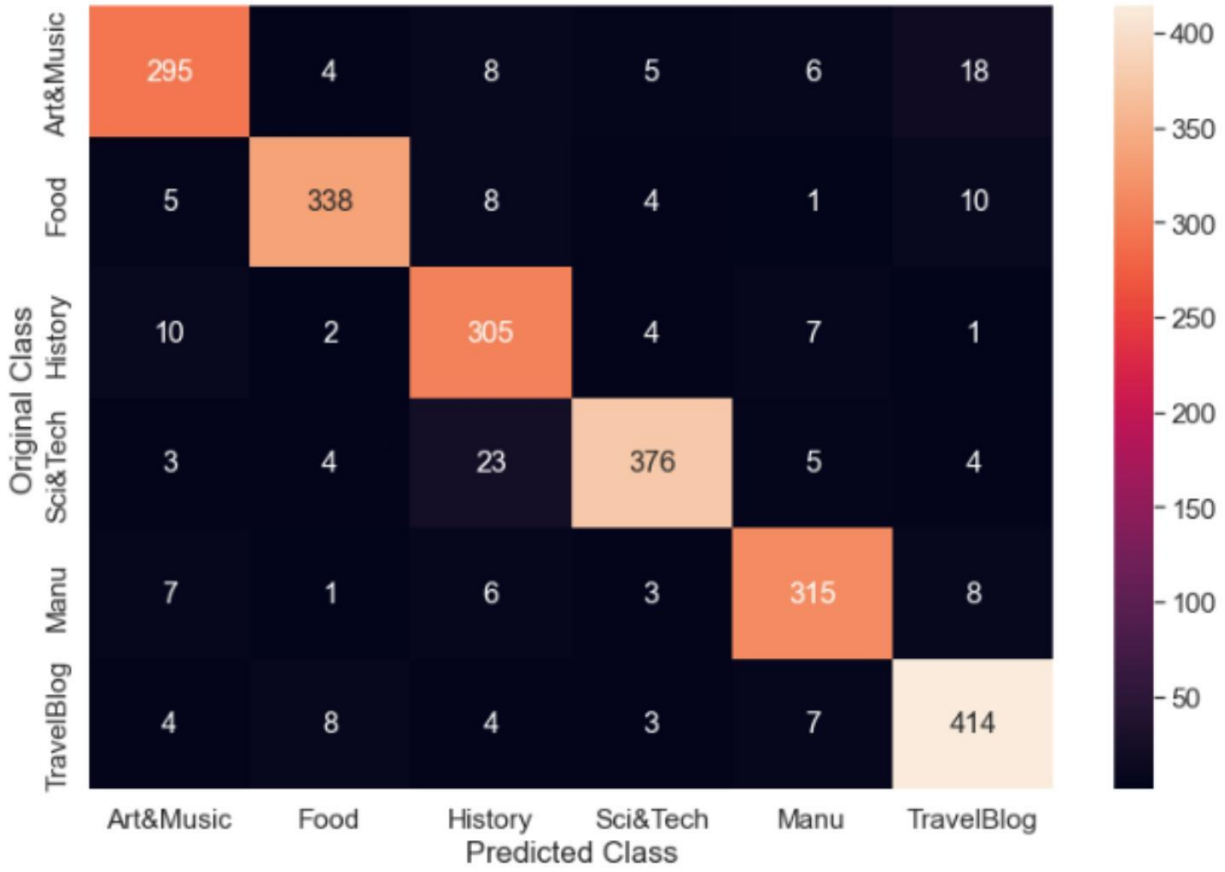
```
clf = SGDClassifier(loss = 'hinge', alpha =0.0001, class_weight='balanced', learning_rate='optimal',eta0=0.001, n_jobs = -1)
clf.fit(x_tr_tfidf,y_train)
y_pred = clf.predict(x_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred,average='macro')))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred,average='macro')))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='macro')))

print("-"*20, "confusion matrix", "-"*20)
plt.figure(figsize=(12,8))
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(6),range(6))
sns.set(font_scale=1.4)#for label size
labels = ['Art&Music','Food','History','Sci&Tech','Manu','TravelBlog']
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
plotPrecisionRecall(y_test,y_pred)
```

This code is implementing a multiclass classification task using the SGDClassifier in scikit-learn library. The classifier is trained on the training set (x_tr_tfidf, y_train) and tested on the test set (x_test_tfidf, y_test). The hyperparameters for the SGDClassifier are set as follows:

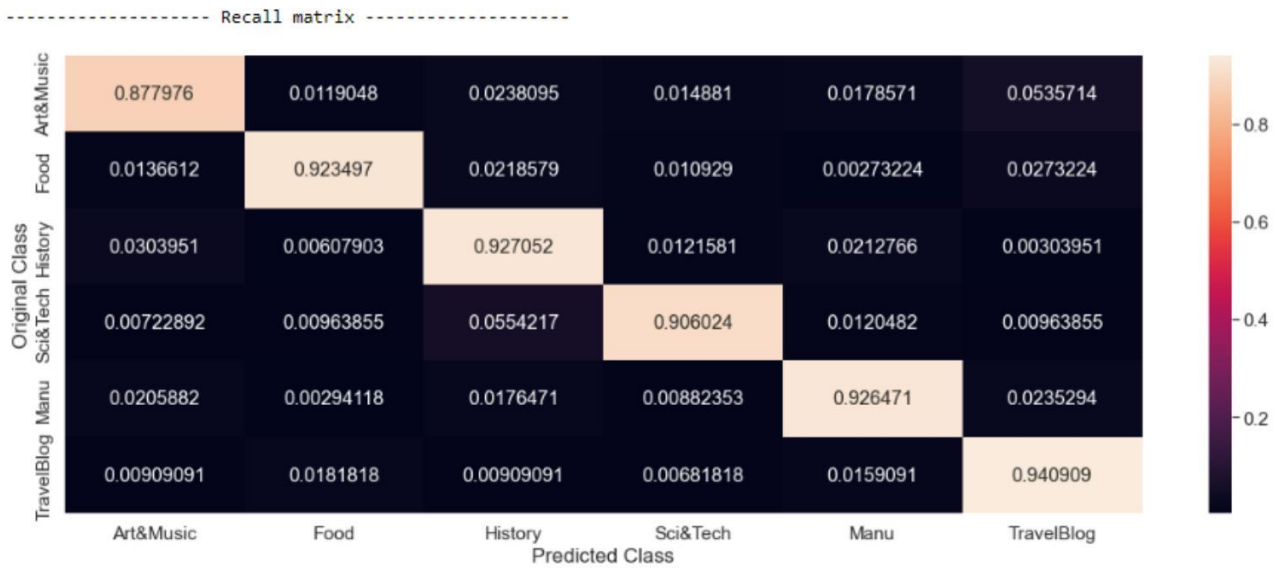
loss='hinge': the hinge loss is used as the loss function
alpha=0.0001: the regularization parameter for L2 regularization is set to 0.0001
class_weight='balanced': class weights are set to 'balanced' to account for class imbalance in the training set
learning_rate='optimal': the learning rate is set to 'optimal' which adapts the learning rate based on the data
eta0=0.001: the initial learning rate for the 'constant' learning rate schedule is set to 0.001
n_jobs=-1: the number of CPU cores used for parallel processing is set to -1, which means all available cores are used.
After training the classifier, the accuracy, precision, recall, and F1-Score are computed using scikit-learn's accuracy_score, precision_score, recall_score, and f1_score functions. The confusion matrix is then plotted using the seaborn library to visualize the performance of the classifier on each class. Finally, a precision-recall curve is plotted using the plotPrecisionRecall function, which shows the trade-off between precision and recall for different classification thresholds.

```
Accuracy on test set: 91.779%
Precision on test set: 0.917
Recall on test set: 0.917
F1-Score on test set: 0.917
```

```
-------------------- confusion matrix --------------------
```




```
-------------------- Precision matrix --------------------
```



Conclusion

- Travel Blog-416 are correctly classified out of 440 Precision of 90.9% and Recall of 89.2%
- Science and Technology-384 are correctly classified out of 415 Precision of 96.3% and Recall of 92.8%
- Food-340 are correctly classified out of 366 Precision of 86.6 % and Recall of 90.8%
- Manufacturing-312 are correctly classified out of 340 Precision of 95.5% and Recall of 92.5%
- Art & Music-300 are correctly classified out of 336 Precision of 90.9% and Recall of 91.7%
- History-299 are correctly classified out of 329 Precision of 91.8% and Recall of 94.5%

Activity 3.1.3: LinearSVM Conclusion

Model	hyper parameter	F1-score cv	F1-score test	Precision test	Recall test	Accuracy Test
unigram	0.01	0.927	0.907	0.908	0.907	90.836%
TF-IDF	0.0001	0.922	0.920	0.920	0.920	92.138%

Activity 3.2: Bagging(Random Forest)

Bagging, also known as Bootstrap Aggregating, is an ensemble learning method that combines multiple models to improve the accuracy and stability of predictions. Random Forest is a type of bagging method that uses decision trees as base estimators. It creates a large number of decision trees by randomly selecting subsets of features and training each tree on a bootstrap sample of the training data. The final prediction is obtained by aggregating the predictions of all trees. Random Forest is a popular machine learning algorithm due to its high accuracy and ability to handle high-dimensional data.

Activity 3.2.1: Splitting data into train,test

```
y_true = data['Category'].values
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.2)
```

This code snippet splits the data into training and testing sets while ensuring that the distribution of the output variable 'Category' is maintained in both sets. The 'stratify=y_true' parameter in the train_test_split function ensures that each class in the output variable is proportionally represented in both the training and testing sets. The testing set is 20% of the original data.

Activity 3.2.2: BOW,TF-IDF

```
x_tr=X_train['Description']
x_test=test_df['Description']
```

BOW

```
bow = CountVectorizer()
x_tr_uni = bow.fit_transform(x_tr)
x_test_uni= bow.transform(x_test)
```

The code snippet uses the CountVectorizer() function from scikit-learn to convert the text data in 'x_tr' and 'x_test' into a matrix of word counts, also known as a Bag-of-Words representation. The 'fit_transform' method fits the vectorizer to the training data and transforms it into a sparse matrix of word counts, while the 'transform' method only transforms the test data using the same vocabulary learned from the training data. The resulting 'x_tr_uni' and 'x_test_uni' matrices can be used as input to train and evaluate machine learning models.

TF-IDF

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
x_tr_tfidf = tf_idf_vect.fit_transform(x_tr)
x_test_tfidf = tf_idf_vect.transform(x_test)
```

The code snippet uses the TfidfVectorizer() function from scikit-learn to convert the text data in 'x_tr' and 'x_test' into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features. The 'ngram_range=(1,2)' parameter specifies that the vectorizer should consider both unigrams and bigrams as features. The 'fit_transform' method fits the vectorizer to the training data and transforms it into a sparse matrix of TF-IDF features, while the 'transform' method only transforms the test data using the same vocabulary

learned from the training data. The resulting 'x_tr_tfidf' and 'x_test_tfidf' matrices can be used as input to train and evaluate machine learning models.

Activity 3.2.3: Unigram(BOW)

```
RF= RandomForestClassifier(n_estimators=16,max_depth=130)
RF.fit(x_tr_uni,y_train)
y_pred =RF.predict(x_test_uni)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred,average='macro')))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred,average='macro')))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='macro')))
print("-"*20, "confusion matrix", "-"*20)
plt.figure(figsize=(12,8))
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(6),range(6))
sns.set(font_scale=1.4)#for label size
labels = ['Art&Music','Food','History','Sci&Tech','Manu','TravelBlog']
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
plotPrecisionRecall(y_test,y_pred)
```

This code fits a Random Forest classifier with 16 trees and a maximum depth of 130 on the training data transformed using bag-of-words representation. The model is used to predict the categories of the test data and performance metrics such as accuracy, precision, recall and F1-score are printed. Additionally, a confusion matrix and a precision-recall curve are plotted to visualize the performance of the model. The confusion matrix shows the number of samples classified correctly and incorrectly for each class, while the precision-recall curve shows how well the model retrieves the positive class samples compared to the negative class samples at different classification thresholds.

```
Accuracy on test set: 86.478%
Precision on test set: 0.864
Recall on test set: 0.862
F1-Score on test set: 0.863
```

---------- confusion matrix ----------

---------- Precision matrix ----------

Conclusion

- Travel Blog-393 are correctly classified out of 440 Precision of 89.3% and Recall of 89.3%
- Science and Technology-368 are correctly classified out of 415 Precision of 92% and Recall of 88.6%
- Food-324 are correctly classified out of 366 Precision of 93% and Recall of 88.5%
- Manufacturing-298 are correctly classified out of 340 Precision of 78.2% and Recall of 87.6%
- Art & Music-276 are correctly classified out of 336 Precision of 83.8% and Recall of 82.1%
- History-258 are correctly classified out of 329 Precision of 83.2% and Recall of 78.4%

Activity 3.2.4: TF-IDF

```
RF= RandomForestClassifier(n_estimators=20,max_depth=190)
RF.fit(x_tr_tfidf,y_train)
y_pred =RF.predict(x_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred,average='macro')))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred,average='macro')))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred,average='macro')))
print("-"*20, "confusion matrix", "-"*20)
plt.figure(figsize=(12,8))
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(6),range(6))
sns.set(font_scale=1.4)#for label size
labels = ['Art&Music','Food','History','Sci&Tech','Manu','TravelBlog']
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

plotPrecisionRecall(y_test,y_pred)
```

In this code, a Random Forest classifier is trained on TF-IDF vectorized data with 20 decision trees and maximum depth of 190. The trained model is used to predict the category of the test data. The accuracy, precision, recall, and F1-score are computed for the predicted output using the ground truth labels. A confusion matrix is plotted to visualize the performance of the classifier for each category. Finally, a precision-recall curve is plotted to understand the tradeoff between precision and recall for different classification thresholds. This code can be used to evaluate the performance of a Random Forest classifier on text data using TF-IDF vectorization.

```
Accuracy on test set: 84.951%
Precision on test set: 0.851
Recall on test set: 0.846
F1-Score on test set: 0.847
-------------------- confusion matrix --------------------
```

Conclusion

- Travel Blog-396 are correctly classified out of 440 Precision of 80.6% and Recall of 90.0%
- Science and Technology-354 are correctly classified out of 415 Precision of 92.6% and Recall of 85.3%
- Food-315 are correctly classified out of 366 Precision of 95.4% and Recall of 86.0%
- Manufacturing-305 are correctly classified out of 340 Precision of 79.0% and Recall of 89.7%
- Art & Music-265 are correctly classified out of 336 Precision of 81.5% and Recall of 78.8%
- History-259 are correctly classified out of 329 Precision of 83.1% and Recall of 78.7%

Milestone 5 : Performance Testing

Activity 1: Comparing all the Models.

Based on the given metrics, the Linear SVM model seems to be performing the best amongst the two tested models

Support Vector Machine

Model	hyper parameter	F1-score cv	F1-score test	Precision test	Recall test	Accuracy Test
unigram	0.01	0.927	0.907	0.908	0.907	90.836%
TF-IDF	0.0001	0.922	0.920	0.920	0.920	92.138%

Random Forest

Model	n_estimators	Max Depth	F1-score cv	F1-score test	Precision test	Recall test	Accuracy Test
unigram	16	130	0.868	0.858	0.860	0.858	86.119%
TF-IDF	20	190	0.872	0.849	0.854	0.848	85.085%

The results show that all models have achieved good accuracy, with the Random Forest Classifier using BOW achieving the highest accuracy of 86.478%. However, the SGDClassifier using TF-IDF achieved the highest precision, recall, and F1-score of 0.917. This suggests that the SGDClassifier using TF-IDF is better at correctly identifying positive instances (precision), identifying all positive instances (recall), and balancing both metrics (F1-score) than the other models. Overall, the choice of vectorization technique (BOW or TF-IDF) and model (Random Forest or SGDClassifier) should be based on the specific requirements of the problem and the trade-off between accuracy and precision/recall.

Milestone 6: Model Deployment

Activity 1: Save and load the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future**.**

After checking the performance, we decide to save the Linear Support Vector Machine model.

```
import joblib

# Saving the models
joblib.dump(clf, 'model.pkl')
```

```
]: ['model.pkl']
```

```
# Loading the models
clf_loaded = joblib.load('model.pkl')
```

We save the model using the pickle library into a file named model.pkl

Activity 2: Save and load the Vectorizer functions

```
joblib.dump(bow, 'bow_model.pkl')
```

```
]: ['bow_model.pkl']
```

```
joblib.dump(tf_idf_vect, 'tf_idf_model.pkl')
```

```
]: ['tf_idf_model.pkl']
```

```
bow_loaded = joblib.load('bow_model.pkl')
```

```
tf_idf_vect_loaded= joblib.load('tf_idf_model.pkl')
```

This code uses the joblib module to perform serialization and deserialization of a scaler object. The joblib.dump function saves the scaler object to a file called sc.pkl. This file can be used to restore the scaler object later on. The joblib.load function loads the saved scaler object from the sc.pkl file, which is then stored in the scaling variable. This process of saving and loading a machine learning model or preprocessing object using joblib can be useful for reusing the same model or preprocessing pipeline across different projects or on different machines.

Activity 3: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

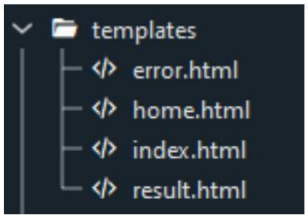
- Building HTML Pages
- Building server-side script
- Run the web application

Activity 3.1: Building HTML pages:

For this project we create two HTML files namely

- Index.html
- Results.html
- Home.html
- Error.html

And we will save them in the templates folder.



Activity 3.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

- Import the necessary Libraries.

The application performs text preprocessing tasks such as removing stop words and stemming using Porter stemming algorithm. It then uses a trained machine learning model to predict the output. Finally, it renders an HTML template to display the results on a web page.

```
from flask import Flask, request, render_template
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import joblib
```

- This Python code loads a pre-trained machine learning model (model.pkl) and two vectorizers (bow_model.pkl and tf_idf_model.pkl) using joblib library. The model is used for prediction and the vectorizers are used to transform the input data into numerical form for the model to make predictions. These files were saved after training the model and vectorizers on a dataset using a separate Python script.

```
# Load the trained model and vectorizer
model = joblib.load('model.pkl')
cv = joblib.load('bow_model.pkl')
tv = joblib.load('tf_idf_model.pkl')
```

- This code creates a new instance of a Flask web application using the Flask class from the Flask library. The __name__ argument specifies the name of the application's module or package.

```
app = Flask(__name__)
```

- This Python code defines a route using the Flask web framework for the home page of a web application. The route ('/') specifies the URL endpoint. The function returns a rendered HTML template named 'home.html' which will be displayed on the home page of the web application.

```
# Define the home page route
@app.route('/')
def home():
    return render_template('home.html')
```

- This Python code defines another route for the web application using the Flask web framework. This route ('/predict') specifies the URL endpoint for a prediction page. When a user navigates to this URL, the function is triggered which renders an HTML template named 'index.html'. This template likely contains a form for users to input data that the machine learning model will use to make a prediction.

```
@app.route('/predict')
def pred():
    return render_template('index.html')
```

- This Python code defines a function for the '/predict' route that accepts POST requests. It retrieves user input from the web form in the request object's 'description' field. It checks if the input is empty or not a string, and returns an error message or an empty string, respectively. If the input is valid, it prints a message to the console indicating its type and content. This function serves to validate the user input before it is processed by the machine learning model.

```
# Define the prediction function
@app.route('/predict', methods=['POST'])
def predict():
    # Get the user input from the form
    description = request.form['description']
    if not description or description.strip() == '':
            # Return an error message if the input is empty
            return render_template('error.html', message='Please enter a description')

    if description is None or not isinstance(description, str):
        print(f"Invalid input: {type(description)}, {description}")
        return ""

    print(f"Valid input: {type(description)}, {description}")
```

- This Python code performs several text preprocessing tasks on the user input 'description' to clean and normalize it before it is processed by the machine learning model. First, it removes any links using a regular expression pattern. Second, it removes any email addresses using another regular expression pattern. Third, it replaces any special characters with a space using a regular expression. Fourth, it replaces any extra spaces with a single space using another regular expression. Fifth, it converts all characters to lower-case. Sixth, it performs stemming on the text using the Porter stemming algorithm from the NLTK library, which reduces each word to its root form. The try-except block ensures that any errors during stemming do not crash the application.

```
#Removing link
url_pattern = r'((http|ftp|https):\/\/)?[\w\-_]+(\.[\w\-_]+)+([\w\-\.,@?^=%&amp;:/~\+#]*[\w\-\@?^=%&a
description = re.sub(url_pattern, ' ', description)
print(f"After removing links: {description}")

#removing email address
email_pattern =r'[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+'
description = re.sub(email_pattern, ' ', description)
print(f"After removing email address: {description}")

# replace every special char with space
description = re.sub('[^a-zA-Z0-9\n]', ' ', description)
print(f"After removing special characters: {description}")

# replace multiple spaces with single space
description = re.sub('\s+',' ', description)
print(f"After removing extra spaces: {description}")

# converting all the chars into lower-case.
description = description.lower()
print(f"After converting to lower-case: {description}")

# stemming the description
try:
    ps = PorterStemmer()
    description = ' '.join([ps.stem(word) for word in description.split() if not word in set(stopwords
    print(f"After stemming: {description}")
except Exception as e:
    print(f"Stemming error: {e}")
    return ""
```

- This Python code vectorizes the preprocessed user input 'description' using the

trained CountVectorizer ('cv') to transform it into a numerical representation that can be fed into the machine learning model. The resulting vector 'X' is passed to the 'predict' method of the trained model to make a prediction on the input text. The predicted category is returned to the user by rendering an HTML template named 'result.html', which likely displays the predicted category in a user-friendly format. This code serves as the final step in the web application's pipeline, taking user input, preprocessing it, and making a prediction before displaying the result to the user.

```
# Vectorize the input using the trained vectorizer
X = cv.transform([description])

# Make the prediction using the trained model
prediction = model.predict(X)[0]

# Return the predicted category to the user
return render_template('result.html', prediction=prediction)
```

Main Function:

This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The if __name__ == '__main__': line checks if the script is the main module being executed, and if so, runs the Flask application using the app.run() method. This method starts the Flask development server, allowing the application to be accessed via a web browser at the appropriate URL.

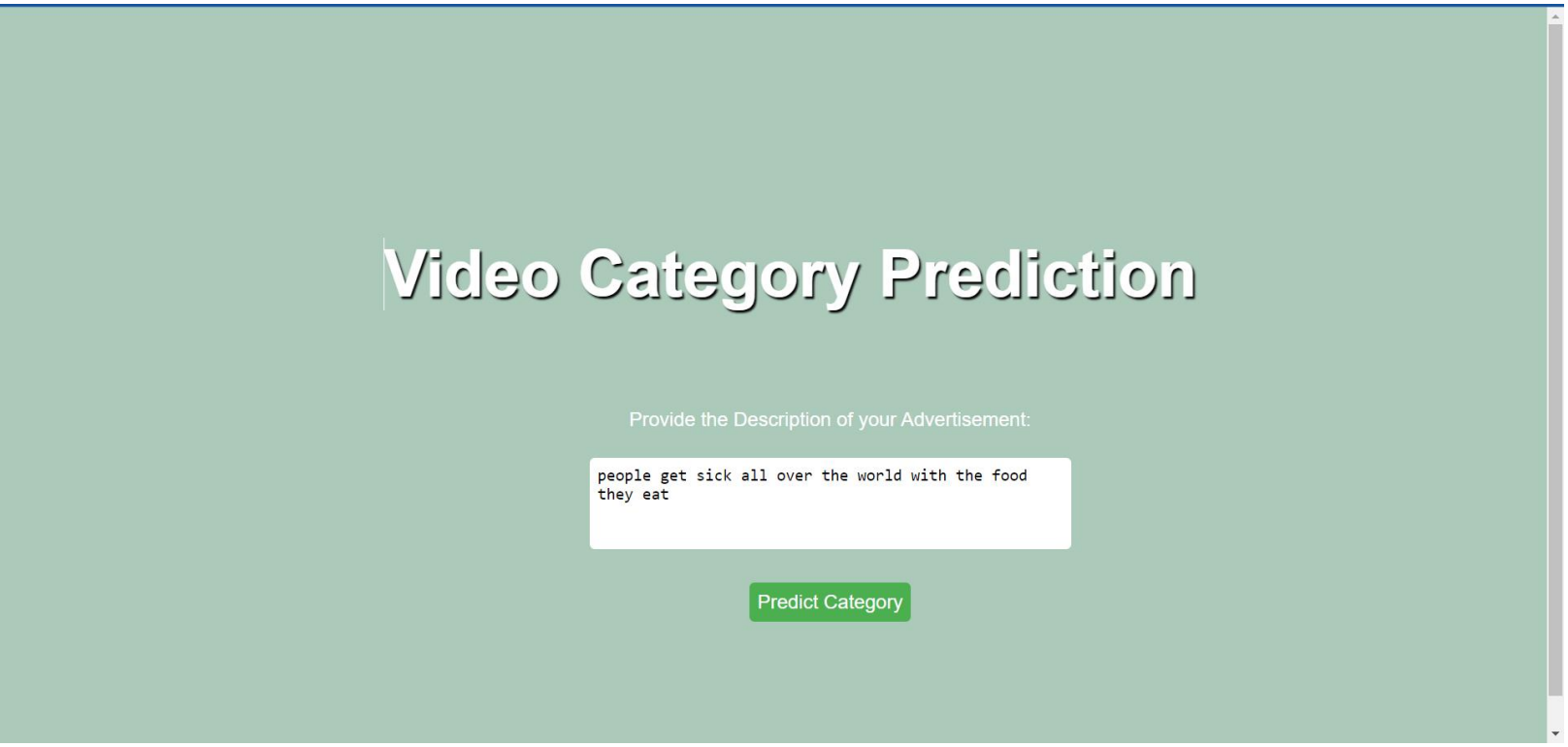
```
if __name__ == '__main__':
    app.run()
```

Activity 3.3: Run the Web Application

Our business name is **'OmniCom'**.

When you run the "app.py" file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.

```
In [1]: runfile('C:/Users/kamya/OneDrive/Desktop/project-2_category/
app.py', wdir='C:/Users/kamya/OneDrive/Desktop/project-2_category')
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a
production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```


Video Category Prediction
Provide the Description of your Advertisement:
people get sick all over the world with the food they eat
Predict Category

Video Category Prediction Result

We think you should go for: **Food**

'Please enter a description'