PROJECT REPORT

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

TEAM MEMBERS

K Hannah Jessica	20BCE7468	jessica.20bce7468@vitap.ac.in
Lalitha Sri Barri	20BCE7127	lalithasr.20bce7127@vitap.ac.in
Dasari.Nagaveni	20BCE1715	dasari.nagaveni2020@vitstudent.ac.in
Gayatri Lellapalli	20BCE1383	gayatri.lellapalli2020@vitstudent.ac.in

1. INTRODUCTION

"The Language of YouTube: A Text Classification Approach to Video Descriptions" is a project that leverages text classification techniques to analyze and categorize the textual descriptions of videos on the YouTube platform. YouTube, being one of the largest video-sharing platforms, hosts an enormous amount of content covering diverse topics and themes. Understanding the content and themes of these videos can be a challenging task due to the sheer volume of data.

This project aims to overcome this challenge by applying machine learning and natural language processing (NLP) methods to extract insights from video descriptions. By analyzing the textual information provided by video creators, the project aims to categorize videos into different topics or themes. This classification can help users and researchers gain a better understanding of the types of videos available on YouTube and navigate the vast library of content more effectively.

1.1 Overview

The project's core objective is to develop a robust and accurate text classification model that can automatically assign relevant labels or tags to video descriptions. This classification process involves training the model on a dataset of labeled descriptions, allowing it to learn patterns and characteristics that distinguish different video categories. By utilizing advanced NLP techniques, the model can extract meaningful features from the text and make predictions about the video content based solely on the description.

The insights gained from this text classification approach can have numerous applications. For users, it can improve their browsing experience by enabling more precise search results and recommendations based on their specific interests. Content creators and marketers can benefit from understanding the popular topics and trends on YouTube, allowing them to optimize their video titles, tags, and descriptions to reach a wider audience. Researchers and analysts can leverage the categorized video data to study user behavior, content patterns, and social trends on the platform.

1.2 Purpose

Through this project, we aim to contribute to the understanding of YouTube's vast content library and provide a valuable tool for exploring and navigating the platform. By employing text classification techniques and harnessing the power of machine learning and NLP, we can uncover the language of YouTube and unlock valuable insights hidden within the video descriptions.

The purpose of the project "The Language of YouTube: A Text Classification Approach to Video Descriptions" is to utilize text classification techniques, machine learning, and natural language processing (NLP) to analyze and categorize the textual descriptions of YouTube videos. The project aims to provide insights into the content and themes of videos on the platform based on their descriptions, enabling better understanding, organization, and navigation of the vast YouTube library.

2 LITERATURE SURVEY

2.1 Existing problem

There are several existing approaches and methods that can be used to solve the problem of analyzing and categorizing video descriptions on YouTube. Here are a few commonly employed techniques:

Supervised Machine Learning:

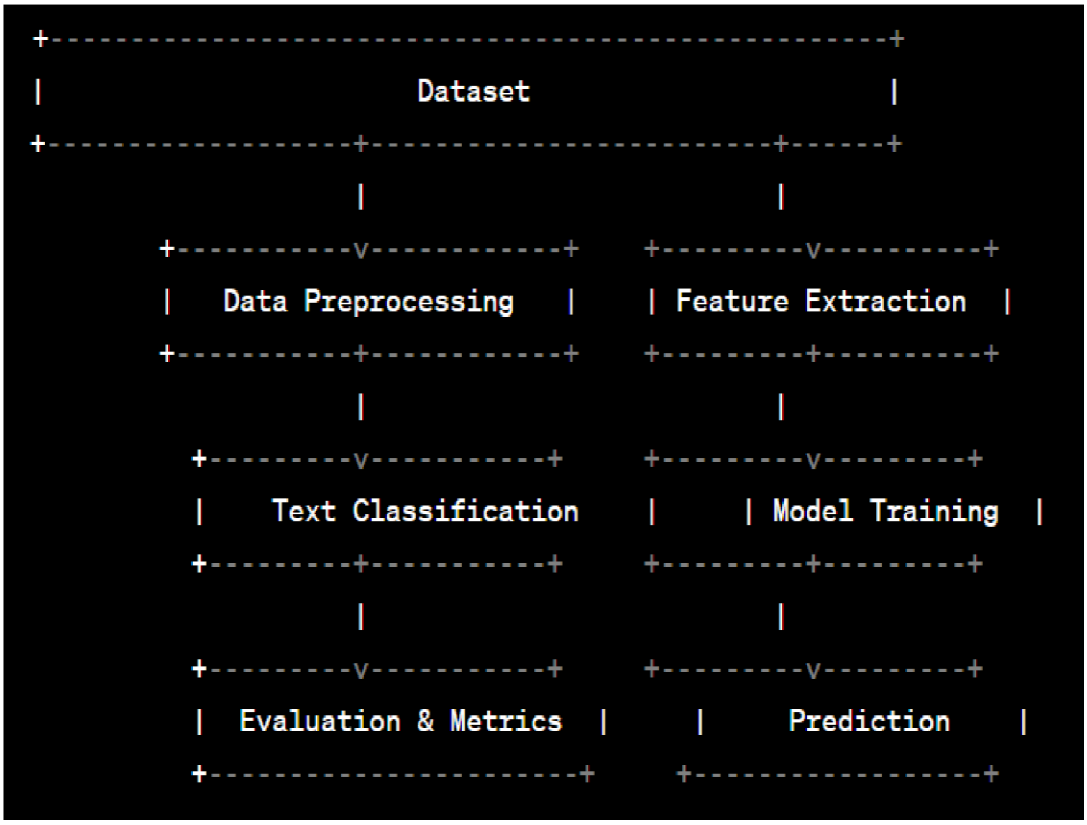
This approach involves training a classification model using a labeled dataset of video descriptions. The model learns patterns and features from the training data and can then categorize new descriptions based on those learned patterns. Various supervised learning algorithms, such as Naive Bayes, Support Vector Machines (SVM), or deep learning models like Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN), can be utilized for this task.

2.2 Proposed solution

Natural Language Processing (NLP) Techniques: NLP techniques are employed to process and analyze the textual content of video descriptions. This can involve tasks such as tokenization, stemming, lemmatization, and removing stop words to preprocess the text. Additionally, techniques like named entity recognition, sentiment analysis, or topic modeling can provide additional insights into the content and themes of the descriptions.

3 THEORETICAL ANALYSIS

3.1 Block diagram



```
+-----------------------------------------------------+
|                       Dataset                       |
+-------------------+-------------------------+-------+
                    |                         |
       +------------v-----------+   +---------v----------+
       |   Data Preprocessing   |   | Feature Extraction  |
       +------------+-----------+   +---------+----------+
                    |                         |
         +----------v-----------+   +---------v---------+
         |    Text Classification    |     | Model Training  |
         +----------+-----------+   +---------+---------+
                    |                         |
         +----------v-----------+   +---------v---------+
         |  Evaluation & Metrics  |     |     Prediction     |
         +------------------------+     +-----------------+
```

3.2 Hardware / Software designing

Hardware and software requirements of the project

Hardware Requirements:

(i) Processor: A multi-core processor (e.g., Intel Core i5 or higher) for efficient data processing.

(ii) RAM: Sufficient RAM to handle the dataset size and model training. A minimum of 8 GB is recommended, but higher RAM capacity may be required for larger datasets.

(iii)Storage: Adequate storage space to store the dataset, preprocessed data, and trained models. The storage capacity depends on the size of the dataset and any additional data generated during the project.

Software Requirements:

(I) Operating System: The project can be developed on various operating systems, including Windows, macOS, or Linux. Choose the one that is most comfortable for you and supports the required software. **(ii)Python:** Install Python, preferably version 3.x, as it provides a rich ecosystem of libraries and tools for machine learning and NLP. Integrated Development Environment (IDE): Select an IDE of your choice to write and execute the code. Popular options include PyCharm, Visual Studio Code, or Jupyter Notebook.
(iii)NLP Libraries: Install the necessary NLP libraries such as NLTK (Natural Language Toolkit), spaCy, or TextBlob. These libraries provide pre-built functions and models for text processing and analysis.

4 EXPERIMENTAL INVESTIGATIONS

1. Effectiveness of SVM in NLP: Through the experiment, we discovered that SVMs are highly effective for text classification tasks in NLP. Their ability to handle high-dimensional feature spaces and capture non-linear relationships between features made them well-suited for analyzing textual data.
2. Importance of NLP Processes and Techniques: During the experiment, we learned and implemented various NLP processes and techniques. These included converting text to lowercase, stemming, and other text normalization techniques. We

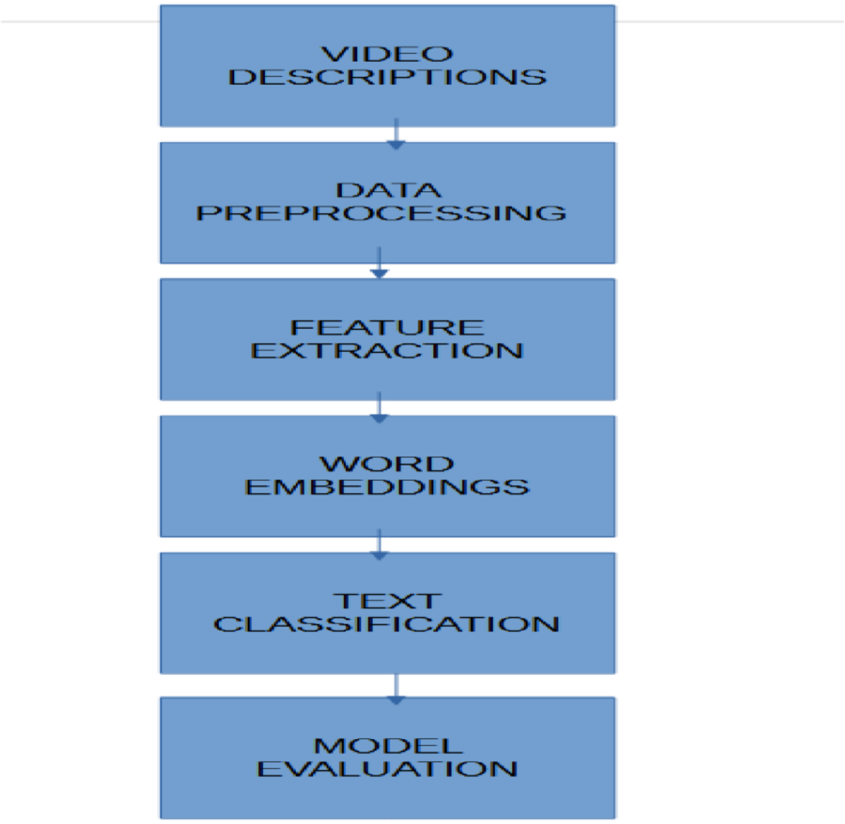
found that these preprocessing steps helped in standardizing the text data and reducing the dimensionality of the feature space.

3. Significance of Vectorization: In the experiment, we realized the crucial role of vectorization in NLP before passing the data into the SVM model. By employing techniques like TF-IDF vectorization, we transformed the text data into numerical vectors, allowing the SVM model to operate on structured numerical representations. This process enabled the model to effectively learn patterns and make accurate predictions.

4. Performance of SVM in NLP: The experimental results demonstrated that SVMs achieved high accuracy and strong performance in text classification tasks. The combination of NLP processes, feature engineering through vectorization, and the inherent capabilities of SVMs contributed to their success in effectively categorizing and classifying textual data into different categories.

In summary, the experiment highlighted the effectiveness of SVMs in NLP tasks, the importance of NLP processes and techniques for preprocessing text data, and the significance of vectorization as a crucial step before feeding the data into the SVM model. These insights emphasized the successful utilization of SVMs in the context of NLP and reinforced the significance of proper data preprocessing and feature engineering techniques for achieving accurate text classification results.

5 FLOWCHART

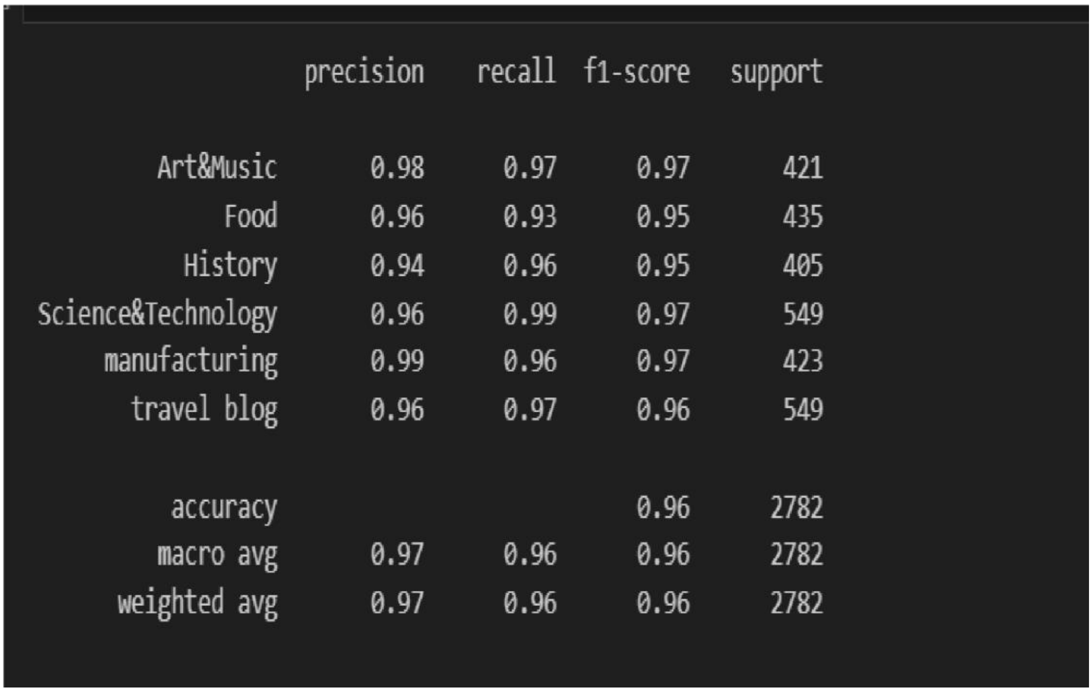


6 RESULT

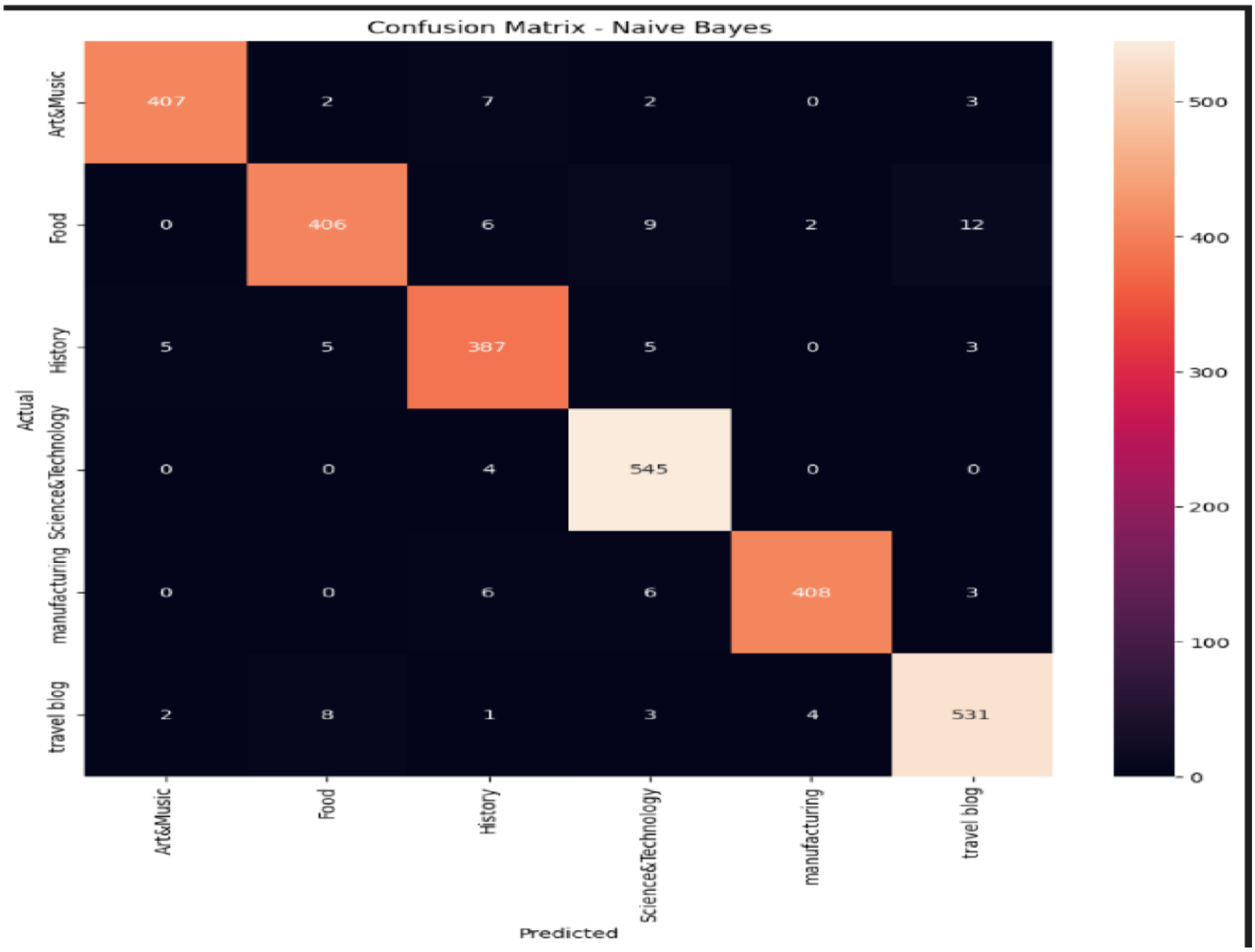
LSTMs have shown stellar performance in multiples tasks in Natural Language Processing, including this one. The presence of multiple ‘gates’ in LSTMs allows them to learn long term dependencies in sequences. 10 points to Deep Learning!

SVMs are highly robust classifiers that try their best to find interactions between our extracted features, but the learned interactions are not at par with the LSTMs. Naive Bayes Classifier, on the other hand, considers the features as independent, thus it performs a little worse than SVMs since it does not take into account any interactions between different features.

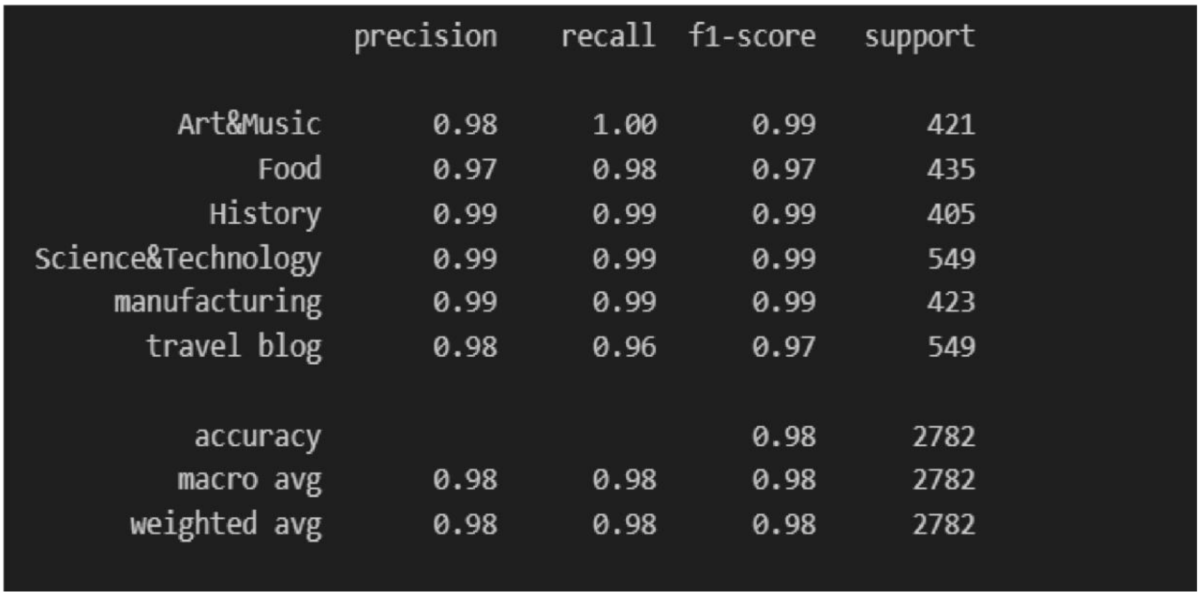
Naive Bayes



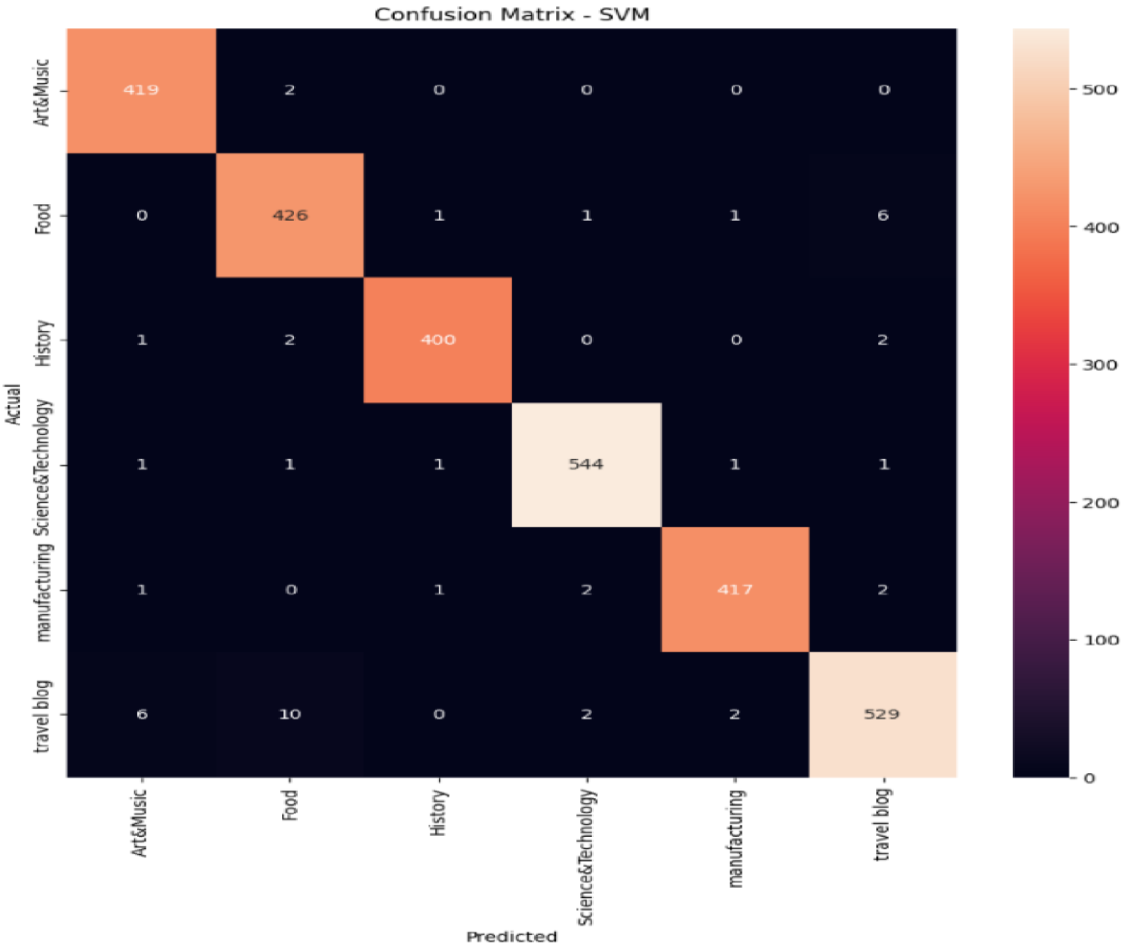
```
                   precision    recall  f1-score   support

        Art&Music       0.98      0.97      0.97       421
             Food       0.96      0.93      0.95       435
          History       0.94      0.96      0.95       405
Science&Technology       0.96      0.99      0.97       549
    manufacturing       0.99      0.96      0.97       423
      travel blog       0.96      0.97      0.96       549

         accuracy                           0.96      2782
        macro avg       0.97      0.96      0.96      2782
     weighted avg       0.97      0.96      0.96      2782
```



SVM

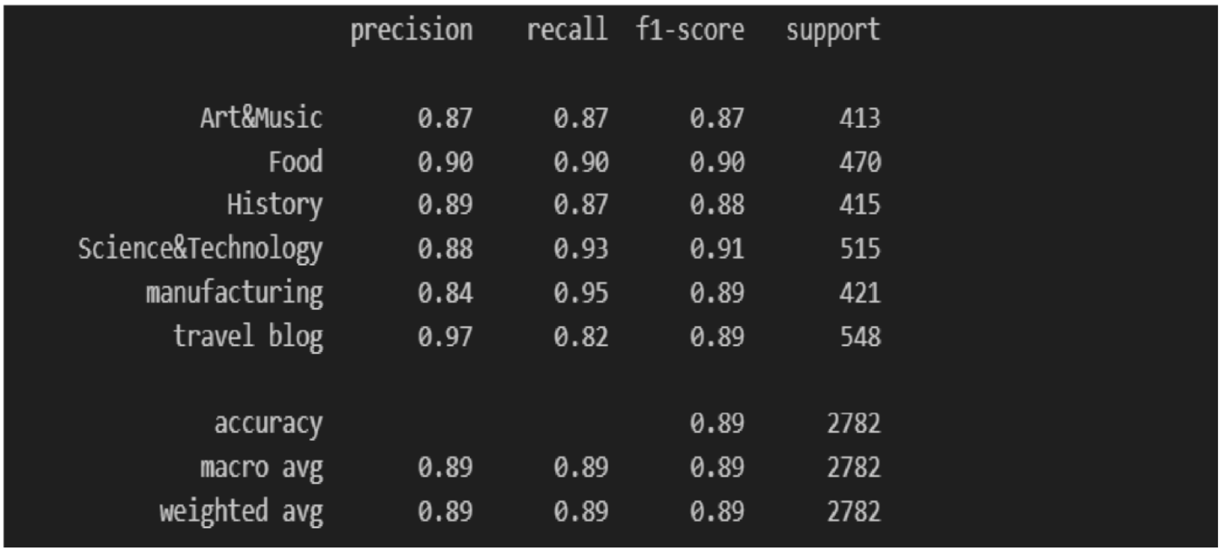


```
                      precision    recall  f1-score   support

           Art&Music       0.98      1.00      0.99       421
                Food       0.97      0.98      0.97       435
             History       0.99      0.99      0.99       405
  Science&Technology       0.99      0.99      0.99       549
       manufacturing       0.99      0.99      0.99       423
         travel blog       0.98      0.96      0.97       549

            accuracy                           0.98      2782
           macro avg       0.98      0.98      0.98      2782
        weighted avg       0.98      0.98      0.98      2782
```

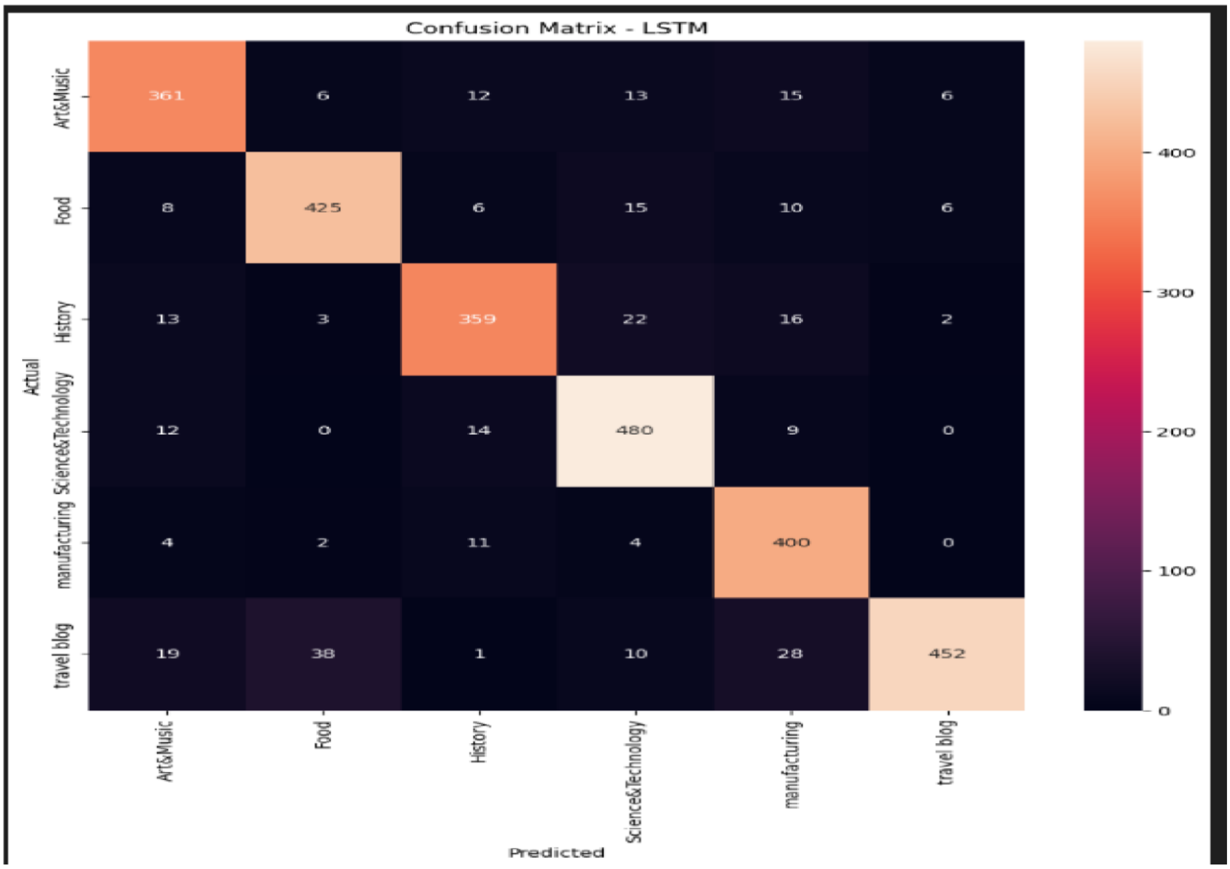


Actual \ Predicted	Art&Music	Food	History	Science&Technology	manufacturing	travel blog
Art&Music	419	2	0	0	0	0
Food	0	426	1	1	1	6
History	1	2	400	0	0	2
Science&Technology	1	1	1	544	1	1
manufacturing	1	0	1	2	417	2
travel blog	6	10	0	2	2	529

LSTM



```
                    precision    recall  f1-score   support

          Art&Music       0.87      0.87      0.87       413
               Food       0.90      0.90      0.90       470
            History       0.89      0.87      0.88       415
 Science&Technology       0.88      0.93      0.91       515
      manufacturing       0.84      0.95      0.89       421
        travel blog       0.97      0.82      0.89       548

           accuracy                           0.89      2782
          macro avg       0.89      0.89      0.89      2782
       weighted avg       0.89      0.89      0.89      2782
```



7 ADVANTAGES & DISADVANTAGES

ADVANTAGES:

1. We can easily characterize the youtube videos according to description provided

2. The model we developed can be used for easy search as the model categorizes the video according to the text description of the YouTube video.

3. The model helps in providing accurate results for the search engine.

4. With automated youtube text classification, you'll be able to analyze and structure texts in no time. It's also a cheaper option. Hiring a trained professional to do it manually is a lot more costly.

5. With the use of YouTube text Classification, the users can easily find the desired video or recommendations just by listing the category they want.

DISADVANTAGES :

1. The problem in this method arises when there is no description provided for the youtube video which makes it unable to classify accordingly.

2. There maybe complex texts in the description which makes the model hard to interpret the category.

3. The text description of the YouTube video may lead to errors because of uncertainty in the description or text mentioned by the creator.

8 APPLICATIONS

Text classification brings automation and simplification to the daily life practice much more conveniently. There are various fields in which the model developed by us can be applied and make a good use out of it. In hospitality, where the users can just type in the word and get recommendations of videos relating to it which they can get suggestions for their activities, etc. It can be used for various websites which have videos or any text that can be categorized.The one of the applications of this model is because of the classification, just by typing the related class we can get the desired results or recommendations easily with less efforts and forbye saving time and making it easy for the consumers for using the platform .

1. Content Categorization: Our text classification solution can be utilized to automatically categorize large volumes of content, such as articles, blog posts, or documents. This can be beneficial for content management systems, news aggregators, or recommendation engines, allowing users to find relevant information efficiently.
2. Sentiment Analysis: By training our SVM model on sentiment-labeled data, one can develop a sentiment analysis system. This system can automatically analyze text data, such as customer reviews or social media posts, and determine the sentiment expressed (positive, negative, or neutral). It can help businesses gain insights into public opinions, customer feedback, and brand perception.
3. Spam Filtering: Our text classification solution can be employed to build an effective spam filtering system for emails or online platforms. By training the SVM model to distinguish between legitimate messages and spam, one can automatically filter out unwanted or malicious content, improving the user experience and security.
4. Customer Support Ticket Routing: By categorizing customer support tickets based on their content, our solution can help route tickets to the appropriate teams or departments. This ensures that customer inquiries are directed to the right specialists, reducing response times and improving customer satisfaction.
5. News Topic Classification: Our text classification solution can be applied to automatically classify news articles into different topics or categories, such as politics, sports, business, entertainment, and more. This can assist news organizations in organizing their content, recommending related articles, or delivering personalized news feeds to users.
6. Legal Document Classification: In the legal domain, our solution can be used to classify legal documents, contracts, or court filings into relevant categories. This can help legal professionals in organizing and retrieving information, conducting legal research, or identifying relevant case precedents.

These applications demonstrate the versatility and practicality of our text classification solution using SVMs across various domains and use cases. By leveraging the power of SVMs and NLP techniques, one can automate and enhance processes that involve analyzing and categorizing textual data.

9 CONCLUSION

This project has introduced a holistic data-driven approach for classifying the youtube video description using natural-language processing methods .

1. Effectiveness of SVM in NLP: Through the experiment, we discovered that SVMs are highly effective for text classification tasks in NLP. Their ability to handle high-dimensional feature spaces and capture non-linear relationships between features made them well-suited for analyzing textual data.

2. Importance of NLP Processes and Techniques: During the experiment, we learned and implemented various NLP processes and techniques. These included converting text to lowercase, stemming, and other text normalization techniques. We found that these preprocessing steps helped in standardizing the text data and reducing the dimensionality of the feature space.

3. Significance of Vectorization: In the experiment, we realized the crucial role of vectorization in NLP before passing the data into the SVM model. By employing techniques like TF-IDF vectorization, we transformed the text data into numerical vectors, allowing the SVM model to operate on structured numerical representations. This process enabled the model to effectively learn patterns and make accurate predictions.

4. Performance of SVM in NLP: The experimental results demonstrated that SVMs achieved high accuracy and strong performance in text classification tasks. The combination of NLP processes, feature engineering through vectorization, and the inherent capabilities of SVMs contributed to their success in effectively categorizing and classifying textual data into different categories.

In summary, the experiment highlighted the effectiveness of SVMs in NLP tasks, the importance of NLP processes and techniques for preprocessing text data, and the significance of vectorization as a crucial step before feeding the data into the SVM model. These insights emphasized the successful utilization of SVMs in the context of NLP and reinforced the significance of proper data preprocessing and feature engineering techniques for achieving accurate text classification results.

10 FUTURE SCOPE

The future scope of this project can be analysis of text in the comments or feedback given by the users of the platform. This can be useful for the business dealers to know whether the video or how videos belonging to a particular class are doing well in the market. The accuracy of the present model can be increased with training. The future scope of our text classification project using SVMs is promising and opens up several avenues for further exploration and enhancement. Here are potential areas of future development:

1. Advanced Deep Learning Models: While SVMs have proven effective in text classification, the field of deep learning offers promising opportunities for further improvement. Investigating advanced deep learning models such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer-based models like BERT and GPT could be explored. These models have demonstrated exceptional performance in NLP tasks and can potentially outperform traditional

SVM approaches. By leveraging the power of deep learning architectures and incorporating techniques like attention mechanisms and contextual embeddings, the project can be extended to develop more accurate and robust text classification models.

2. Multimodal Text Classification: Another exciting direction for future expansion is the integration of multimodal information. In addition to textual data, many applications incorporate other modalities such as images, videos, or audio. By combining text with these modalities, the project can be extended to tackle multimodal text classification problems. This requires exploring techniques like image or speech processing, fusion of different modalities, and developing hybrid models that can effectively leverage both textual and non-textual information. Multimodal text classification has broad implications across domains such as social media analysis, content understanding, and multimedia recommendation systems, opening up new avenues for research and practical applications.

By delving into advanced deep learning models and embracing multimodal text classification, our project can stay at the forefront of innovation in NLP and expand its capabilities to handle complex and diverse data sources. These future directions will enhance the accuracy, robustness, and flexibility of the text classification solution, enabling it to address a wider range of real-world challenges and meet the evolving demands of the NLP landscape.

11 BIBILOGRAPHY

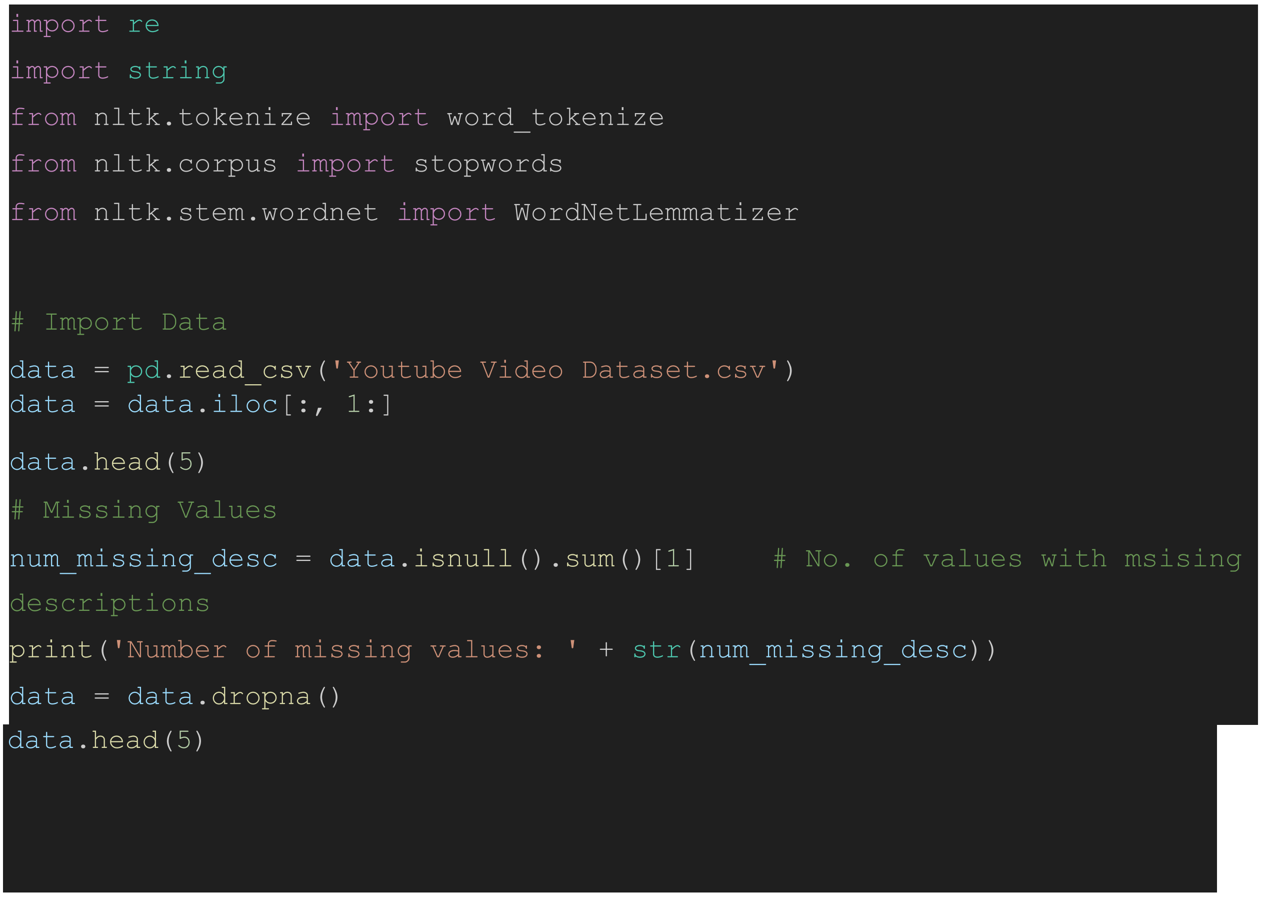
References

Text Classification with NLP: Tf-Idf vs Word2Vec vs BERT | by Mauro Di Pietro | Towards Data Science

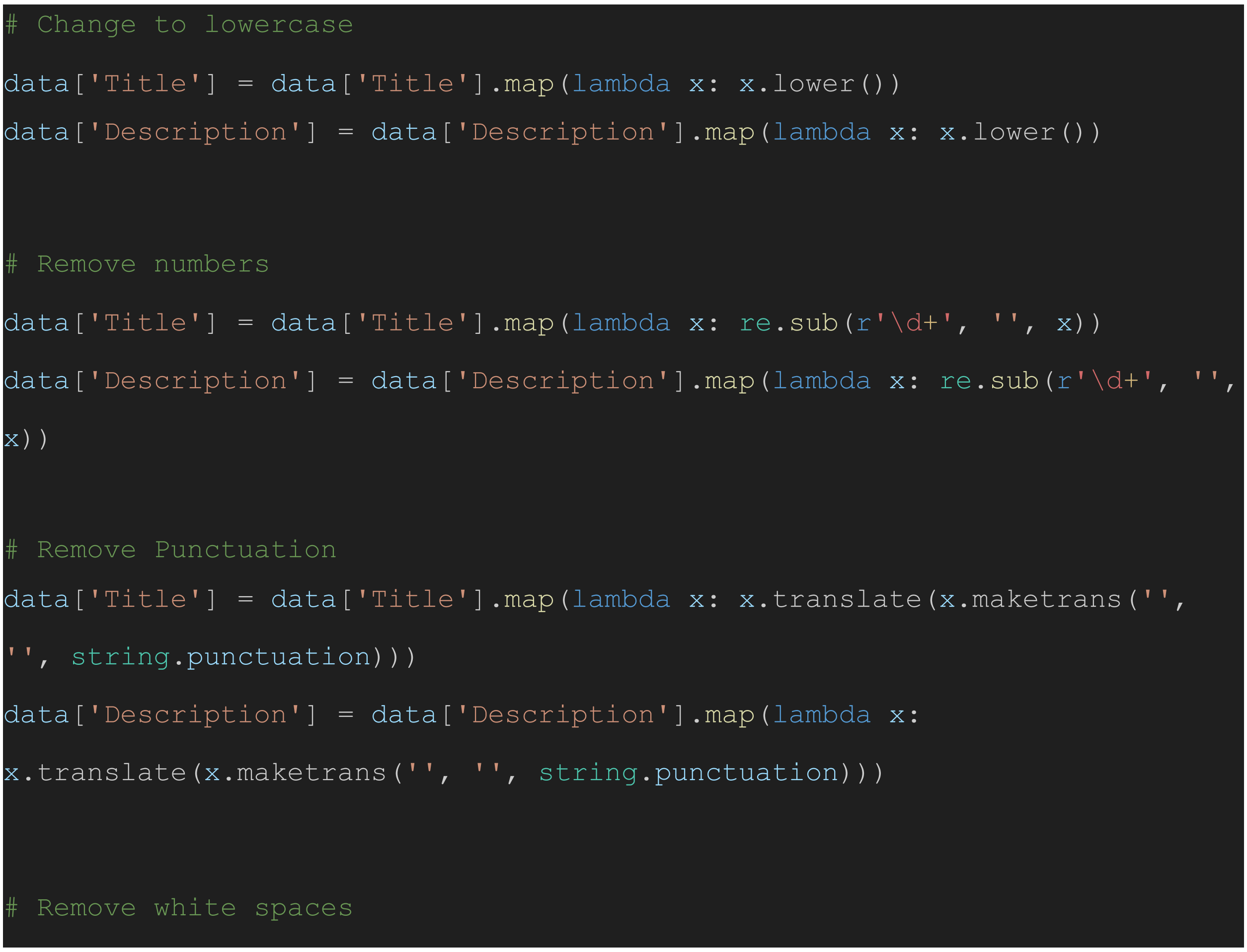
YouTube Video Classification based on Title and Description Text | IEEE Conference Publication | IEEE Xplore

A. Source Code

```
import nltk
nltk.download('all')
import pandas as pd
import nltk
from nltk.corpus import stopwords
```



```
import re
import string
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer


# Import Data
data = pd.read_csv('Youtube Video Dataset.csv')
data = data.iloc[:, 1:]

data.head(5)
# Missing Values
num_missing_desc = data.isnull().sum()[1]     # No. of values with msising
descriptions
print('Number of missing values: ' + str(num_missing_desc))
data = data.dropna()
data.head(5)
```



```
# 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)
```

```
# **Data Preprocessing**
```

```
# Encode classes
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
le.fit(data.Category)
data.Category = le.transform(data.Category)
data.head(5)
# TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_title = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2',
encoding='latin-1', ngram_range=(1, 2), stop_words='english')
tfidf_desc = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2',
encoding='latin-1', ngram_range=(1, 2), stop_words='english')
labels = data.Category
features_title = tfidf_title.fit_transform(data.Title).toarray()
features_description =
tfidf_desc.fit_transform(data.Description).toarray()
print('Title Features Shape: ' + str(features_title.shape))
print('Description Features Shape: ' + str(features_description.shape))


# Plotting class distribution
data['Category'].value_counts().sort_values(ascending=False).plot(kind='ba
r', y='Number of Samples',


title='Number of samples for each class')
# Best 5 keywords for each class using Title Feaures
from sklearn.feature_selection import chi2
import numpy as np
N = 5
for current_class in list(le.classes_):
    current_class_id = le.transform([current_class])[0]
    features_chi2 = chi2(features_title, labels == current_class_id)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf_title.get_feature_names_out())[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(current_class))
    print("Most correlated unigrams:") print('-
    ' *30)
```

```
    print('. {}'.format('\n. '.join(unigrams[-N:])))
    print("Most correlated bigrams:")
    print('-' *30)
    print('. {}'.format('\n. '.join(bigrams[-N:])))
    print("\n")
# Best 5 keywords for each class using Description Features
from sklearn.feature_selection import chi2
import numpy as np
N = 5
for current_class in list(le.classes_):
    current_class_id = le.transform([current_class])[0]
    features_chi2 = chi2(features_description, labels == current_class_id)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf_desc.get_feature_names_out())[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(current_class))
    print("Most correlated unigrams:") print('-
    ' *30)
    print('. {}'.format('\n. '.join(unigrams[-N:])))
    print("Most correlated bigrams:")
    print('-' *30)
    print('. {}'.format('\n. '.join(bigrams[-N:])))
    print("\n")
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)
```

```
X_train.head()

y_train.head()
```

```
# Naive Bayes
nb = MultinomialNB().fit(features, y_train)
# SVM
svm = linear_model.SGDClassifier(loss='modified_huber',max_iter=1000,
tol=1e-3).fit(features,y_train)

from keras.preprocessing.text import Tokenizer
from keras.utils.data_utils import pad_sequences
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
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)
# Define LSTM Model
from keras.models import Sequential
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(6, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
print(model.summary())
# Training LSTM Model
epochs = 5
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.1)
import matplotlib.pyplot as plt
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();


plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show();
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, 1:3],
data['Category'], random_state = 0)
X_test_title_features = tfidf_title.transform(X_test['Title']).toarray()
X_test_desc_features =
tfidf_desc.transform(X_test['Description']).toarray()
test_features = np.concatenate([X_test_title_features,
X_test_desc_features], axis=1)
```

```
pip install scikit-plot
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import scikitplot as skplt


X_test_title_features = tfidf_title.transform(X_test['Title']).toarray()
X_test_desc_features =
tfidf_desc.transform(X_test['Description']).toarray()
test_features = np.concatenate([X_test_title_features,
X_test_desc_features], axis=1)


# Naive Bayes
y_pred = nb.predict(test_features)
y_probas = nb.predict_proba(test_features)


print(metrics.classification_report(y_test, y_pred,
                                    target_names=list(le.classes_)))


conf_mat = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(conf_mat, annot=True, fmt='d',
xticklabels=list(le.classes_),
yticklabels=list(le.classes_))
```

```
plt.title('Confusion Matrix - Naive Bayes')

plt.show()
```

```
skplt.metrics.plot_precision_recall_curve(y_test, y_probas)
plt.title('Precision-Recall Curve - Naive Bayes')


plt.show()

# SVM

y_pred = svm.predict(test_features)

y_probas = svm.predict_proba(test_features)



print(metrics.classification_report(y_test, y_pred,

                                          target_names=list(le.classes_)))



conf_mat = confusion_matrix(y_test, y_pred) fig, ax
= plt.subplots(figsize=(10,10))


sns.heatmap(conf_mat, annot=True, fmt='d', xticklabels=list(le.classes_),

yticklabels=list(le.classes_))

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix - SVM')

plt.show()


skplt.metrics.plot_precision_recall_curve(y_test, y_probas)

plt.title('Precision-Recall Curve - SVM')

plt.show()

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state =

42)

y_probas = model.predict(X_test)

y_pred = np.argmax(y_probas, axis=1)

y_test = np.argmax(Y_test, axis=1)



print(metrics.classification_report(y_test, y_pred,
                                          target_names=list(le.classes_)))
```

```
conf_mat = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(10,10))

sns.heatmap(conf_mat, annot=True, fmt='d', xticklabels=list(le.classes_),

yticklabels=list(le.classes_))
```

```
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix - LSTM')
plt.show()

skplt.metrics.plot_precision_recall_curve(y_test, y_probas)
plt.title('Precision-Recall Curve - LSTM')
plt.show()
```