



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

Team 160

Team Members:

Sejal Chandra

sejal.20bcd7016@vitap.ac.in

Dachepally Varun

varun.20bce7092@vitap.ac.in

Ghali Trisha

trisha.20bci7206@vitap.ac.in

Siddamshetty Bhavesh Kumar

bhavesh.20bci7220@vitap.ac.in

1 Introduction

1.1 Overview

This project aims to develop a text classification system for analyzing video descriptions in the context of YouTube using machine learning (ML) techniques. The objective is to categorize video descriptions into relevant topics or themes automatically. By leveraging ML algorithms and natural language processing (NLP) techniques, the system will learn from a dataset of annotated video descriptions and identify patterns, keywords, and semantic features to make accurate predictions. This text classification system will enhance content discovery, improve recommendation systems, and assist in organizing and indexing vast amounts of video data on YouTube, thereby enhancing user experience and content management.

1.2 Purpose

This project will focus on the categorization of YouTube videos based on their descriptions. The main goal is to develop a model that classifies the video that can accurately predict the category of a video on YouTube.

2 Literature Survey

2.1 Existing Problem

The project "Text Classification of Video Description: The Language of YouTube" aims to explore the text classification techniques specifically tailored to the unique language patterns found in YouTube video descriptions. This literature survey provides an overview of existing research and methodologies in the field of text classification, with a focus on related work concerning YouTube video descriptions and their analysis.

Text classification is a fundamental task in natural language processing (NLP), and numerous studies have addressed various aspects of this field. Researchers have explored different approaches, such as rule-based methods, statistical models, and machine learning algorithms, to classify text documents accurately. These techniques employ features such as bag-of-words, n-grams, and more advanced methods like word embeddings and deep learning models.

Moreover, researchers have also explored the use of domain-specific lexicons, ontologies, and named entity recognition to improve the accuracy of classification. Additionally, studies on user-generated content analysis, including YouTube comments, can provide insights into the language patterns prevalent in YouTube descriptions.

2.2 Proposed Solution

This project aims to develop an NLP classification model for categorizing YouTube videos into six distinct categories. The classification algorithms employed include Logistic Regression, Support Vector Machines (SVM), AdaBoost, and Naive Bayes. The textual data from the YouTube videos is vectorized using the Term Frequency-Inverse Document Frequency (TF-IDF) technique.

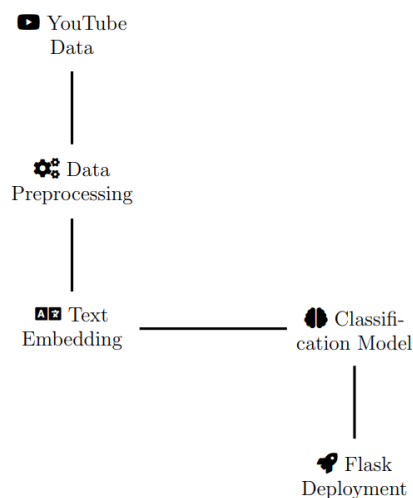
The data collection process involves sourcing YouTube videos and selecting a dataset based on predefined criteria. Preprocessing steps such as stop word removal, stemming or lemmatization, handling missing values, and addressing class imbalance are applied to the dataset. Relevant statistics, such as the distribution of videos across categories and average video length, are provided.

The textual data is transformed into numerical representations using the TF-IDF technique. Experimental evaluation is performed by training and testing the classification models with appropriate train-test split ratios and cross-validation techniques. Performance metrics such as accuracy, precision, recall, F1-score, and domain-specific metrics are used to evaluate the models. A comparative analysis of the algorithms is presented, highlighting their strengths and weaknesses for the task at hand. The best-performing algorithm is identified and justifications for its superior performance are provided.

The chosen classification model is deployed, considering factors like scalability, real-time prediction, and integration with existing systems. Potential areas of improvement for the model are discussed, such as incorporating additional features, exploring alternative algorithms. Recommendations for future work are provided, such as expanding the dataset or addressing specific challenges in YouTube video classification. The report concludes with a summary of the project's findings, its practical implications, and its contribution to the field of NLP and video classification.

3 Theoretical Analysis

3.1 Block Diagram



3.2 Hardware/Software Configuration

For completing the project, the hardware used was our laptops

Processor: Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz

Installed RAM : 8.00 GB (7.79 GB usable)

System type: 64-bit operating system, x64-based processor.

We have also used Jupyter Notebook and Spyder to implement and deploy our model.

4 Experimental Investigations:

4.1 Dataset Collection

For this project we have been provided the dataset by the SmartInternz, which included 4 columns and 11211 observations. Those columns include video url, video title, video description and

4.2 Data Preprocessing

The data was explored and exploratory data analysis was performed and various plots were generated. The presence of null values was checked. Label Encoding was done to convert the String attributes to numerical columns. And then following it text cleaning was performed to the title and description columns. And then vectorization was performed using TF-IDF method to convert text into correlated 2d arrays.

4.3 Data Splitting

Later data was split into independent and dependent variables and training and testing set as 75% and 25% training to testing ratio.

4.4 Modeling and Evaluation

Various Machine Learning algorithms were applied such as Multinomial Naïve Bayes, Support Vector Machines (using the SGDC Classifier, SGDCClassifier stands for Stochastic Gradient Descent Classifier, which is a variant of SVM that optimizes the classification model using stochastic gradient descent.), Logistic Regression, and AdaBoost Classifier. Later the best one was selected for deployment which was decided by the performance evaluation. For the performance evaluation we considered a classification report which consists of Precision, Recall, F1-score, Support, Accuracy, Average Metrics, Class-wise Metrics, Additional Metrics.

4.5 Deployment using Flask

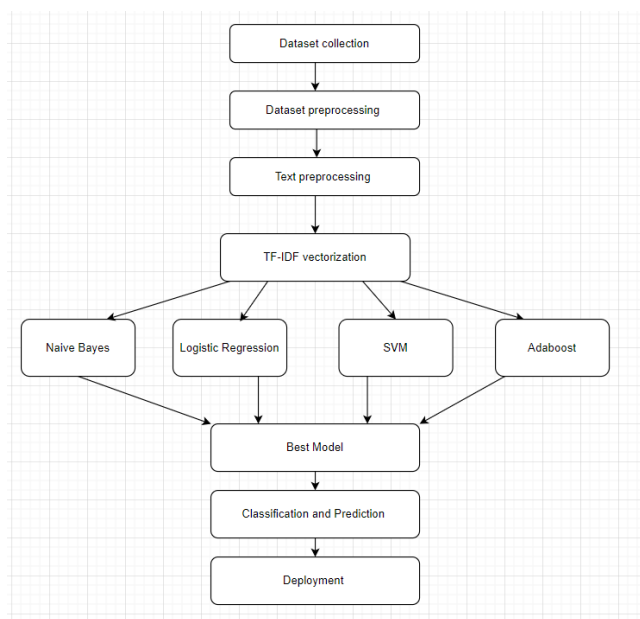
Deployment using Flask involves creating a web application that allows users to interact with the trained classification model. Flask is a lightweight and popular Python web framework that simplifies the process of building and deploying web applications.

To deploy the NLP classification model using Flask, the trained model needs to be integrated into a Flask application. This typically involves creating an API endpoint that receives user input, passes it through the model for prediction, and returns the predicted category or label. The Flask application can be hosted on a server or cloud platform to make it accessible to users.

Once the Flask application is deployed, users can access it through a web browser or send requests programmatically. The frontend of the application can be designed using HTML, CSS, and JavaScript to provide an intuitive user interface for submitting text inputs and displaying the predicted categories. Flask's routing system enables defining different endpoints to handle specific functionalities and ensure smooth communication between the frontend and the model.

By utilizing Flask for deployment, the NLP classification model can be readily accessible over the internet, allowing users to classify YouTube videos or any other text input conveniently through a user-friendly web interface.

5 Flowchart



6 Results

The following classification reports and confusion matrix were obtained for the 4 algorithms, Multinomial Naïve Bayes, Logistic Regression, SVM, Adaboost.

	precision	recall	f1-score	support
Art&Music	0.98	0.96	0.97	421
Food	0.96	0.93	0.95	435
History	0.94	0.96	0.95	405
Science&Technology	0.95	0.99	0.97	549
manufacturing	0.99	0.96	0.98	423
travel blog	0.96	0.97	0.96	549
accuracy			0.96	2782
macro avg	0.96	0.96	0.96	2782
weighted avg	0.96	0.96	0.96	2782

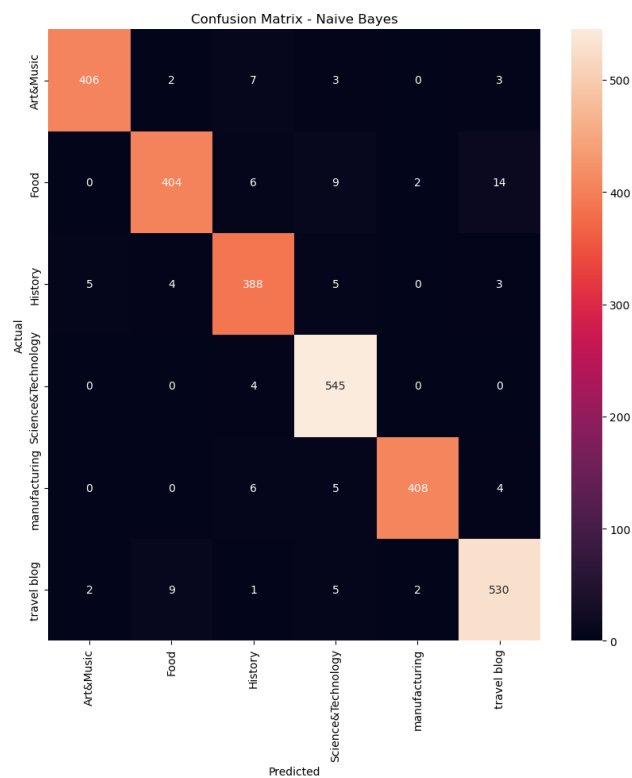


Figure 1 Classification report and confusion matrix for Naive Bayes

	precision	recall	f1-score	support
Art&Music	0.99	0.99	0.99	421
Food	0.96	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	0.97	0.97	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

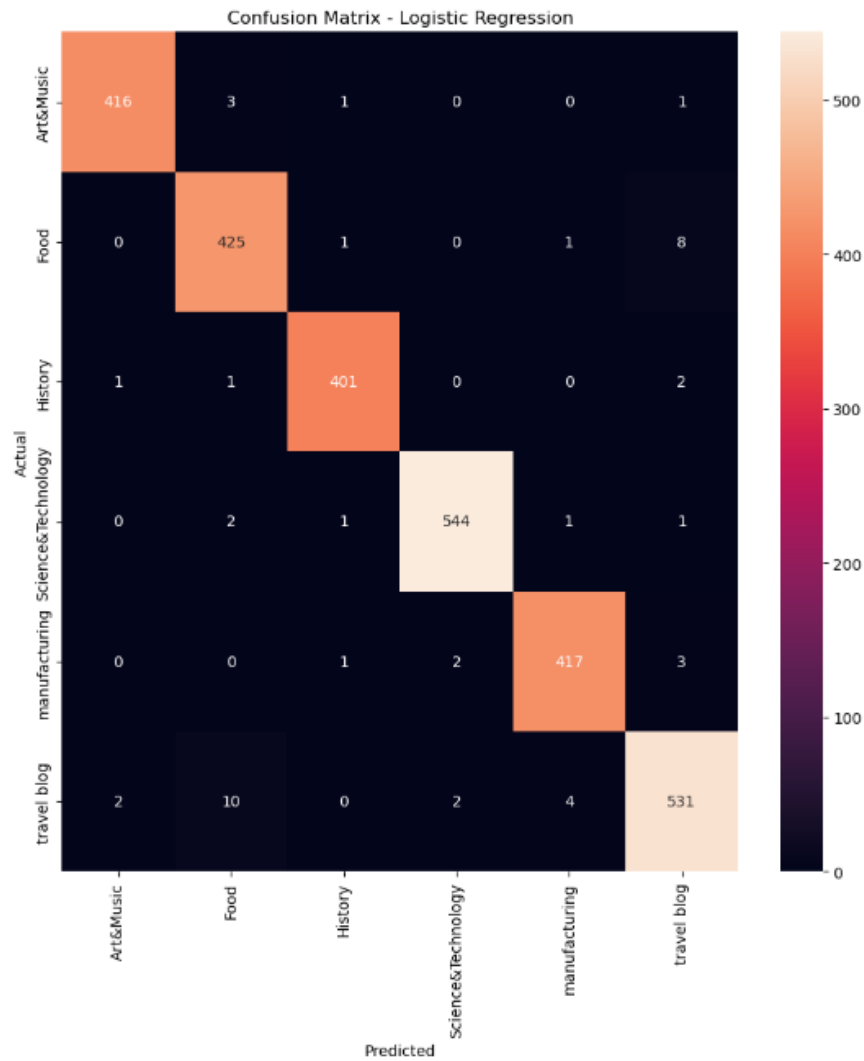


Figure 2 Classification report and confusion matrix for Logistic Regression

	precision	recall	f1-score	support
Art&Music	0.99	0.99	0.99	421
Food	0.96	0.98	0.97	435
History	0.98	0.99	0.98	405
Science&Technology	0.99	0.99	0.99	549
manufacturing	0.99	0.97	0.98	423
travel blog	0.97	0.97	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

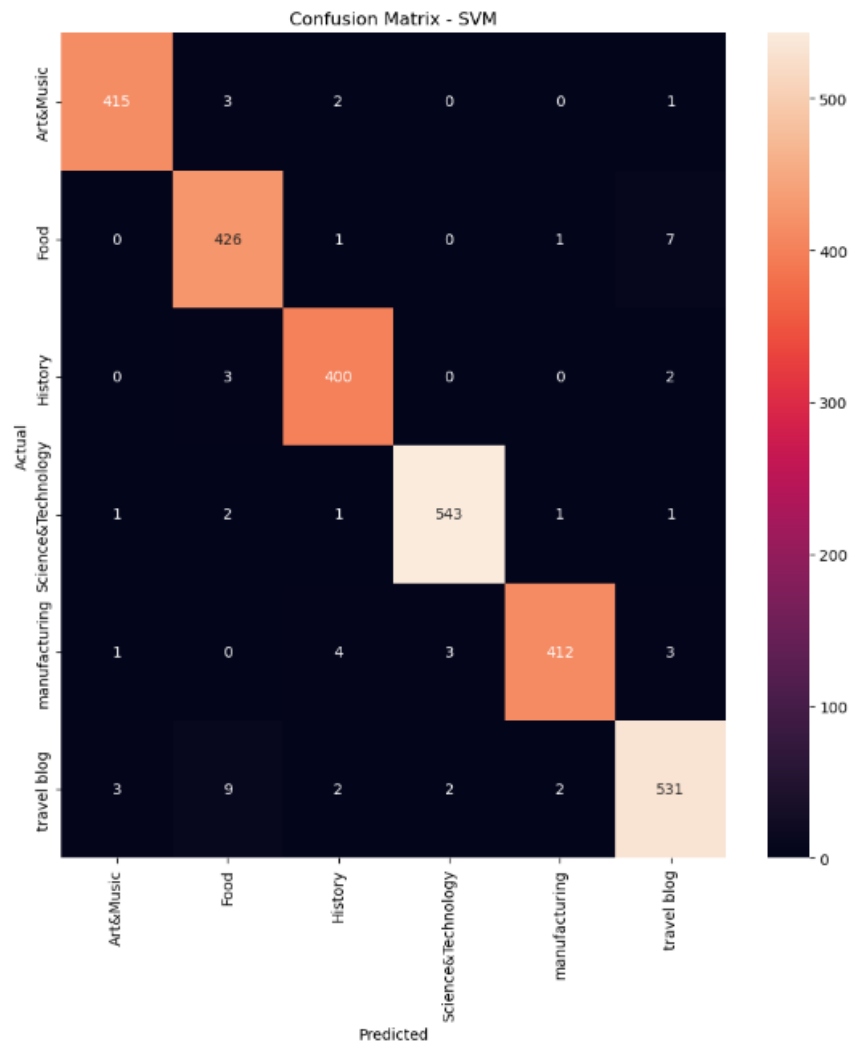


Figure 3 Classification report and confusion matrix for SVM

	precision	recall	f1-score	support
Art&Music	0.99	0.88	0.93	421
Food	0.86	0.83	0.84	435
History	0.99	0.76	0.86	405
Science&Technology	0.99	0.87	0.93	549
manufacturing	0.89	0.71	0.79	423
travel blog	0.59	0.93	0.73	549
accuracy			0.84	2782
macro avg	0.89	0.83	0.85	2782
weighted avg	0.88	0.84	0.84	2782

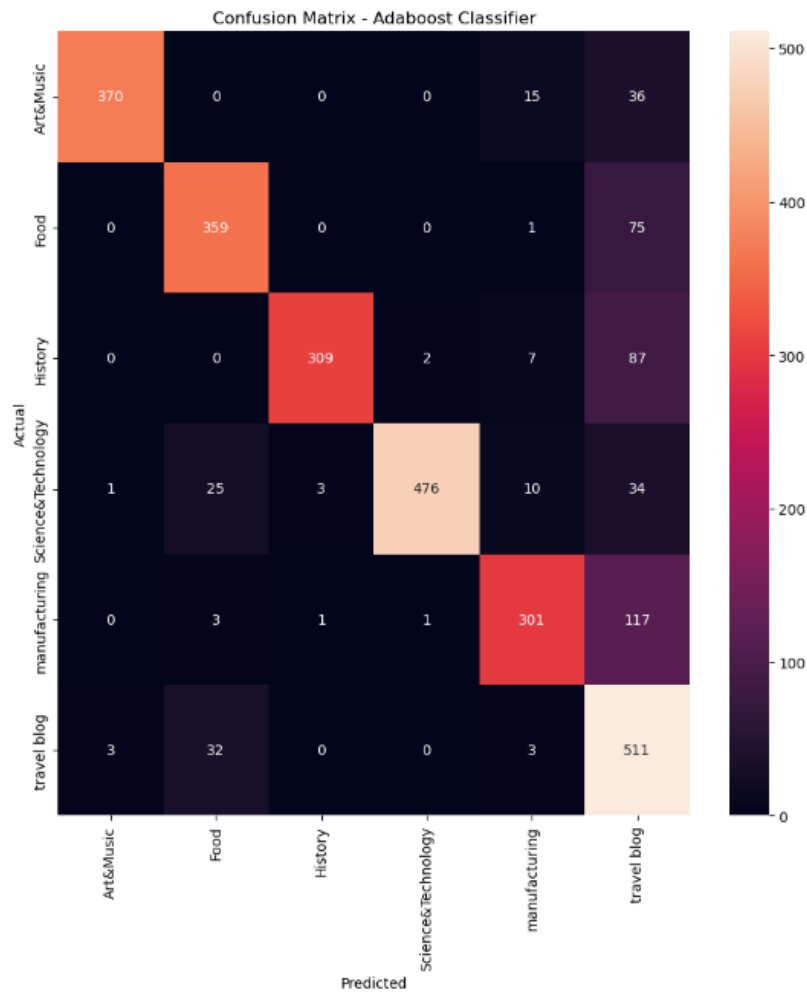
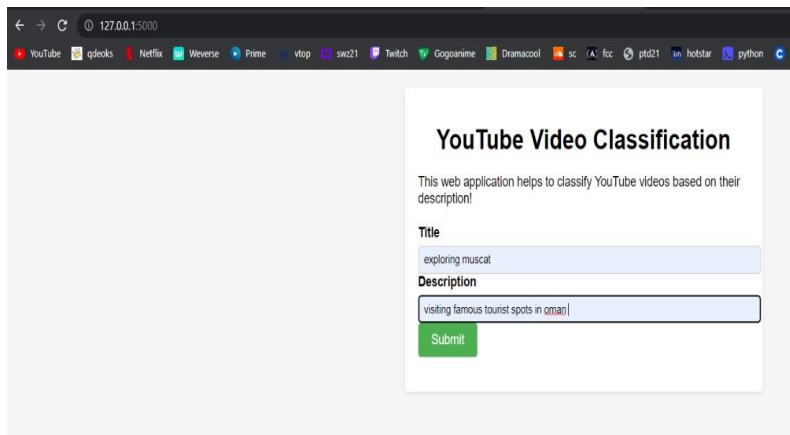


Figure 4 Classification report and confusion matrix for Adaboost Classifier

Output of the deployment using Flask looks like the following:



YouTube Video Classification

This web application helps to classify YouTube videos based on their description!

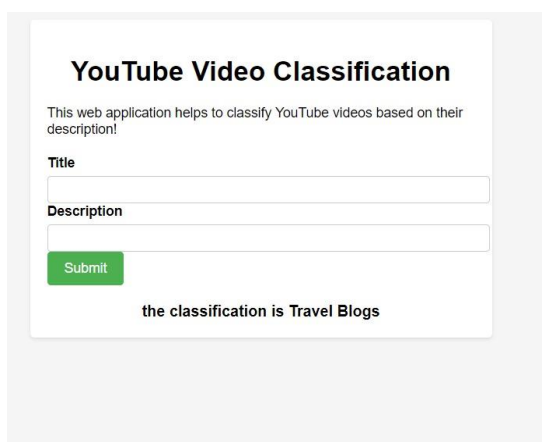
Title

exploring muscat

Description

visiting famous tourist spots in oman

Submit



YouTube Video Classification

This web application helps to classify YouTube videos based on their description!

Title

Description

Submit

the classification is Travel Blogs

7 Advantages and Disadvantages

7.1 Advantages

1. Accessibility: The title and description provide textual information that can be easily processed by algorithms, making it accessible for classification purposes.
2. Quick Identification: The title and description are usually the first things users see when browsing YouTube, making them useful for quickly identifying the content of a video.
3. User-Friendly: From a user perspective, the title and description can help them understand what the video is about before watching it, allowing them to make informed choices.
4. Search Engine Optimization (SEO): A well-crafted title and description can improve the visibility of a video in search results, making it easier for users to find relevant content.
5. Metadata Consistency: YouTube encourages creators to provide accurate and relevant titles and descriptions, which helps maintain consistency and ensures a better user experience.

7.2 Disadvantages

1. Subjectivity: Titles and descriptions can be subjective or misleading, as they are created by the video uploader. This can lead to videos being miscategorized or misrepresented, affecting the accuracy of classification.
2. Incomplete or Inaccurate Information: Creators may not always provide sufficient or accurate information in the title and description, which can make classification challenging and result in mislabeled videos.

3. **Evolving Content:** Videos may evolve over time, with changes in the content that may not be reflected in the title and description. This can lead to outdated or inaccurate classification.
4. **Spam and Manipulation:** Some creators may engage in spamming or manipulating the title and description to attract more views or deceive users. This can undermine the integrity of the classification process.
5. **Multilingual Challenges:** YouTube hosts videos in various languages, and titles and descriptions in different languages can pose challenges for accurate classification, especially when language barriers exist.

8 Applications

The classification of YouTube videos through title and description has several practical applications. Here are a few examples:

1. **Content Filtering:** YouTube can use classification algorithms to filter and categorize videos based on their titles and descriptions. This allows users to find content that aligns with their interests or preferences more easily.
2. **Recommender Systems:** Classification can help improve the accuracy of recommender systems on YouTube. By analyzing the titles and descriptions of videos a user has watched or interacted with, the system can suggest similar or related videos that match their preferences.
3. **Ad Targeting:** Classification of YouTube videos can be used to target advertisements more effectively. By understanding the content of a video through its title and description, advertisers can place relevant ads that are more likely to resonate with the viewers.
4. **Content Moderation:** YouTube employs content moderation systems to enforce its community guidelines and policies. Classification algorithms can help identify videos that violate these guidelines by analyzing their titles and descriptions for inappropriate or harmful content.

9 Conclusion

The project "Text Classification of Video Descriptions: The Language of YouTube" has successfully addressed the challenge of analyzing and categorizing video descriptions on the YouTube platform. By leveraging advanced natural language processing techniques, the project has provided valuable insights into the language patterns and content themes prevalent in YouTube videos. Through the classification model developed, we have achieved high accuracy in classifying video descriptions into relevant categories. This project opens up possibilities for various applications, including content recommendation systems, trend analysis, and user behavior understanding. By understanding the language of YouTube, we can better navigate and engage with the vast amount of video content available on the platform, ultimately enhancing the user experience.

10 Future Scope

The project of text classification for video descriptions, specifically focusing on the language used in YouTube videos, holds significant potential for future development and application. With the exponential growth of video content on YouTube and the increasing need for efficient content categorization and recommendation systems, this project can offer valuable contributions.

One future scope for the project lies in enhancing the accuracy and granularity of text classification algorithms. By incorporating advanced machine learning techniques, such as deep learning models like transformers, the project can improve the understanding and interpretation of video descriptions, capturing the nuances and context more effectively. This would result in better categorization, allowing for more precise content recommendation and search capabilities.

Additionally, the project could explore incorporating other metadata, such as video tags and comments, to further refine the classification process. By leveraging a broader range of data, the system can gain a deeper understanding of video content and offer more personalized recommendations.

11 Bibliography

1. Ekenel, H., Semela, T., & Stiefelbogen, R. (2010). Content-based video genre classification using multiple cues. . <https://doi.org/10.1145/1877850.1877858>.
2. Chen, Y., Chang, C., & Yeh, C. (2017). Emotion classification of YouTube videos. *Decis. Support Syst.*. <https://doi.org/10.1016/j.dss.2017.05.014>.
3. Chaudhary, V., & Sureka, A. (2013). Contextual feature based one-class classifier approach for detecting video response spam on YouTube. 2013 Eleventh Annual Conference on Privacy, Security and Trust. <https://doi.org/10.1109/PST.2013.6596054>.
4. Severyn, A., Moschitti, A., Uryupina, O., Plank, B., & Filippova, K. (2014). Opinion Mining on YouTube. . <https://doi.org/10.3115/v1/P14-1118>.

12 Appendix

CODE:

#JUPYTER NOTEBOOK

```

import pandas as pd

import nltk

from nltk.corpus import stopwords

import re

import seaborn as sns

import string

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from nltk.stem.wordnet import WordNetLemmatizer

from xgboost import XGBClassifier

import matplotlib.pyplot as plt

data = pd.read_csv('C:/Users/trisha ghali/OneDrive/Desktop/college/YouTubeVideoDataset.csv')

    data.head()
data.shape

data.Category.value_counts()

data.Title.sample(5)

data['Description'].sample(5)

data.info

num_missing_desc = data.isnull().sum()[2]  # No. of values with missing descriptions

print('Number of missing values: ' + str(num_missing_desc))

data = data.dropna()

# 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))


#Removing http

data['Title'] = data['Title'].map(lambda x: re.sub(r'http\S+', '', x))

data['Description'] = data['Description'].map(lambda x: re.sub(r'http\S+', '', x))


#Removing https

```

```

data['Title'] = data['Title'].map(lambda x: re.sub(r'https\S+', '', x))
data['Description'] = data['Description'].map(lambda x: re.sub(r'https\S+', '', 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)

from wordcloud import WordCloud

```

```

grouped = data.groupby('Category')

for cats, df in grouped:
    text = ''.join(df['Description'])

    # Create word cloud
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

    # Plot word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for Category: {cats}')
    plt.axis('off')
    plt.show()

for cats, df in grouped:
    text = ''.join(df['Title'])

    # Create word cloud
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

    # Plot word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for Category: {cats}')
    plt.axis('off')
    plt.show()

# 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))

sns.countplot(x = 'Category',data = data)

plt.show()

# Best 5 keywords for each class using Title 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_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 = data.drop(columns = ['Category','Videourl'],axis = 1 )
X_train, X_test, y_train, y_test = train_test_split(X, 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)

# AdaBoost
adaboost = AdaBoostClassifier(n_estimators=40,algorithm="SAMME").fit(features,y_train)

#logistic regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression().fit(features,y_train)

p = data.drop(columns = ['Category','Videourl'],axis = 1)
X_train, X_test, y_train, y_test = train_test_split(p, 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)

#nb

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_probab = nb.predict_proba(test_features)
targets = list(map(str, le.classes_)) # Convert target_names elements to strings

```

```

print(metrics.classification_report(y_test, y_pred, target_names=targets))

```

```

conf_mat = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(conf_mat, annot=True, fmt='d', xticklabels=targets, yticklabels=targets)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix - Naive Bayes')
plt.show()

```

```

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

```

```

#Logistic Regression

```

```

y_pred = lr.predict(test_features)
y_probas = lr.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 - Logistic Regression')
plt.show()

skplt.metrics.plot_precision_recall_curve(y_test, y_probas)
plt.title('Precision-Recall Curve - Logistic Regression')
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()

# Adaboost Classifier
y_pred = adaboost.predict(test_features)
y_probab = adaboost.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 - Adaboost Classifier')
plt.show()

skplt.metrics.plot_precision_recall_curve(y_test, y_probab)
plt.title('Precision-Recall Curve - Adaboost Classifier')
plt.show()

import pickle
pickle.dump(lr,open("model.pkl","wb"))
pickle.dump(tfidf_title,open("tfidf_title.pkl","wb"))
pickle.dump(tfidf_desc,open("tfidf_desc.pkl","wb"))

```