EUROPEAN UNIVERSITY OF LEFKE

FACULTY OF ENGINEERING

Graduation Project 2

SENTIMENT ANALYSIS FOR MOVIE RECOMMENDATIONS

HILARIO JUNIOR NENGARE

174682

Machine Learning

- Machine Learning is a network of models that take input data to predict future inputs.
- It is a branch of Artificial Intelligence which is focused on the implementation of models and mathematical algorithms to make models learn from datasets.
- The goal is to implement models that can learn on their own or adhere to mediated learning.
- To improve models, we expose models to more data sets and improve feedback loops and so forth.
- Although Machine Learning is a subset of Artificial Intelligence, it comprises of its own subsets; one of which is the focal point of this paper – SENTIMENT ANALYSIS.

Supervisor

Dr. Zafer Erenel

Publish Date

14-03-2024

Table Of Contents

HILARIO JUNIOR NENGARE

174682

1.Introduction

Sentiment Analysis and Its Levels

Document level Sentiment Analysis

- Document level analysis is carried out on a single document
- It's not used as often as other levels of sentiment analysis.
- Document level sentiment analysis is usually applied to chapters and/or pages of books to yield a single polarity.
- With document-level analysis, domain-specificity produces high accuracy.
- The feature vector must be constrained and very specific to a particular domain.

Sentence level Sentiment Analysis

- Within this level; analysis of single sentences is done, and a polarity is appointed.
- Documents carry sentences with different sentiments associated with them.
- A larger training dataset is used, and more processing resources are needed.
- These sentence-based polarity can be used independently or aggregated to yield document-based polarity.

Phrase level Sentiment analysis

- Phrases contain either single aspects or many aspects.
- Analysis is carried out on those aspects.

Finally,

Aspect level sentiment analysis

- In this level we analyse every word and assign a polarity to each word.
- At this level analysis becomes more insightful and granular hence I will employ this level throughout the project.

1.1 Problem definition

- Sentiment Analysis, also known as emotion AI or opinion mining, is a way in which we extract, investigate, find traces of affective states and intent behind text.
- Sentiment Analysis is applied in user and/or client written feedback e.g. movie reviews (which will be my focus in this project), social media, survey material and so forth.
- Sentiment Analysis is a process in which we can classify text using reviewer sentiments. This includes positive polarity, negative polarity and neutral.
- For instance, we could have a data set with user reviews on a certain movie

Table 1: An example dataset of user reviews

Viewers	Review	Polarity
User 1	I hate this movie.	negative
User 2	I love this movie.	positive
User 3	I don't love it, but I don't hate it neither.	neutral

- As you can see, different users can share different sentiments towards a movie, and it is these reviews that we look at and then classify them in accordance with positive or negative polarity, sometimes we come across sentiments with weak polarity such as user 3's sentiment, that would be very difficult to classify.
- Sentiment Analysis basically figures out if text is positive, negative or neutral and then we can devise a recommendation system from thereon.

Sentiment Analysis and Natural Language Processing

- Natural Language Processing is an interdisciplinary field of Computer Science and Linguistics which is primarily focused on computer to human interaction and vice versa.
- It enables computers to dissect text corpora into well tagged and meaningful input or output.
- NLP consists of algorithms and models that are used for generating human-like text or analysing bodies of text and "understanding" bodies of text to produce output or input.
- Now, there are many tasks of NLP that I would be deploying in this project in order to carry out proper Opinion Mining on Movie Reviews.

- These tasks include;
    - **Text And Speech Processing** – which involves many subtasks like Optical Character Recognition, Speech Segmentation, Speech Recognition, Text to Speech et al. My primary focus will be on Word Segmentation also known as tokenization in which I will be separating blocks of text into single words.
    - **Morphological Analysis** – Processes like Lemmatization in which one reduces a word to its dictionary form known as a Lemma, stemming which is the same as Lemmatization but does not deflate a word to its dictionary form but to its base form.
    - **Syntactic Analysis**- This is where we analyse the grammatical set up of a sentence using Grammar Induction process, sentence breaking process and parsing.

- There are other subtasks pertaining to NLP, but the question that all this poses is **what Does NLP have to Do with Sentiment Analysis?**
- Well, the subtasks I listed form the foundation of Sentiment Analysis. I plan to use the subtasks to extract features from text data and then determine the sentiment beneath.
- This will be done with machine learning classifier models that I will train on labelled movie review datasets to perform sentiment analysis.

1 Natural Language Processing engulfs all the tasks that are tied to computer science and linguistics whereas Sentiment Analysis is the specific application of Natural Language Processing techniques to figure out the emotion behind text corpora.

1.2 Goals

- What I am gunning for in this project is to exhibit the impact sentiment analysis has or can possibly have in the foreseeable AI future.
- I will be focusing on a movie reviews' dataset, and then weed through verbatim to tag negative, positive and neutral affective states.
- In turn I intend to attain acute knowledge of Natural Language Processing which I will use within my career as a Software Engineer.
- And for users, a personalised experience and movie suggestions aligned with their preferences, mind you this can easily be spread out to other recommender systems also.
- Concisely, I aim to use Sentiment Analysis in the movies domain to attain:
    - **Personalised movie recommendations**

    Improve viewer space by recommending movies that are tailored towards the viewer's sentiments.

    - **User Engagement**

    Offering personalised recommendations tend to enhance viewer engagement as well ensuring user traffic of sorts.

    - **Context**

    Very vital to offer context-aware recommendations to users. Context-aware recommendations will align with the user's mood state and recommend perfect movies.

    - **User Preferences**

    We can track nuanced preferences that include genres and actors, and we can base these preferences on emotional context.

- **User Feedback**

In any product user feedback is such a prolific aspect of the product life cycle as we can improve something from user feedback alone.

- **Improved Content Discovery**

We can recommend movies users did not explicitly consider, but they will be based on their emotional criteria.

2. Literature Survey

- There is a plethora of research with findings and challenges pertaining to Sentiment Analysis for Movie Recommendations.
- Research suggests that box office success of movies can be predicted by analysing sentiments within viewer reviews.

Other Projects

- Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan produced a study to address the difficulties faced in classifying documents using sentiments in contrast to topic-based classification of documents [1].

- They chose to focus on a movie reviews dataset to clearly distinguish positive to negative polarity and neutral.

- In their research they annotated case studies and applications aligning with topical document classification – classifying documents in terms of its subject matter. Some of these case studies are as follows

- **Source Style Document Classification (Biber, 1988)**
    - This aimed to classify documents in accordance with the document's source or source-style using statistical methods.
    - It served as a methodology to discern patterns or variations or trends in the way the document's language was being used.
    - SSDC analysed variations of the writing style and utilised computational mathematics' algorithm to manipulate large text corpora.

- **High-brow vs. popular or low-brow (Monstellar and Wallace 1984)**

Focus was placed on language style variations, distinguishing high-brow from low-brow.

- **Stylistic Variation in Language (Argamon and Engelson et al. 1998)**

Extended the study of language-style and contributed to the factors that influence the variations in language.

- **Stylistic Analysis (Tomokiyo and Jones 2001)**

Related to language variations; it was an extended exploration focused on statistical methods used in denoting patterns found in written text.

- **Sentiment Analysis and Language Style (Kessler et al. 1997)**

Contributed insights of classifying text corpora on the grounds of style and/or sentiment.

- **Subjective Genres (Karlgren and Cutting 1994)**
  - Objective was to determine the genre of the text and included subjective genres. This introduced classification by genre as a new field of research.
  - Finn and others in 2002 further explored subjective genres building upon the work of Karlgren and Cutting.
- **Features and Subjective Language (McKeown 1997, Hatzivassiloglou and Wiebe 2000, 2001)**
  - In 1997 McKeown and Hatzivassiloglou research was grounded on classifying the semantic orientation of words and phrases.
  - In 2000 Hatzivassiloglou and Wiebe focused on detecting features indicating the usage of subjective language which they utilise for categorising documents by subjectivity.

- **Sentiment Based Classification (Turney and Littman 2002)**
  - Turney and Littman explored sentiment-based classification of documents using unsupervised learning.
  - This introduced unsupervised learning algorithms for sentiment analysis techniques.

- **Knowledge-based Sentiment Analysis (Hearst 1992 and Sack 1994)**

- Hearst dived into the study that involved classification of documents using models that are influenced by cognitive linguistics.
- Sack expressed the idea to integrate cognitive linguistics into sentiment-based categorization.

- **Discriminant Word Lexicons (Huettner and Subasic 2000, Das and Chen 2001, Tong 2001)**
  - Huettner and Subasic explored the idea of sentiment-based categorization of text corpora using discriminant word lexicons. Expanding on the notion of knowledge-based sentiment analysis.
  - Tong, Das and Chen in the year 2001 built on the work of Huettner and Subasic where they focused on sentiment-based categorization of text corpora.

- **Unsupervised Learning Based on Mutual Information (Turney 2002)**
  - Turney researched unsupervised learning for sentiment analysis.
  - At this point we see the dawn of current studies being carried out leading to sentiment analysis.
  - There are other baseline experiments exploring the inherent difficulty of sentiment-based classification with comparative analysis of different methods, highlighting the challenges of sentiment analysis.
  - Turney in 2002 figured that movie reviews are the most difficult to classify using sentiments.
  - He reported an accuracy that fell around 65.83% on a dataset of size 120

3. Background Information

3.1 Required software

Below are the tools I will be using in this project

- **Python:**
    - Python is fit for Natural Language Processing as it resembles English language and pseudocode.
    - Its simplicity is ideal, as one gets to focus more on Natural Language Processing rather than focus on the language's syntax.
    - Python offers a big and robust support community, many problems have ready-made solutions on platforms like stack overflow, Quora et al.
    - Python offers a plethora of NLP libraries as well as machine learning libraries e.g NLTK, TextBlob, sci-kit learn et cetera.
    - I have used NLTK and scikit-learn to train the model for predictions.

- **NLTK:**
    - NLTK is a Python library that works with human language data.
    - It is powerful and provides simplistic interfaces to over 50 corpora and lexical resources.
    - It has a large support online community.
    - NLTK has many libraries that one can utilise for classification, parsing, tagging et al.
    - NLTK is compatible with Windows, Mac OS X and Linux.

- NLTK is the top tier library to perform sentiment analysis.
- NLTK has provided me with classes such as PorterStemmer to stem reviews and a tokenization class to tokenize the reviews into tokens/aspects/words.
- I used NLTK in the file below to perform
    - stopword removal
    - tokenization
    - stemming

- But before using NLTK, it's necessary to download the required resources such as tokenizers, stopwords, and stemmers.
- In this code, I download the punkt tokenizer and the stopwords corpus.

- **JUPYTERLAB:**
    - It is versatile platform on which one can carry machine learning tasks.
    - It enables step-by-step development which is beneficial for xploring datasets and visualisations.
    - Its interactivity is an advantage, it gives real-time feedback allowing one to debug their code immediately.
- **Scikit-learn**
    - Matplotlib is a python library that is used for creating visualisations of data.
    - It is a perfect tool for one to create plots and charts.
    - Will use matplotlib to create visualisations of sentiment analysis results.

- Will use matplotlib to map out distributions of positive, negative and neutral sentiments on a chart.

- **FLASK**

3.2 Other software

- **Git :**
  - Used for repository
- **Flask:**
  - Flask is a python framework to serve static and html templates.
  - Flask allowed me to receive user reviews through the request.form[] method and then use the saved model to make a prediction.
  - Then the prediction is used to filter movies out that has similar attributes.

3.3 Hardware

N/A

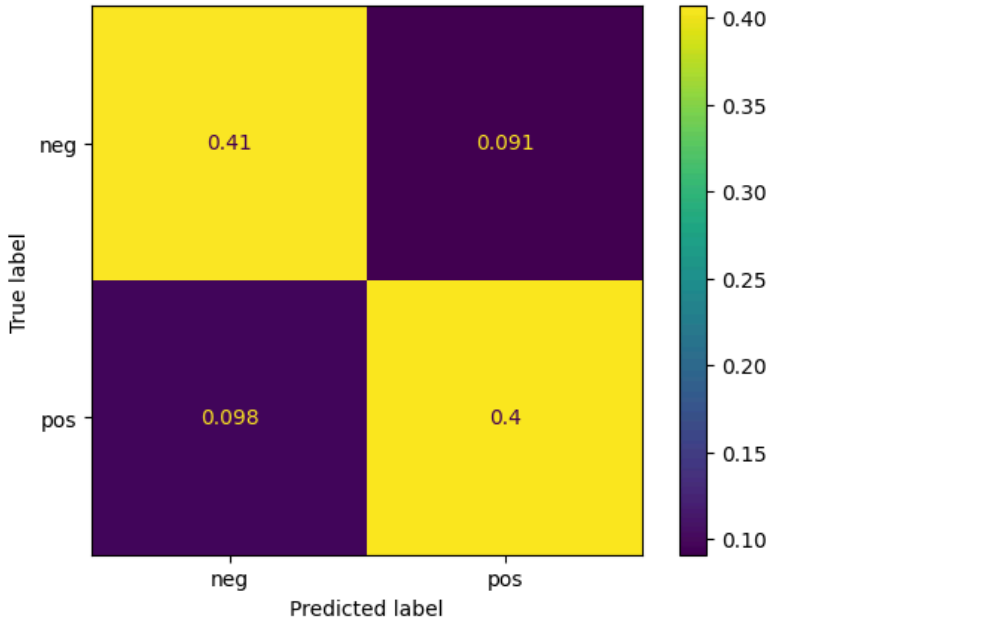
4. Design Documents

4.1. Confusion Matrix

- I utilise the confusion matrix below to gauge the SVM model's performance.
- Knowing true trues and false trues and so forth helps optimise the SVM model in terms of its hyperparameter setting.

- Now that SVM seems to perform the best among all, we take this as our base model and further fine-tune its hyperparameter using cross-validation and Grid Search.

```
: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b4794b7fe10>
```



5. Methodology

- In this section I will explain each file I have implemented in completion of this project, detailing the crucial functions, member functions and classes.

Preprocessing in the DATA directory

5.1. data_collection.py module

File Overview

- This Python script scrapes data from IMDb website for movie reviews and stores it in a CSV file.

Classes

Data Class

- This class **encapsulates** methods for initializing a WebDriver, scraping IMDb data, and converting the gathered data to a CSV format.

Attributes of Data Class

- **driver** is the WebDriver object for controlling the web browser.
- **url** is the URL of the IMDb page being scraped.
- **soup** is the BeautifulSoup object for parsing HTML content.
- **collected_data** is the List data structure to store the scraped data.

Methods of Data Class

- initialize_driver() is the method which initializes the Chrome WebDriver with some options and stealth configuration.
- scrape_imdb(url) method implements the IMDb scraping logic, collects data such as movie titles and user reviews from IMDb pages.
- to_csv(data_structure, filename) this is a function I implemented that converts the gathered data to CSV format and writes it to a file.

Functions

execute_data_gathering

- Orchestrates the IMDb data gathering and CSV conversion process.
- My idea behind this, is that the function is called when the script is executed

- It sort of initializes the `Data` object, scrapes data from IMDb, and stores it in a CSV file.
- It goes something like this:
    - Initializes the scraper_object as an instance of the Data class.
    - Calls the initialize_driver() to set up the WebDriver.
    - Invokes the scrape_imdb(IMDB_URL) to scrape data from IMDb.
    - If data is scraped successfully, it calls to_csv(scraped_data, CSV_FILENAME) to convert and store data in a CSV file.
    - Another thing is that it handles exceptions and prints error messages if any.
    - All the CSV files are saved within the data directory.

5.2. data_cleaning.ipynb notebook

File Overview

- This Python notebook is for processing scraped data from the saved CSV file containing movie reviews and performing cleaning operations using the pandas library.

Functions and Operations

- Loads the scraped data set from the CSV file by using the **read_csv()** function from the pandas library to read the CSV file into a DataFrame in order to perform analysis and processing.
- I use the Description utilities to display general information about the loaded DataFrame particularly the **info()** method of the DataFrame object.
- I orchestrated this to gain an overview of the DataFrame, including column names, data types, and missing values.

- The retrieved information coupled with the **drop_duplicates()** method of the DataFrame object removes the duplicate data from our reviews set and ensures data integrity and eliminates redundancy in the DataFrame.
- The **describe()** method of the DataFrame object provides a summary statistics of the DataFrame after removing duplicates so that we can examine the distribution of numerical data and identify any outliers or patterns.
- I check if the number of rows in the **User Reviews** column matches the number of rows in the **Title** column by deploying the **info()** method of the DataFrame object to ensure data consistency and integrity before further processing.
- I convert the string representation of user reviews to Python lists by deploying the apply() function with ast.literal_eval and explode() methods to achieve aspect-level design and facilitate analysis of individual user reviews.
- The analysed reviews are saved as CSV files as well.

5.3. data_tokenization.ipynb notebook

File Overview

- In this Python notebook I process scraped data from a CSV file containing movie reviews and performs cleaning operations using the pandas library and displays general information about the loaded DataFrame.
- Some steps include:
    - removing duplicate rows from the DataFrame based on the **Title** column.
    - generating summary statistics of the DataFrame after removing duplicates.
    - verifying if the number of rows in the 'User Reviews' column matches the number of rows in the Title column.

- then converts the string representation of user reviews to Python lists.

- Eventually we save the analyzed reviews as CSV files for visualization and PICKLE for future reading.

5.4. data_stemming.ipynb notebook

File Overview

- In this notebook I aim to preprocess user reviews by stemming the words using the NLTK PorterStemmer, and then saving the preprocessed data into pickle and CSV formats for reading data and visualizing it respectively.

Functions

- **stem_words(input_sentence_list):**
  - This function stems each word in the input list of sentences using the PorterStemmer library from NLTK.
  - It iterates through each word in the input list and applies stemming using the PorterStemmer.
  - This function returns a list of stemmed words.

- **execute_stemming():**
  - In this function I orchestrate the stemming process where by I read the user reviews data from a pickle file (data_set_v3.pkl).
  - I apply the stemming to each sentence using the stem_words function.
  - Then I construct a DataFrame with the stemmed user reviews.

- Eventually saving the DataFrame containing stemmed data into a pickle file (data_set_v4.pkl) to maintain the list structure

Workflow

So the basic workflow is that I start of by

- Importing the necessary libraries such as nltk.stem.PorterStemmer, pandas.
- Defining the stem_words function to stem words in a list of sentences.
- Reading the user reviews data from the pickle file (data_set_v3.pkl) into a DataFrame.
- Applying the stemming to each sentence in the **User Review**s column using the stem_words function.
- Constructing a new DataFrame with the stemmed user reviews.
- Then saving the DataFrame containing stemmed data into a pickle file (data_set_v4.pkl) to maintain the list structure.
- And eventually saving the DataFrame as a CSV file (data_set_v4.csv) for visualization, excluding the index column.

5.5. data_regexp.ipynb notebook

Overview

In this Python script we filter out special characters from the stemmed user reviews and save the processed data into pickle and CSV formats.

Functions

filter_special_chars(review)

- This function removes special characters from a given review.
- We use regular expressions to substitute any non-alphanumeric characters with an empty string.
- Then it returns the review with special characters removed.

Workflow

- So I import necessary libraries - pandas, re.
- Then I read the DataFrame containing stemmed user reviews from the pickle file (data_set_v4.pkl).
- Afterwards I define the filter_special_chars function to remove special characters from a review.
- Applying the filter_special_chars function to each review in the **Stemmed User Reviews** column of the DataFrame.
- Then I construct a Series with the filtered stemmed user reviews.

Training in the MODELS directory

5.6. training.ipynb notebook

Overview

- The training.py script is THE comprehensive tool for sentiment analysis on movie reviews in this project.
- In this file I aim to determine the sentiment or opinion expressed in the processed reviews, which can be positive, negative, or neutral.
- This script specifically focuses on classifying movie reviews as positive or negative based on the sentiment they convey.
- This notebook leverages machine learning techniques, particularly Support Vector Machine (SVM) classification, to analyze and classify our movie reviews.
- SVM is a powerful supervised learning algorithm used for classification tasks, which works by finding the hyperplane that best separates the classes in the feature space.
- I chose SVM as it performed better than the other models such as Naive Bayes.
- The workflow of the script involves several key steps, for example data preprocessing, text vectorization, model training, evaluation, hyperparameter tuning, and post-hoc analysis.
- These step plays very crucial role in the overall sentiment analysis process, ensuring accurate classification of movie reviews.

Data Preprocessing

- Data preprocessing is a fundamental step in machine learning.
- And it involves cleaning and transforming raw data into a format suitable for analysis and modeling.

- In this context of sentiment analysis on movie reviews, data preprocessing typically includes tasks such as loading the dataset, removing noise, and preparing the data for feature extraction.

Loading the Dataset

- I start off by loading the movie_reviews corpus provided by the NLTK library.
- The movie_reviews corpus contains a collection of movie reviews.

```
import nltk

from nltk.corpus import movie_reviews



# Download movie_reviews corpus using NLTK

nltk.download('movie_reviews')



# Load movie reviews dataset and split into training and testing sets

documents = [(' '.join(movie_reviews.words(fileid)), category)
```

```
        for category in movie_reviews.categories()

        for fileid in movie_reviews.fileids(category)]

train, test = train_test_split(documents, test_size=0.22, random_state=42)

X_train = [text for (text, label) in train]

X_test = [text for (text, label) in test]

y_train = [label for (text, label) in train]

y_test = [label for (text, label) in test]
```

- In this step, I load the movie_reviews corpus, and then each movie review is paired with its corresponding sentiment label either positive or negative.
- I split the dataset into training and testing sets using the **train_test_split()** function from the **sklearn.model_selection** module.
- This ensures that the model is trained on a portion of the data and evaluated on a separate portion.

Exploratory Data Analysis

- Exploratory Data Analysis (EDA) involves examining the dataset to gain insight into the structure, characteristics, and potential issues of the data.
- Some common tasks in EDA include checking for missing values, exploring the distribution of labels, and analyzing the length of text samples.

```
# Exploratory Data Analysis

print("Number of training examples:", len(X_train))

print("Number of testing examples:", len(X_test))

print("Sample review:", X_train[0])

print("Sample label:", y_train[0])
```

- In the above code, I print out some basic statistics about the dataset, such as the number of training and testing examples.
- I also display a sample review and its corresponding label to get a sense of the data's structure.

Text Preprocessing

- Text preprocessing is a crucial step in preparing textual data for analysis.

- We clean the text, removing noise, and standardizing the format of the text samples
- Common text preprocessing techniques include tokenization, lowercasing, removing punctuation, and handling stopwords.

```
import re



# Preprocessing function to clean text

def preprocess_text(text):

    # Remove special characters and punctuation

    text = re.sub(r'[^\w\s]', '', text)

    # Convert text to lowercase
```

```
    text = text.lower()

    return text


# Apply preprocessing to training and testing data

X_train = [preprocess_text(text) for text in X_train]

X_test = [preprocess_text(text) for text in X_test]
```

- In this code I define a preprocessing function **preprocess_text** to clean the text by removing special characters and punctuation then converting the text it to lowercase.
- I then apply this preprocessing function to both the training and testing data to ensure consistency in the text format.

Text Vectorization

- In this process I convert the textual data into **numerical vectors** that can be understood by machine learning algorithms.
- In this context, **text vectorization** transforms the raw text of movie reviews into a format that can be used as input for training a machine learning model.

TF-IDF Vectorization

- Term Frequency-Inverse Document Frequency or simply TFIDF is a technique for text vectorization in natural language processing.
- It assigns **weights** to words in a document based on their **frequency and rarity** across all documents in the corpus.
- Words which appear frequently in a document but **rarely** in the corpus are considered more important and receive higher weights.

```
from sklearn.feature_extraction.text import TfidfVectorizer



# Initialize TF-IDF vectorizer with specified parameters

tfidf_vec = TfidfVectorizer(min_df=10, token_pattern=r'[a-zA-Z]+')



# Perform TF-IDF vectorization on the training set

X_train_tfidf = tfidf_vec.fit_transform(X_train)



# Perform TF-IDF vectorization on the testing set

X_test_tfidf = tfidf_vec.transform(X_test)
```

- In my code above, I initialize a **TFIDF** vectorizer with specified parameters such as **min_df** (minimum document frequency) and token_pattern (pattern for tokenization).
- Which I use the fit_transform method to perform **TFIDF** vectorization on the training set and then **transform** method to transform the testing set using the same vectorizer.

Model Training and Selection

- Model training involves fitting a machine learning algorithm to the preprocessed data, allowing it to learn patterns that can later be used to make predictions.
- In this project I have tried other models, but I will use a Support Vector Machine (SVM) model for sentiment analysis.

Selection of Machine Learning Algorithm

- Support Vector Machine (SVM) is a powerful and versatile machine learning model and very well-suited for classification tasks.
- It works by finding the **optimal hyperplane** that best separates the data into different classes.

```
from sklearn.svm import SVC



# Initialize the SVM model with specific parameters

model_svm = SVC(C=8.0, kernel='linear')



# Fit the model on the vectorized training data

model_svm.fit(X_train_tfidf.toarray(), y_train)
```

- In the snippet above I initialize an SVM with a linear kernel and a **regularization** parameter C of 8.0.
- The fit () method is used to train the model on the training data, where X_train_tfidf.toarray() converts the sparse matrix to a dense array suitable for training with SVM in scikit-learn.

Hyperparameter Tuning

- Hyperparameter tuning according to wikipedia is the process of selecting the set of optimal hyperparameters for a learning algorithm.
- So in scikit-learn algorithm a very common approach is to use Grid Search:

```
from sklearn.model_selection import GridSearchCV



parameters = {'kernel': ('linear', 'rbf'), 'C': (1, 4, 8, 16, 32)}

svc = SVC()

clf = GridSearchCV(svc, parameters, cv=10, n_jobs=-1)  # `-1` means using

all processors



# Fit the GridSearchCV object to find the best model

clf.fit(X_train_tfidf.toarray(), y_train)
```

```
# Best model parameters

clf.best_params_
```

- In this Grid Search implementation I evaluate all combinations of the specified parameter values and select the combination that results in the best model performance, measured by cross-validation on the training data.
- Which leads to my next topic

Cross-Validation

- Cross-validation is used to ensure that the model performs well on unseen data and reduces the likelihood of model overfitting.

```
from sklearn.model_selection import cross_val_score



# Perform cross-validation and compute scores

scores = cross_val_score(model_svm, X_train_tfidf, y_train, cv=10)



# Average cross-validation score
```

```
scores.mean()
```

- I split the training dataset into a specified number (cv) of smaller sets, or folds, using each fold as a test set while training the model on the remaining folds.
- We then repeat this process until each fold has been used as the test set.
- And the performance metric for each fold is recorded and an average score is computed to provide an overall performance estimate.

Model Evaluation

- After training and tuning the model it is crucial to evaluate its performance on the test set to assess how well it generalizes to new, unseen data.

```
from sklearn.metrics import accuracy_score, f1_score, ConfusionMatrixDisplay



# Predict the labels for the test set

y_pred = model_svm.predict(X_test_tfidf.toarray())
```

```
# Calculate accuracy and F1 score

accuracy = accuracy_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred, average=None,

labels=movie_reviews.categories())



# Display the confusion matrix

ConfusionMatrixDisplay.from_estimator(model_svm, X_test_tfidf.toarray(),

y_test, normalize='all')
```

- Accuracy and F1 score are common metrics for evaluating classification models.
- The confusion matrix provides a visual representation of the model's performance across different classes.

Model Deployment and Evaluation

- After training and selecting the best model the next step would be to deploy it for real-world use and evaluate its performance on unseen data.
- Well in this stage is crucial for understanding how the model will perform in operational settings and ensuring its reliability and accuracy.

Model Deployment

- In the deployment phase it involves integrating the trained model into the existing production environment where it can receive new inputs and provide predictions.
- In this process it may involve setting up a web service, a cloud deployment, or an on-premise solution.

```
import pickle


# Save the trained model to a file
with open('./model_svm.pkl', 'wb') as f:
    pickle.dump(model_svm, f)


# Save the entire pipeline
with open('./pipeline.pkl', 'wb') as f:
    pickle.dump(pipeline, f)
```

- I use Python's pickle module to serialize the model and pipeline into a file that can be loaded later to make predictions without retraining.

- **Serialization** which is the process of converting an object into a stream of bytes to save it to a file from which it can be restored later.

Making Predictions

- Once deployed, the model can be used to make predictions on new data.
- This is typically done by processing the new data using the same transformations as the training data, then applying the model to make predictions.

```
new_reviews = [
    'This movie was an excellent portrayal of a historical event.',
    'The plot was dull and the characters were not convincing.'
]

# Transform new data using the pipeline
new_reviews_transformed = pipeline.transform(new_reviews)
```

```
# Predict using the trained model

predictions = model_svm.predict(new_reviews_transformed)
```

- The pipeline in the above code includes steps for text vectorization then the trained SVM model.
- The **predict()** method is used on the transformed reviews to generate predictions about their sentiments.

Performance Evaluation

- Evaluating the model's performance on a test set or new data helps to validate its effectiveness.

```
# Evaluate model performance using test data

test_accuracy = model_svm.score(X_test_tfidf.toarray(), y_test)

print(f'Test Accuracy: {test_accuracy}')



from sklearn.metrics import f1_score

f1 = f1_score(y_test, predictions, average='weighted')

print(f'F1 Score: {f1}')
```

- The **score()** method provides a way to measure the accuracy of the model and the **F1 score** provides a balance between precision and recall which are useful in cases where classes are imbalanced.

Advanced Model Interpretation

- In the next step understanding why the model makes certain predictions can be crucial, especially for complex models. Tools like LIME (Local Interpretable Model-agnostic Explanations) can help.

```
from lime.lime_text import LimeTextExplainer



# Initialize LIME explainer

explainer = LimeTextExplainer(class_names=['negative',

'positive'])



# Explain a prediction

idx = 0  # Index of the review to explain
```

```
explanation = explainer.explain_instance(new_reviews[idx],
pipeline.predict_proba, num_features=10)


# Display the explanation
explanation.show_in_notebook(text=True)
```

- What LIME does is it provides explanations for individual predictions which is especially useful for understanding the contributions of many features to the decision made by the model.

Serving The App in the Root directory

5.7. main.py

- After pipelining the trained model, I use the model to make predictions on the user input reviews using python's flask framework to serve and host the model.

```
import json
import pickle
import requests
from flask_wtf import FlaskForm
```

```
from flask_session import Session
from wtforms.validators import DataRequired
from wtforms import TextAreaField, SubmitField
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from flask import Flask, render_template, request,redirect, url_for,
session, flash



app = Flask(__name__, template_folder='./src/templates/',
            static_folder='./src/static/')



# Configurations
app.config['SECRET_KEY'] = 'sickrat'
app.config['SESSION_TYPE'] = 'filesystem'

Session(app)

# Load the SVM model
with open('./models/pipeline.pkl', 'rb') as model_svm:
    pipeline = pickle.load(model_svm)

# Load movie data from API
def load_movies(query):
    page_number = 10
    results = []
    all_results = []
```

```
    for page in range(1, page_number + 1):
        url =
f"http://www.omdbapi.com/?apikey=e308b9e6&s={query}&page={page}"
        response = requests.get(url)
        data = response.json()
        if 'Search' in data:
            results.extend(data['Search'])
        else:
            break

    # Fetch plot for each movie
    for movie in results:
        plot_url =
f"http://www.omdbapi.com/?apikey=e308b9e6&i={movie['imdbID']}&plot=f
ull"
        plot_response = requests.get(plot_url)
        plot_data = plot_response.json()
        if 'Plot' in plot_data:
            all_results.append(plot_data)

    return all_results



def load_movie(title):
    url = f"http://www.omdbapi.com/?apikey=e308b9e6&t={title}"
    response = requests.get(url)
    if response:
```

```
        return [response.json()]
    else:
        return []


# Form class
class ReviewForm(FlaskForm):
    review = TextAreaField('Review', validators=[DataRequired()],
render_kw={
                           "placeholder": "Enter your review here"})
    submit = SubmitField('Submit')



# Content based filtering
class ContentBasedFilter:
    def __init__(self, movies):
        self.movies = movies
        self.features = []
        self.imdb_ids = []
        self.vectorizer = TfidfVectorizer()
        self.similarities = None
        self._extract_features()
        self._calculate_similarity()


    def _extract_features(self):
        for movie in self.movies:
            feature = movie["Genre"] + " " + movie["Plot"] + " " +
movie["Director"] + " " + movie["Actors"]
            self.features.append(feature)
```

```
            self.imdb_ids.append(movie["imdbID"])


    def _calculate_similarity(self):
        X = self.vectorizer.fit_transform(self.features)
        self.similarities = cosine_similarity(X, X)


    def recommend_similar_movies(self, imdb_id):
        movie_index = self.imdb_ids.index(imdb_id)
        similarity_scores =
list(enumerate(self.similarities[movie_index]))
        similarity_scores.sort(key=lambda x: x[1], reverse=True)
        similarity_scores = similarity_scores[1:]
        similar_movie_indices = [score[0] for score in
similarity_scores[:10]]

        all_results = [self.movies[index] for index in
similar_movie_indices]

        return all_results




@app.route('/', methods=['POST', 'GET'])
def index():
    movies = []
    form = ReviewForm()
    title = 'Review and Recommend'
```

```
    if request.method == 'POST' and form.validate_on_submit():
        movie_id = request.form['movie_id']
        review = form.review.data
        prediction = pipeline.predict([review])
        if prediction == ['pos']:
            cbf = ContentBasedFilter(load_movies('all'))
            recommended_movies =
cbf.recommend_similar_movies(movie_id)
            session['recommended_movies'] = recommended_movies
            flash('Thanks for the positive review, here are your
movie recommendations!', 'info')
            return redirect(url_for('recommended_movies_page'))

        else:
            flash('Sorry to hear about that, you can keep on
reviewing.', 'danger')
            return redirect(url_for('index'))

    genre = request.args.get('genre')
    search = request.args.get('search')
    if genre:
        movies = load_movies(genre)
        title = genre.capitalize()
    elif search:
        movies = load_movie(search)
        title = search.capitalize()
    else:
        movies = load_movies('all')
```

```
    return render_template('index.html', form=form, movies=movies,
title=title)


@app.route('/recommended', methods=['POST','GET'])
def recommended_movies_page():
    recommended_movies = session.get('recommended_movies')
    title = 'Recommended Movies'
    return render_template('recommended.html',
movies=recommended_movies, title=title)


if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

Dependencies

- **json** - Module for handling JSON data.
- **pickle** - Module for serializing and deserializing Python objects.
- **requests** -Module for sending HTTP requests.
- **Flask** - Micro web framework for Python.
- **Flask-WTF** - Flask extension for handling web forms.
- **Flask-Session -** Flask extension for managing user sessions.
- **wtforms.validators -** Module for form validation.
- **wtforms -** Library for creating web forms.
- **sklearn.metrics.pairwise** - Module for computing cosine similarity.
- **sklearn.feature_extraction.text** - Module for text vectorization.

- **Flask -** Web framework for building web applications.
- **render_template** - Function for rendering HTML templates.
- **request** - Object for handling HTTP requests.
- **redirect** - Function for redirecting to a different URL.
- **url_for** - Function for generating URLs
- **session -** Object for managing user sessions.
- **flash -** Function for displaying flash messages.

Utility Functions

load_movies(query):

- This function fetches movie data from the OMDB API based on users search query.
- It then returns a list of movie details including the plot of the movie saved in as list of json objects.

load_movie(title):

- The **load_movie** fetches details of a single movie based on its title from the OMDB API.
- And then returns movie details as a list.

Routes

index()

- This is the main route for the home page.
- Which handles form submission for movie reviews and then predicts sentiment of the review using the saved SVM model.
- In turn it recommends similar movies based on the sentiment whether positive or negative and renders the home page template with movie data and form.

recommended_movies_page():

- Its the route for the recommended movies page obtained as part of a session from the **index** route.
- Then retrieves recommended movies from the session.
- And then renders the recommended movies template.

Content Based Filtering

ContentBasedFilter

- Is a class for content-based filtering that recommends similar movies based on movie features such as genre, plot, director, and actors.

Methods and Attributes:

__init__(movies)

- This constructor method initializes the ContentBasedFilter class with movie data received from **load_movies()**.
- The constructor accepts a list of movie details as input.
- And then extracts relevant features from each movie such as genre, plot, director, and actors.
- We then calculate the **similarity matrix** between movies based on extracted features.

_extract_features()

- Is a private method that extracts features from movie data.
- It combines the genre, plot, director, and actors information into a single feature vector for each movie.

_calculate_similarity()

- In this private method I calculate the similarity matrix between movies.
- Uses TF-IDF vectorization to transform feature vectors into numerical representations.
- And then calculates cosine similarity between feature vectors to measure similarity between movies.

recommend_similar_movies(imdb_id)

- This is a public method that recommends similar movies based on a given movie ID.
- This accepts the IMDb ID of a movie as input.
- Then finds the index of the movie in the similarity matrix and sorts the similarity scores and returns the top similar movies excluding the input movie.
- Which returns a list of recommended movies based on similarity scores.

movies:

- The movies attribute is a list containing details of all movies used for content-based filtering.
- Each movie is represented as a dictionary with keys such as Title, Genre, Plot, Director, Actors, etc.

6. Conclusion

6.1. Benefits

a. Benefits to users

1. **Context based recommendations** - Users get to watch movies that are specific to their sentimental decisions. Each watch is tailored to the user's sentiments and thus enhancing user retainment and engagement.

2. **Recommendations tailored towards current mood** – Movies can be aligned with the user's current mood. If a user is in a happy state - happy-themed movies are then recommended. Users' emotional responses will mould their recommendations enabling the system to suggest movies based on the user sentiments.

3. **Personal recommendations –** The system understands the user's emotional response and recommend movies that align with that. The movie catalogue is then tailored towards a personalised taste for the user.

4. **Discovery of new content –** System introduces the user to movies they might have not found otherwise but falls within their preference. Analysing reviews will lead to discovery of other content that aligns with the user's sentiments.

5. **Reduced decision making –** reduced decision making and scrolling to find movies and therefore time saving for the user. Users can focus on a concentrated set of movies/options.

6. **Adaptive recommender** systems for users which enhances user retention, by automatically knowing what the user prefers and in what case they prefer it.

7. **Increased satisfaction** when users receive recommendations that align with their preferences and sentiments, they are more likely to enjoy the recommended movies.

b. Benefits to me

1. Well, this project is a bit complex and hence, I gain more experience in planning, requirements engineering, data analysis and model training.

2. Artificial Intelligence based software is the next era and hence starting now, would add a beneficial skill to my diverse skillset.

3. This might help with future software projects I might work on that require recommender systems to be powered by user sentiments.

4. With this project I explore various NLP techniques, models and algorithms, it is a massive learning opportunity to diversify skills.

6.2. Ethics

- **Privacy** – Any data the user shares or uses is kept as such – private. This data is going to be handled in accordance with privacy regulations and will not be used outside of the terms of service.
- **Consent** - Clear statement of how the user's data is to be used in this recommender system and no data is used that is not consented or legal. Clearly communicate how user's data is to be handled and used.

• **Biases and Fairness** – Removal of bias from the dataset, this enhances the integrity of the input leading to fair recommendations.

• **Openness** – Transparency to standards that are used to determine the results of this analysis. Clear communication of the criteria used to influence the recommendations. Users ought to be well informed.

• **Inclusive Usage** – This recommender system ought to be diverse and usable with anyone and anywhere. The system ought to accommodate users from different demographics and cultures.

• **Security** – User data ought to be protected within a robust setting. Protection from spam ware and phishing. Recommender systems often rely on user preferences and user profiles hence its necessary t o safeguard that information.

Why did I choose this project?

- There are several learning benefits and opportunities in picking sentiment analysis as graduation project such as;

1. Sentiment analysis is a very relevant field of NLP and hence there loads of previous findings and research pertaining to this field.

2. Its popularity ensures that there are a lot of resources such as libraries and datasets to build upon.

3. It can be applied to other fields that use recommender systems and dependence is upon user feedback.

4. Diversity in interdisciplinary subjects such as statistics, linguistics and computer science henceforth attaining a pretty good-looking skillset.

5. Contribution to decision making in the movie industry, by means of user feedback.

6. Innovative software design as Sentiment analysis would improve the overall feel of applications by providing personalised suggestions.

- EXPERIENCE, EXPERIENCE...can't stress this much but sentiment analysis in this movie domain is a very complex field which will give me a hands-on software development practice.

6.3. Future Works

I am not going to keep working on this project, but I plan to be a part of the Artificial Intelligence field in the future.

7. References

[1] : Sentiment classification using machine learning techniques (2018) Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan retrieved from https://www.cs.cornell.edu/home/llee/papers/sentiment.home.html

[2] : Extracting Lexically Divergent Paraphrases from Twitter by Wei Xu. Retrieved from https://dblp.org/pid/32/1213-4.html.

[3] : Feature selection in machine learning: A new perspective by Cai Jie, Luo Jiawei, Wang Shulin, Yang Sheng.

Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0925231218302911?fr=RR-2&ref=pdf_download&rr=83b0e8b219245a99

[4] Features extraction retrieved from https://deepai.org/machine-learning-glossary-and-terms/feature-extraction#:~:text=What%20is%20Feature%20Extraction%3F,losing%20important%20or%20relevant%20information.

[5] Sentiment analysis and subjectivity by Bi Liu(2010). Retrieved from https://www.researchgate.net/publication/228667268_Sentiment_analysis_and_subjectivity.

[6] Detection of Social Network Spam Based on Improved Extreme Learning Machine by Zhijie Zhang, Rui Hou, Jin Yang. Retrieved from https://www.researchgate.net/publication/342226344_Detection_of_Social_Network_Spam_Based_on_Improved_Extreme_Learning_Machine?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19

[7] Sentiment Analysis using bag of words retrieved from https://medium.com/swlh/sentiment-analysis-with-bag-of-words-4b9786e967ca

[8] Sentiment Analysis life cycle from spanish to english retrieved from https://www.researchgate.net/publication/309149383_A_Comparison_Between_Two_Spanish_Sentiment_Lexicons_in_the_Twitter_Sentiment_Analysis_Task