INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY

ABERDEEN

Deep Learning - CM3604

Coursework Report

Module Leader - Mr. Prasan Yapa

<u>Group Members</u>

Anuttara Rajasinghe - 20210216 - 2117946

Amandi Wijesuriya - 20210983 - 2118688

Jathusharini Basker - 20210452 - 2117530

Table of Contents

GIT Repository link - https://github.com/AnuttaraR/deep-learning-coursework

1. Introduction

In recent times, deep learning techniques have been rapidly growing, especially in understanding language. This report shares our journey in a university group project where we focused on analyzing the meaning of Yelp reviews.

Yelp, a place known for user reviews, gives us a diverse dataset to explore sentiments and meaning in what users write. By using Hugging Face models, known for their effectiveness in understanding language, our project aims to find patterns and sentiments hidden in the words of Yelp reviews. This report gives a thorough overview of how we went about it—covering steps like preparing the data, choosing models, training them, and measuring their success

2. Sentiment Analysis with bert- based-uncased

Bert - based - uncased model is a neural network language model that uses BERT(Bidirectional Encoder Representations from Transformers) architecture. Tasks involving natural language processing are the focus of BERT. "Uncased" denotes that there is no distinction made by the model between lowercase and uppercase letters. It bidirectionally records contextualized word representations improving comprehension in a range of language-related tasks, including named entity recognition, sentiment analysis, and question answering.

In sentiment analysis, the Bert-based- uncased model proves that it is effective due to its ability to capture contextualized word representations. It understands the specifics of the language, considering the order and the meaning of words in a sentence. Due to this reason we are using bert-based- model for our use case.

a. Corpus preparation

After the essential libraries are loaded, the first dataset to be loaded is the review dataset from the yelp academic dataset. The Yelp dataset is a comprehensive collection of data from the Yelp platform, a popular online review platform for businesses. In this model we are using the two datasets for review and business information, these are stored in json format.

When we first load the review dataset we can see that there are several rows present.

	review_id	user_id	business_id	stars	useful	funny	cool	text	date
0	KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3	0	0	0	If you decide to eat here, just be aware it is...	2018-07-07 22:09:11
1	BiTunyQ73aT9WBnpR9DZGw	OyoGAe7OKpv6SyGZT5g77Q	7ATYjTIgM3jUlt4UM3IypQ	5	1	0	1	I've taken a lot of spin classes over the year...	2012-01-03 15:28:18
2	saUsX_uimxRlCVr67Z4Jig	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	3	0	0	0	Family diner. Had the buffet. Eclectic assortm...	2014-02-05 20:30:30
3	AqPFMleE6RsU23_auESxiA	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	5	1	0	1	Wow! Yummy, different, delicious. Our favo...	2015-01-04 00:01:03

Initial Review Dataset

Upon initial review, the length of the dataset is 10000.

We can see that some columns of data won't be necessary for our model so we will remove them.

	business_id	stars	text
0	XQfwVwDr-v0ZS3_CbbE5Xw	3	If you decide to eat here, just be aware it is...
1	7ATYjTIgM3jUlt4UM3IypQ	5	I've taken a lot of spin classes over the year...
2	YjUWPpI6HXG530lwP-fb2A	3	Family diner. Had the buffet. Eclectic assortm...
3	kxX2SOes4o-D3ZQBkiMRfA	5	Wow! Yummy, different, delicious. Our favo...
4	e4Vwtrqf-wpJfwesgvdgxQ	4	Cute interior and owner (?) gave us tour of up...

Review Dataset after columns dropped

After loading the business dataset, we see that it contains these rows of data also with columns we don't need:

	business_id	name	address	city	state	postal_code	latitude	longitude	stars	review_count	is_op
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.426679	-119.711197	5.0	7	
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.551126	-90.335695	3.0	15	
2	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.223236	-110.880452	3.5	22	

Initial business dataset

Then we filter out rows with null 'categories' and create a subset, df_business_new, containing only businesses labelled as restaurants. The resulting DataFrame is refined to include only 'business_id' and 'categories,'

	business_id	categories
3	MTSW4McQd7CbVtyjqoe9mw	Restaurants, Food, Bubble Tea, Coffee & Tea, B...
5	CF33F8-E6oudUQ46HnavjQ	Burgers, Fast Food, Sandwiches, Food, Ice Crea...
8	k0hlBqXX-Bt0vf1op7Jr1w	Pubs, Restaurants, Italian, Bars, American (Tr...

Business labeled as restaurants

Then we move on to merging the two dataframes, df_review and df_business_new, using an inner join operation based on the common 'business_id' column. The resulting dataframe, df_joined, combines information from both DataFrames, linking reviews with corresponding restaurant details.

	business_id	stars	text	categories
0	XQfwVwDr-v0ZS3_CbbE5Xw	3	If you decide to eat here, just be aware it is...	Restaurants, Breakfast & Brunch, Food, Juice B...
1	XQfwVwDr-v0ZS3_CbbE5Xw	2	This is the second time we tried turning point...	Restaurants, Breakfast & Brunch, Food, Juice B...
2	XQfwVwDr-v0ZS3_CbbE5Xw	4	The place is cute and the staff was very frien...	Restaurants, Breakfast & Brunch, Food, Juice B...
3	XQfwVwDr-v0ZS3_CbbE5Xw	3	We came on a Saturday morning after waiting a ...	Restaurants, Breakfast & Brunch, Food, Juice B...
4	kxX2SOes4o-D3ZQBkiMRfA	5	Wow! Yummy, different, delicious. Our favo...	Halal, Pakistani, Restaurants, Indian

Merged dataset

Next, it renames the 'text' column to 'restaurant_reviews' and drops the 'business_id' column. The language of each review in the 'restaurant_reviews' column is detected using the langdetect library. Rows where the detected language is not English ('en') are removed, ensuring analysis is focused on English reviews, and NaN values are checked. Duplicate rows are removed to ensure data integrity, and reviews with a 3-star rating are excluded.

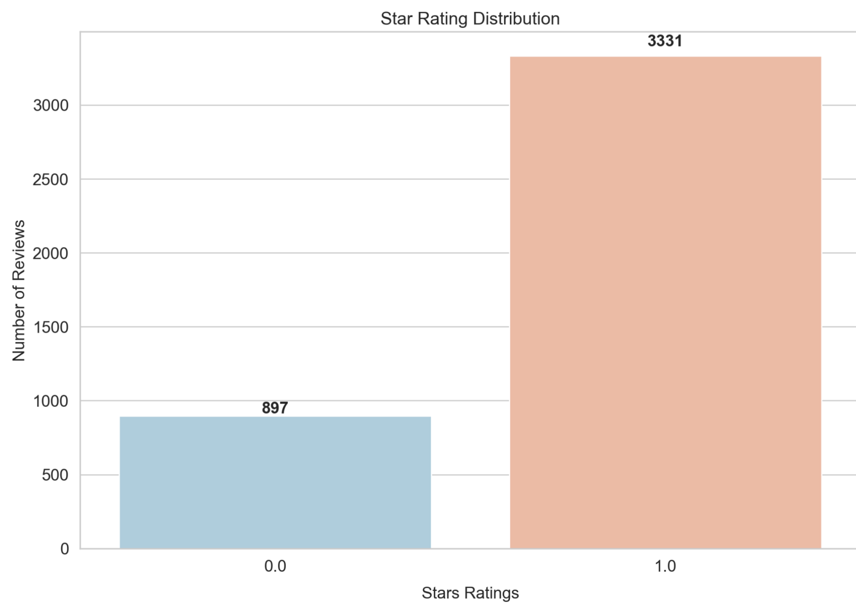
Now our dataset is down to 4228 rows.

Furthermore, sentiment labels are assigned to the DataFrame df_joined based on review ratings. Reviews with a rating of fewer than 3 stars are labelled as 0, indicating a negative sentiment, while those with a rating greater than 3 stars are labelled as 1, denoting a positive sentiment. The 'stars' column is then dropped from the DataFrame, leaving behind a refined dataset where each review is associated with a binary sentiment label.

	restaurant_reviews	categories	detect	sentiment
1	This is the second time we tried turning point...	Restaurants, Breakfast & Brunch, Food, Juice B...	en	0.0
2	The place is cute and the staff was very frien...	Restaurants, Breakfast & Brunch, Food, Juice B...	en	1.0
4	Wow! Yummy, different, delicious. Our favo...	Halal, Pakistani, Restaurants, Indian	en	1.0
5	Dine-in gets 2 stars. Disappointing service & ...	Halal, Pakistani, Restaurants, Indian	en	0.0
6	After a long hiatus from reviewing I have awak...	Halal, Pakistani, Restaurants, Indian	en	1.0

Df with binary sentimental classification

For visualization purposes, we added a bar plot, created with seaborn to display the count of reviews for each sentiment category. The x-axis represents the sentiment labels (0 for negative, 1 for positive), and the y-axis shows the corresponding number of reviews. This provides a visual overview of the sentiment distribution in the dataset.



Star ratings vs number of reviews

This shows that the positive and negative reviews are imbalanced. So to resolve this we had to resample the negative reviews to match the positive reviews.

```
sentiment
0.0    3331
1.0    3331
Name: count, dtype: int64
```

Once that's done, spaCy is utilized to remove stopwords from a restaurant review dataset. The English language model (en_core_web_sm) is loaded, and the default set of spaCy stopwords is retrieved. Specific stopwords, such as 'no' and 'not', are excluded from the set. The text preprocessing function text_preprocessing is defined to clean raw restaurant reviews. This function removes non-alphabetic characters, converts text to lowercase, tokenizes words, and filters out spaCy stopwords.

The cleaning process is applied to the 'restaurant_reviews' column, and the preprocessed reviews are stored in a new 'cleaned_reviews' column in the DataFrame df_clean. The final DataFrame df is then reset with a new index for subsequent analysis on the cleaned reviews.

	restaurant_reviews	categories	detect	sentiment	cleaned_reviews
0	Wow have things changed at this location. I us...	Vegetarian, American (New), Mediterranean, San...	en	0.0	wow things changed location come regular basis...
1	The service was great and the restaurant was c...	Restaurants, Italian, Pizza	en	0.0	service great restaurant clean food okay lingu...
2	Decent, quick serve food. Certainly not gourme...	Mexican, Tacos, Restaurants, Tex-Mex, Fast Foo...	en	0.0	decent quick serve food certainly gourmet taco...
3	Please be careful when you order food to go f...	Restaurants, American (New), Southern, Chinese	en	0.0	careful order food restaurant website updated ...
4	i wish i could have given it a zero or a negat...	Vegetarian, American (New), Mediterranean, San...	en	0.0	wish given zero negative experience pleasant o...

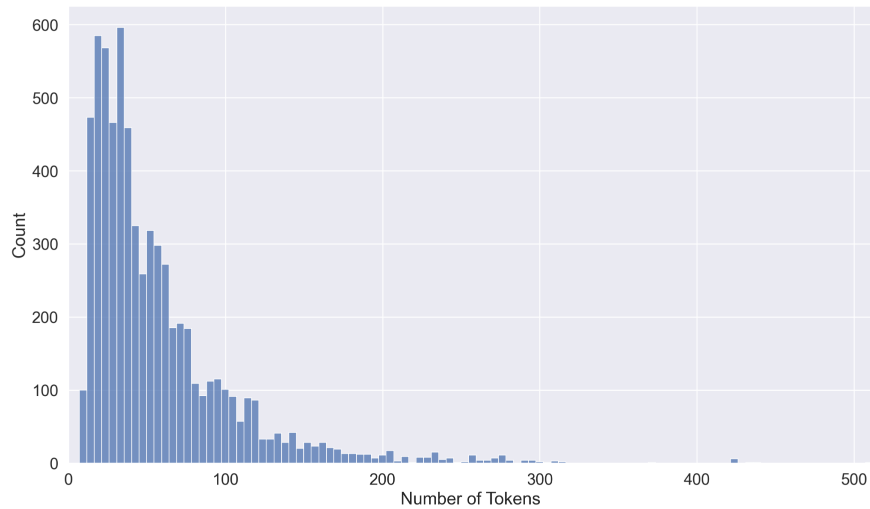
Final cleaned dataset

b. Solution methodology

Now for the model. We utilize the BERT (Bidirectional Encoder Representations from Transformers) tokenizer from the 'bert-base-uncased' pre-trained model to analyze the distribution of token lengths in the cleaned restaurant reviews.

The variable token_lens is created to store the length of tokens for each review. The loop iterates over the cleaned reviews, encoding them with the BERT tokenizer while considering a maximum length of 512 tokens.

To analyze the number of tokens used to craft reviews, it can be visualized using a histogram plot. The x-axis represents the number of tokens, and the y-axis shows the frequency of reviews with a specific token length. The plot is configured to limit the x-axis range to 512 tokens.



The BERT tokenizer is used to tokenize the cleaned restaurant reviews, and the maximum sequence length is set to 260 tokens by our choice because most of the reviews seems to have less than 260 tokens.

We will use 70% of the dataset for training, then 15% each for validation and testing. This ensures that the distribution of observations across different classes in each subset mirrors the original dataset's proportions, maintaining consistency with the initial distribution.

Once the sets are converted to lists, we generate a set of token IDs (input IDs) for each review through the tokenizer. The conversion was done, which translates a string into a series of integer input IDs by utilizing the tokenizer and vocabulary. This process involves tokenizing the string and assigning each token its respective ID. For classification tasks, we added special tokens—[CLS] and [SEP]—to enhance the input sequence's representation.

We ensured uniformity in sequence lengths through both padding and truncating techniques. For padding, zeros were added as needed to extend sequences to the specified length of MAX_SEQ_LENGTH. In cases where sequences surpassed this length, we employed truncation to ensure they conformed to the maximum length constraint

For attention masks, we distinguish genuine tokens from placeholder [PAD] tokens. We then transform all lists containing input IDs, labels, and attention masks into Torch tensors.

We implemented a DataLoader to efficiently handle the loading of our datasets, functioning as an iterator and significantly conserving memory during the training process as opposed to traditional for loops.

Our approach involves leveraging the BertForSequenceClassification model to address our classification task. This model extends the Bert Model transformer by incorporating an additional layer specifically designed for classification purposes.

To optimize our Bert Classifier, we opted for the AdamW optimizer due to its incorporation of gradient bias correction and weight decay. Additionally, we adopted a linear scheduler devoid of warm-up steps. In the fine-tuning process, we considered various settings recommended by the authors:

- Batch size: Choices included 16 and 32.
- Learning rate (Adam): Options were 5e-5, 3e-5, and 2e-5.
- Number of epochs: We deliberated on 2, 3, or 4 epochs.

Ultimately, our selections were a batch size of **16**, a learning rate of **3e-5**, and a duration of **2** epochs, aligning with the given recommendations.

Throughout each epoch, we collect and record the training and validation loss and accuracy metrics. By doing so, we have the ability to visually represent the progress of the training loop using plots or tables.

3. Sentiment Analysis with Linear Support Vector Machine Algorithm

Linear Support Vector Machine algorithm is a classification algorithm that finds a hyperplane to separate different classes in a feature space. In sentiment analysis Linear SVM model is used to classify text sentiments. It takes input features, which are frequently extracted from text data, maps them into a high dimensional space, and then finds the best hyperplane to maximally divide emotion groups.

SVMs are useful for applications like sentiment analysis in review or comments on social media because of this linear separation's ability to distinguish sentiments. The success of the model in sentiment analysis applications to its capacity to handle high - dimensional feature spaces and generalize effectively. These features of the model made us choose the Linear Support Vector Machine algorithm as the second model for our sentiment analysis use case.

a. Corpus preparation

First we checked for the null values, duplicates and missing values. Since there were no null

```
Null Values:
review_id      0
user_id        0
business_id    0
stars          0
useful         0
funny          0
cool           0
text           0
date           0
dtype: int64

Duplicates:
Empty DataFrame
Columns: [review_id, user_id, business_id, stars, useful, funny, cool, text, date]
Index: []

Missing Values:
Empty DataFrame
Columns: [review_id, user_id, business_id, stars, useful, funny, cool, text, date]
Index: []
```

values, duplicates and missing values we had no requirement to do a data cleaning. Therefore we decided to proceed with the dataset.

Once the data cleaning part was completed, We also created a code to convert the JSON data into .csv data format for ease of use, and also decided to use only 100,000 rows of data for efficiency.

The YELP dataset typically have 9 columns but we decided to drop the rest and use only the needed 5 columns.

DL_SVM.ipynb | dataset_analysis.py | json_csv.py | yelp_csv_dataset.csv

```python
import pandas as pd
import json

# Specify the paths
json_file_path = 'yelp_academic_dataset_review.json'
csv_output_path = 'yelp_csv_dataset.csv'

# Open the JSON file and read lines
with open(json_file_path, 'r', encoding='utf-8') as json_file:
    # Read the first 10,000 lines from the JSON file
    json_lines = [json_file.readline() for _ in range(100000)]

# Parse each JSON line and store the data in a list
data = [json.loads(line) for line in json_lines]

# Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(data)

# Save the DataFrame to a CSV file
df.to_csv(csv_output_path, index=False)

# Display the first few rows of the resulting CSV file
print(f"Subset of Yelp dataset saved to '{csv_output_path}':")
print(df.head())

```

	review_id	business_id	user_id	text	stars
0	KU_O5udG6zpxOg-VcAEodg	XQfwVwDr-v0ZS3_CbbE5Xw	mh_-eMZ6K5RLWhZyISBhwA	If you decide to eat here, just be aware it is...	3.0
1	BiTunyQ73aT9WBnpR9DZGw	7ATYjTIgM3jUlt4UM3IypQ	OyoGAe7OKpv6SyGZT5g77Q	I've taken a lot of spin classes over the year...	5.0
2	saUsX_uimxRlCVr67Z4Jig	YjUWPpI6HXG530lwP-fb2A	8g_iMtfSiwikVnbP2etR0A	Family diner. Had the buffet. Eclectic assortm...	3.0
3	AqPFMleE6RsU23_auESxiA	kxX2SOes4o-D3ZQBkiMRfA	_7bHUi9Uuf5__HHc_Q8guQ	Wow! Yummy, different, delicious. Our favo...	5.0
4	Sx8TMOWLNuJBWer-0pcmoA	e4Vwtrqf-wpJfwesgvdgxQ	bcjbaE6dDog4jkNY91ncLQ	Cute interior and owner (?) gave us tour of up...	4.0

Datasets after dropping unnecessary columns

To facilitate NLP, a cleaning function (get_clean_text) was implemented, encompassing lowercase conversion, special character removal, punctuation elimination, and stopword exclusion.

Additionally, a tokenization function (get_words) was defined to extract a list of words from the cleaned text. TF-IDF vectorization was employed using scikit-learn's TfidfVectorizer, transforming the cleaned text into numerical vectors.

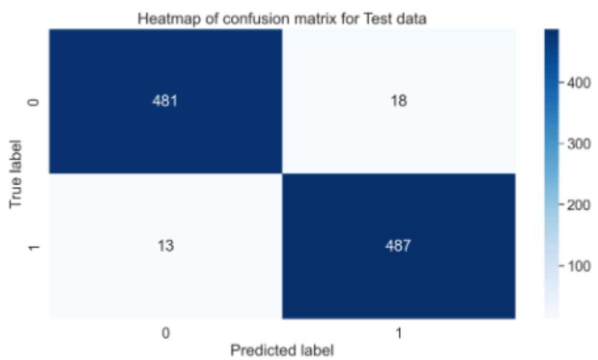
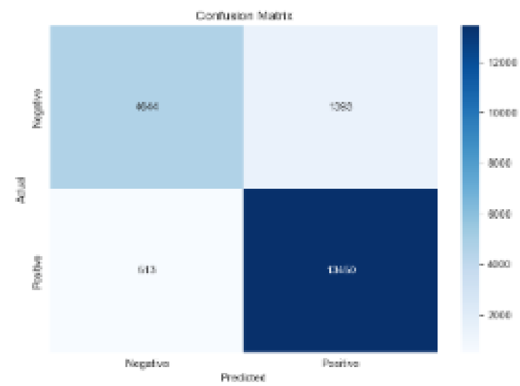
b. Solution methodology

The solution methodology involves first analyzing and preprocessing Yelp review data, concentrating on relevant columns, and using text cleaning methods. Feature engineering includes TF-IDF vectorization of cleaned text for numerical representation. Two approaches were used for the sentiment analysis task: first, a Linear Support Vector Machine (SVM) is used for multi-class sentiment analysis, which predicts exact star ratings; second, a different SVM is used for binary sentiment classification, which converts star ratings into positive or negative sentiments. Metrics like recall, accuracy, precision, and confusion matrices are used to train and assess the models. Key insights, such as star rating distributions and confusion matrices, are visualized to be understood better.

4. Evaluation metrics

The evaluation metrics for the sentiment analysis models include accuracy, precision, recall, and F1 score for multi-class sentiment analysis. These metrics provide a detailed understanding of the models' performance in predicting star ratings, and the ability to correctly classify reviews into various sentiment classes. The confusion matrix visually encapsulates these metrics. In the binary sentiment classification, metrics like accuracy, precision, recall, provide a nuanced evaluation of the model's proficiency in distinguishing positive from negative sentiments. The classification report dissects these metrics further. Visualization of a confusion matrix helps in showing true positives, true negatives, false positives, and false negatives.

5. Evaluation metrics comparison

<table>
<tr><th></th><th>Bert-based-uncased model</th><th>Linear Support Vector Machine</th></tr>
<tr><td>Confusion matrix</td><td>
</td><td>
</td></tr>
<tr><td>Classification report</td><td>
<table>
<tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr>
<tr><td>0</td><td>0.974</td><td>0.964</td><td>0.969</td><td>499</td></tr>
<tr><td>1</td><td>0.964</td><td>0.974</td><td>0.969</td><td>500</td></tr>
<tr><td>accuracy</td><td></td><td></td><td>0.969</td><td>999</td></tr>
<tr><td>macro avg</td><td>0.969</td><td>0.969</td><td>0.969</td><td>999</td></tr>
<tr><td>weighted avg</td><td>0.969</td><td>0.969</td><td>0.969</td><td>999</td></tr>
</table>
</td><td>
Classification Report:
<table>
<tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr>
<tr><td>negative</td><td>0.90</td><td>0.77</td><td>0.83</td><td>6037</td></tr>
<tr><td>positive</td><td>0.91</td><td>0.96</td><td>0.93</td><td>13963</td></tr>
<tr><td>accuracy</td><td></td><td></td><td>0.90</td><td>20000</td></tr>
<tr><td>macro avg</td><td>0.90</td><td>0.87</td><td>0.88</td><td>20000</td></tr>
<tr><td>weighted avg</td><td>0.90</td><td>0.90</td><td>0.90</td><td>20000</td></tr>
</table>
</td></tr>
<tr><td>Accuracy</td><td>accuracy 0.969</td><td>Accuracy: 90.47%</td></tr>
</table>

6. Limitations and future enhancements

a. Limitations

Both Bert-based-uncased and Linear SVM are computationally expensive and require substantial resources for training and inference. The representation and quality of the training data have a major impact on these models' performance. Performance may be impacted by training set biases or constraints.

BERT- based models are deep neural networks, which means that their decisions are difficult to interpret. As such, they are frequently referred to as “black box” models. Although easier to understand, SVMs may have trouble processing data with intricate relationships.
Let’s discuss the limitations of bert-based-uncased model and linear support vector machine algorithm separately.

Limitations of bert-based-uncased algorithm

Due to the size, BERT models require a lot of memory and may not be feasible to implement in settings with limited resources. It can take a lot of effort and time to train BERT models from scratch. Although fine-tuning is common , significant processing power may still be needed.

Limitation of Linear Support Vector Machine algorithms

Assuming a linear decision boundary, linear support vector machines might miss intricate relationships in highly non-linear data. Tasks requiring subtle expressions of sentiment may be difficult for them. SVMs, especially linear ones, frequently rely on single features and might not be as good at capturing contextual dependencies as BERT, which could lead to the loss of subtle linguistic sentiment cues.

b. Future Enhancements

The focus of sentiment analysis's upcoming improvements is on solving current problems and bringing in fresh ideas. The main goal is to reduce model biases and guarantee impartial and fair predictions for a wide range of user demographics. Subsequent investigation focuses on incorporating contextual data to enhance comprehension of complex emotional expressions. The goal of domain adaptation and transfer learning advances is to make models more flexible in response to changing language usage and dynamic content. A more thorough understanding of sentiment in multimedia content is made possible by the investigation of multimodal sentiment analysis, which incorporates both visual and auditory cues. Furthermore, the search for interpretable deep learning models and techniques for real-time sentiment analysis opens the door to the development of sentiment analysis systems that are more precise, flexible, and morally upright.

7. Conclusion

In summary, the BERT-based-uncased model achieved a higher accuracy of 96.9%, outperforming the linear support vector machine model which has an accuracy of 90.47%. The BERT model performs better than other models because it can capture complex language patterns that take word order and context into account. Although the SVM algorithm produced predictions that were quite accurate, BERT produced predictions that were more accurate due to its ability to discern the nuances of sentiment expressions. Depending on the needs of the particular application, we can choose between these models; BERT provides a reliable solution for complex sentiment analysis.

8. Appendix

a. Bert-based-uncased model

```
!pip install transformers langdetect spacy
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import rc
from pylab import rcParams
import matplotlib.pyplot as plt
from textwrap import wrap
from collections import defaultdict
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score,f1_score,precision_score, recall_score
import nltk
import re
import spacy
```

```
from nltk.tokenize import word_tokenize
import transformers
from transformers import BertModel, BertTokenizer, BertForSequenceClassification
from transformers import AdamW, get_linear_schedule_with_warmup
import torch
from torch import nn,optim
from torch.utils.data import Dataset,DataLoader,TensorDataset, RandomSampler,
SequentialSampler
import torch.nn as nn
import torch.nn.functional as F
import time
import datetime
import tensorflow as tf
import sys
import os
import warnings

sp = spacy.load('en_core_web_sm')
nltk.download('punkt')
device=torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

%matplotlib inline
%config InlineBackend.figure_format='retina'

sns.set(style='whitegrid',palette='muted',font_scale=1.2)
color_palette=['#ffb6c1','#add8e6','#98fb98','#fffacd','#dcb4e4','#ffdab9']
sns.set_palette(sns.color_palette(color_palette))
```

```
rcParams['figure.figsize']= 10,5
seed=42
np.random.seed(seed)
torch.manual_seed(seed)

if not sys.warnoptions:
    warnings.simplefilter("ignore")
    os.environ["PYTHONWARNINGS"] = "ignore"
#%%
df_review =
pd.read_json("C:/Users/USER/Downloads/yelp_dataset/yelp_academic_dataset_review
.json", nrows=10000, lines=True)
#%%
df_review.head()
#%%
len(df_review)
#%%
cols_to_drop = ['review_id', 'user_id', 'useful', 'funny', 'cool', 'date']
df_review.drop(cols_to_drop, axis=1, inplace=True)
#%%
df_review.head()
#%%
df_business =
pd.read_json("C:/Users/USER/Downloads/yelp_dataset/yelp_academic_dataset_busine
ss.json", nrows=10000, lines=True)
#%%
```

```
df_business.head()
#%%
df_business = df_business[df_business['categories'].notnull()]
df_business_new = df_business[df_business['categories'].str.contains('Restaurant')]

df_business_new =  df_business_new[['business_id', 'categories']]


#%%
df_business_new.head()
#%%
df_joined = df_review.merge(df_business_new, how='inner', on='business_id')
#%%
df_joined.head()
#%%
# Rename and drop columns
df_joined.rename(columns={'text':'restaurant_reviews'}, inplace=True)
df_joined.drop('business_id', axis=1, inplace=True)


#%%
# Analyze only English reviews
from langdetect import detect

df_joined['detect'] = df_joined['restaurant_reviews'].apply(detect)
df_joined = df_joined[df_joined['detect'] == 'en'].reset_index(drop=True)


#%%
# Check for NaN values
```

```
df_joined.isnull().values.any()

# Remove duplicate rows
df_joined.drop_duplicates(inplace=True)

# Remove 3-star reviews
df_joined = df_joined[df_joined['stars'] != 3]
#%%
df_joined.head()
#%%
len(df_joined)
#%%
# Label reviews as positive or negative
df_joined.loc[df_joined['stars'] < 3, 'sentiment'] = 0 # negative lable given to rating lower than 3 stars
df_joined.loc[df_joined['stars'] > 3, 'sentiment'] = 1 # positive label given to ratings higher than 3 stars
df_joined.drop('stars', axis=1, inplace=True)
#%%
df_joined.head()
#%%
warnings.simplefilter(action='ignore', category=FutureWarning)
plt.figure(figsize=(12,8))
grouped = df_joined.sentiment.value_counts().sort_index()
sns.barplot(x=grouped.index, y=grouped.values, palette=sns.color_palette("RdBu_r", len(grouped)))
plt.xlabel('Stars Ratings', labelpad=10, fontsize=14)
```

```
plt.ylabel('Number of Reviews', fontsize=14)
plt.title('Star Rating Distribution', fontsize=15)
plt.tick_params(labelsize=14)
for  i, v in enumerate(grouped):
        plt.text(i,   v*1.02,   str(v),   horizontalalignment   ='center',fontweight='bold',
fontsize=14)

#%%
from sklearn.utils import resample

df_more = df_joined[(df_joined['sentiment']==1)]
df_less = df_joined[(df_joined['sentiment']==0)]



df_less_resampled = resample(df_less,
                        replace=True,
                        n_samples= len(df_more), # to match majority class
                        random_state=42)
df_clean = pd.concat([df_less_resampled, df_more])

df_clean.sentiment.value_counts()
#%%
# Remove spaCy stopwords
sp = spacy.load('en_core_web_sm')
stopwords = sp.Defaults.stop_words
exclude_stopwords = ['no', 'not']
for word in exclude_stopwords:
```

```
    stopwords.remove(word)
#%%
# Define text preprocessing function
# The input is a single string (a raw restaurant review), and the output is a single string
(a preprocessed restaurant review)

def text_preprocessing( raw_review ):
    # 1. Remove non-letters
    review_text_letters_only = re.sub("[^a-zA-Z]", " ", raw_review)
    # 2. Convert to lower case
    review_preprocessed = review_text_letters_only.lower()
    # 3. Word tokenization
    review_tokens = word_tokenize(review_preprocessed)
    # 4. Filter the stopwords
    filtered_sentence =[]
    for word in review_tokens:
        lexeme = sp.vocab[word]
        if lexeme.is_stop == False:
            filtered_sentence.append(word)

    return " ".join(filtered_sentence)
#%%
# Apply text preprocessing to create cleaned_reviews column
df_clean['cleaned_reviews'] = df_clean['restaurant_reviews'].apply(text_preprocessing)
df = df_clean.reset_index(drop=True)
#%%
df.head()
```

```
#%%
pre_trained_model='bert-base-uncased'
tokenizer=BertTokenizer.from_pretrained(pre_trained_model)
#%%
token_lens = []

for txt in df.cleaned_reviews:
  tokens = tokenizer.encode(txt, max_length=512, truncation=True)
  token_lens.append(len(tokens))

sns.set(font_scale=1.4)
plt.rcParams["figure.figsize"] = (14,8)
sns.histplot(token_lens)
plt.xlim([0, 512])
plt.xlabel('Number of Tokens')
plt.show()
#%%
#Deciding max length based on the lengths
MAX_SEQ_LENGTH = 260
#%%
x_train, x_test, y_train, y_test = train_test_split(df['cleaned_reviews'],df.sentiment,
test_size=0.3, random_state = 42, stratify=df.sentiment)
#%%
x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.5, random_state
= 42, stratify= y_test)
#%%
# Create TensorDatasets
```

```
y_train = y_train.astype(int)
y_val = y_val.astype(int)
y_test = y_test.astype(int)

train_reviews = x_train.tolist()
val_reviews = x_val.tolist()
test_reviews = x_test.tolist()
#%%
# Train dataset
train_input_ids = [tokenizer.encode(train_reviews[i],add_special_tokens = True, max_length=MAX_SEQ_LENGTH, truncation=True) for i in range(0,len(train_reviews))]
# Val dataset
val_input_ids = [tokenizer.encode(val_reviews[i],add_special_tokens = True, max_length=MAX_SEQ_LENGTH, truncation=True) for i in range(0,len(val_reviews))]
# Test dataset
test_input_ids = [tokenizer.encode(test_reviews[i],add_special_tokens = True, max_length=MAX_SEQ_LENGTH, truncation=True) for i in range(0,len(test_reviews))]



#%%
from keras.preprocessing.sequence import pad_sequences    # Pad utility function to pad sequences to maximum length.

# Padding value: is optional, the default is 0.
```

```
# Train dataset
train_input_ids = pad_sequences(train_input_ids, maxlen=MAX_SEQ_LENGTH, dtype="long",
                  value=0, truncating="post", padding="post")

# Validation dataset
val_input_ids = pad_sequences(val_input_ids, maxlen=MAX_SEQ_LENGTH, dtype="long",
                  value=0, truncating="post", padding="post")

# Test dataset
test_input_ids = pad_sequences(test_input_ids, maxlen=MAX_SEQ_LENGTH, dtype="long",
                  value=0, truncating="post", padding="post")
#%%
# Create attention masks

# Train dataset
train_attention_masks = [[int(token_id > 0) for token_id in review]
                  for review in train_input_ids]
# dev dataset
val_attention_masks = [[int(token_id > 0) for token_id in review]
                  for review in val_input_ids]
# Test dataset
test_attention_masks = [[int(token_id > 0) for token_id in review]
                  for review in test_input_ids]
```

```
#%%
# input_ids
train_inputs = torch.tensor(train_input_ids)
val_inputs = torch.tensor(val_input_ids)
test_inputs = torch.tensor(test_input_ids)
# labels
train_labels = torch.tensor(y_train.values)
val_labels = torch.tensor(y_val.values)
test_labels = torch.tensor(y_test.values)
# attention masks
train_masks = torch.tensor(train_attention_masks)
val_masks = torch.tensor(val_attention_masks)
test_masks = torch.tensor(test_attention_masks)
#%%
batch_size = 16

# Create the DataLoader for our training set.
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)

# Create the DataLoader for our validation set.
val_data = TensorDataset(val_inputs, val_masks, val_labels)
val_sampler = SequentialSampler(val_data)
val_dataloader = DataLoader(val_data, sampler=val_sampler, batch_size=batch_size)
```

```
# Create the DataLoader for our test set.
test_data = TensorDataset(test_inputs, test_masks, test_labels)
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=batch_size)
#%%
# Number of classes / labels
n_classes = y_train.nunique()
n_classes
#%%
from transformers import logging
logging.set_verbosity_error()

model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # The 12-layer BERT model with an uncased vocab
    num_labels = 2, # For binary classification
    output_attentions = False, # Not to return attentions weights
    output_hidden_states = False, # Not to return all hidden-states
)
#%%
model = model.to(device)
#%%
epochs=2

optimizer=AdamW(model.parameters(),lr=3e-5)
total_steps=len(train_dataloader)*epochs

# Create the learning rate scheduler
```

```
scheduler=get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=total_steps
)
 # Define loss function and move it to GPU
loss_fn=nn.CrossEntropyLoss().to(device)
#%%
def format_time(elapsed):
    # Round to the nearest second
    elapsed_round = int(round(elapsed))
    # Format time in hh:mm:ss
    return str(datetime.timedelta(seconds = elapsed_round))

def accuracy(preds, labels):
    preds = np.argmax(preds, axis=1).flatten()
    labels = labels.flatten()
    return np.sum(preds == labels) / len(labels)
#%%
from tqdm import tqdm
import time

# Assuming you have already imported the necessary libraries and defined your model, dataloaders, etc.

# Store for each epoch
loss_train_values = []
```

```
acc_train_values = []
loss_val_values = []
acc_val_values = []

# Training loop
for epoch in range(epochs):
   # --- Train ---

   # Perform forward pass over the training dataset
   print("\n Epoch {:}/{:} :".format(epoch + 1, epochs))
   print('Training....')

   # Measure how long the training epoch takes
   t0 = time.time()
   # Reset total loss and accuracy for this epoch
   total_loss = 0
   total_acc = 0

   # Put the model in training mode
   model.train()

   # For each batch of training data
        for step, batch in enumerate(tqdm(train_dataloader, desc=f'Epoch {epoch +
1}/{epochs}')):
      # Update progress for each 100 steps
      if (step % 100 == 0) and (not step == 0):
         # Calculate elapsed time in minutes
```

```
            elapsed = format_time((time.time() - t0))
            # Report progress
            print(' Batch {:>5,} of {:>5,}. Elapsed:{:}.'.format(step, len(train_dataloader), elapsed))

        # Unpack training batch from trainloader and move to GPU
        b_input_ids = batch[0].to(device)  # 0 - input ids tensor
        b_attention_mask = batch[1].to(device)  # 1 - input masks tensor
        b_labels = batch[2].long().to(device)  # Convert to torch.long

        # Clear any previously calculated gradients in PyTorch before performing a backward pass
        model.zero_grad()

        # Output the results
        outputs = model(input_ids=b_input_ids, attention_mask=b_attention_mask, labels=b_labels)  # Return tuple
        # Loss value from output
        loss = outputs.loss  # Loss

        # Update total loss
        total_loss += loss.item()

        preds = outputs.logits  # Output probabilities
        # Move logits and labels to CPU
        preds = preds.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()
```

```
        # Calculate the accuracy for this batch
        tmp_train_accuracy = accuracy(preds, label_ids)

        # Accumulate the total accuracy
        total_acc += tmp_train_accuracy

        # Perform a backward pass to calculate gradients
        loss.backward()

        # To avoid exploding vanishing gradients problem, clip the norm of the gradients
to 1.0
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        # Update the parameters (weights)
        optimizer.step()

        # Update the learning rate
        scheduler.step()

    # Calculate the average loss over training data
    avg_total_loss = total_loss / len(train_dataloader)

    # Store the loss values
    loss_train_values.append(avg_total_loss)

    # Calculate the average accuracy over the training data
```

```
avg_train_acc = total_acc / len(train_dataloader)

# Store the accuracy values
acc_train_values.append(avg_train_acc)

print("")
print("\nAverage training accuracy: {0:.2f}".format(avg_train_acc))

print('Average training loss : {0:.2f}'.format(avg_total_loss))
print('Training epoch took: {:}'.format(format_time(time.time() - t0)))

# --- VALIDATION ---

# After each epoch perform validation to check model performance
print('\n Running validation...')

t0 = time.time()
# Put model in evaluation mode
model.eval()

# Tracking variables
total_eval_accuracy = 0
total_eval_loss = 0

# Unpack validation batch from trainloader and move to GPU
for batch in tqdm(val_dataloader, desc='Validation'):
    b_input_ids = batch[0].to(device)
```

```
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].long().to(device)  # Convert to torch.long

        # Tell model not to compute gradients to save memory and accelerate validation
        with torch.no_grad():
            # Forward pass, calculate logit prediction
            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
        loss = outputs.loss
        logits = outputs.logits
        # Update total evaluation loss
        total_eval_loss += loss.item()

        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Calculate the accuracy for this batch and accumulate it over all batches
        total_eval_accuracy += accuracy(logits, label_ids)

    # Compute the average accuracy over all of the batches
    avg_val_accuracy = total_eval_accuracy / len(val_dataloader)

    # Store the accuracy values
    acc_val_values.append(avg_val_accuracy)

    # Compute the average loss over all of the batches
```

```
    avg_val_loss = total_eval_loss / len(val_dataloader)

    # Store the loss values
    loss_val_values.append(avg_val_loss)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))
    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

#%%
import os

model_save_path = 'Saved_models/sentiment_model.pth'
os.makedirs('Saved_models/', exist_ok=True)  # Create the directory if it doesn't exist
torch.save(model.state_dict(), model_save_path)
print(f'Model saved to {model_save_path}')
#%%
import pickle

# Assuming you have the lists you want to save: loss_train_values, acc_train_values, loss_val_values, acc_val_values

# Save the lists to a file
with open('Saved_models/training_metrics.pkl', 'wb') as f:
    pickle.dump({
```

```
        'loss_train_values': loss_train_values,
        'acc_train_values': acc_train_values,
        'loss_val_values': loss_val_values,
        'acc_val_values': acc_val_values
    }, f)
#%%
# Load the lists from the file
with open('Saved_models/training_metrics.pkl', 'rb') as f:
    loaded_metrics = pickle.load(f)

# Access the loaded lists
loaded_loss_train_values = loaded_metrics['loss_train_values']
loaded_acc_train_values = loaded_metrics['acc_train_values']
loaded_loss_val_values = loaded_metrics['loss_val_values']
loaded_acc_val_values = loaded_metrics['acc_val_values']
#%%
model_save_path = 'Saved_models/sentiment_model.pth'
model.load_state_dict(torch.load(model_save_path))

# Move the model to the device (GPU or CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Set the model to evaluation mode
model.eval()
#%%
df_acc = pd.DataFrame(acc_val_values,columns=['Accuracy'])
```

```
df_acc.index+=1
#%%
# Use plot styling from seaborn
sns.set(style='darkgrid')

# Increase the plot size and font size
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (12,6)

# Plot the learning curve
sns.lineplot(data=df_acc,x=df_acc.index,y=df_acc.Accuracy)

# Label the plot
plt.title("Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")

plt.show()

#%%

df_loss = pd.DataFrame(loss_val_values,columns=['Loss'])
df_loss.index+=1
#%%


# Use plot styling from seaborn
```

```
sns.set(style='darkgrid')

# Increase the plot size and font size
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (12,6)

# Plot the learning curve
sns.lineplot(data=df_loss,x=df_loss.index,y=df_loss.Loss)

# Label the plot
plt.title("Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.show()


#%%
# Put model in evaluation mode
model.eval()

# Tracking variables
predictions, true_labels = [], []

# Create a tqdm progress bar for the test dataloader
for batch in tqdm(test_dataloader, desc='Predicting', leave=False):
    # Move batch to GPU
```

```
    batch = tuple(t.to(device) for t in batch)
    # Unpack inputs from test dataloader
    b_input_ids, b_attention_mask, b_labels = batch
    # Tell model not to compute gradients to save memory and accelerate validation
    with torch.no_grad():
        # Forward pass, calculate logit prediction
        outputs = model(input_ids=b_input_ids, attention_mask=b_attention_mask)
    # logits are class probabilities and get them from outputs
    logits = outputs[0]

    # Store predictions and true labels
    predictions.extend(logits.tolist())
    true_labels.extend(b_labels.tolist())

print('Predictions Done')

#%%



import torch.nn.functional as F
preds = torch.tensor(predictions)

preds = F.softmax(preds,dim=1)



preds = np.array(preds)
true_labels = np.array(true_labels)
```

```
#%%


def evaluate(y_test, predictions):
    cf_matrix = confusion_matrix(true_labels, preds.argmax(1))
    sns.heatmap(cf_matrix, annot = True, fmt = 'd',cmap="Blues")
    plt.title('Heatmap of confusion matrix for Test data')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
evaluate(true_labels, preds.argmax(1))


#%%


class_report= classification_report(true_labels, preds.argmax(1), digits=3)
print(class_report)
```

b. Linear Support Vector Machine model

```
#%%
# Importing libraries for plotting and data analysis
%pylab inline
import warnings

# Ignore warnings
warnings.filterwarnings('ignore')

# Configure the IPython shell
from IPython.core.interactiveshell import InteractiveShell

# Setting the display behavior to show the output of all expressions in a single cell
InteractiveShell.ast_node_interactivity = "all"
#%%
%%html
<style>
.output_wrapper, .output {
    height:auto !important;
    max-height:2000px;  /* your desired max-height here */
}
.output_scroll {
    box-shadow:none !important;
    webkit-box-shadow:none !important;
}
</style>
#%%
# Importing libraries for DA/NLP
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
# nltk.download('stopwords')

# tqdm for displaying progress bars during iterations
from tqdm import tqdm_notebook as tqdm
from collections import Counter
#%%
# Loading dataset
reviews_dataset = pd.read_csv('yelp_csv_dataset.csv')

#Dropping unwanted columns and selecting only needed columns
selected_columns = ['review_id', 'business_id', 'user_id', 'text', 'stars']
reviews_dataset = reviews_dataset[selected_columns]

# Display the modified dataset
display(reviews_dataset.head())
#%%
# Check No.of Rows
print("Total No. of Reviews: {}".format(reviews_dataset.shape))
#%%
reviews_dataset.shape
#%%
```

```
import string

def get_clean_text(sample_review):
    # List of English stopwords from NLTK
    stopwords = nltk.corpus.stopwords.words('english')
    # Additional custom stopwords to be excluded
    newStopWords = ['ive', 'hadnt', 'couldnt', 'didnt', 'id']
    stopwords.extend(newStopWords)

    # Original text from the sample review
    text = sample_review

    # Convert to lowercase
    text = text[2: len(sample_review)-1].lower()

    # Replace special characters with spaces
    text = text.replace('\\n', ' ').replace('\\t', ' ')

    # Remove punctuation
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)

    # Remove stopwords
    l = [word for word in nopunc.split() if word.lower() not in stopwords]

    # Reconstruct the cleaned text
    clean_text = " ".join(l)
```

```
    return clean_text.strip()
#%%
# Display the cleaned text using the get_clean_text function
sample_review = reviews_dataset.text[50]
display(get_clean_text(sample_review))
#%%
import string

def get_words(text):
    # List of English stopwords from NLTK
    stopwords = nltk.corpus.stopwords.words('english')

    # Additional custom stopwords to be excluded
    newStopWords = ['ive', 'hadnt', 'couldnt', 'didnt', 'id']
    stopwords.extend(newStopWords)

    # Case normalization
    text = text[2: len(text)-1].lower()
    text = text.replace('\\n', ' ').replace('\\t', ' ')

    # Display the processed text
    display(text)
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    display(nopunc)
```

```
    # Remove stopwords and create a list of words
    words_list = [word for word in nopunc.split() if word.lower() not in stopwords]

    # Return the list of words and its length
    return words_list, len(words_list)
#%%
for i in range(1):
    # Convert the review to a string
    sample_review = str(reviews_dataset.text[i])
    print(sample_review)

    # Process the review using the get_words function
    result = get_words(sample_review)

    # Display the list of words
    display(result[0])
#%%
pd.set_option('display.precision', 2)
reviews_dataset.describe()
#%%
# Count the occurrences of each unique value
reviews_dataset["stars"].value_counts()
type(reviews_dataset["stars"].value_counts())
#%%
#Visualize the distribution
labels = '5-Stars', '4-Stars', '1-Star', '3-Stars', '2-Stars'
sizes = reviews_dataset["stars"].value_counts()
```

```
colors = ['lightpink', 'yellowgreen', 'orange', 'lightskyblue','green']

# Plot
plt.pie(sizes, labels=labels, colors =colors, autopct='%1.1f%%')
plt.axis('equal')
plt.show()
#%%
# Initialize an empty list to store cleaned texts
texts = []
stars = [reviews_dataset['stars'] for review in reviews_dataset]

# Iterate through rows in the reviews_dataset df
pbar = tqdm(total=reviews_dataset.shape[0] + 1)

# Iterate through each row in the DataFrame
for index, row in reviews_dataset.iterrows():
    cleaned_text = get_clean_text(row['text'])

    # Append the cleaned text to the texts list
    texts.append(cleaned_text)
    pbar.update(1)

pbar.close()

#%% md

#%%
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(ngram_range=(1,3))
vectors = vectorizer.fit_transform(texts)
# The vectors matrix now contains the numerical representation of the texts using TF-IDF
#%%
from sklearn.model_selection import train_test_split
#Spit the dataset into train and test parts
X_train, X_test, y_train, y_test = train_test_split(vectors, stars[1], test_size=0.15, random_state=42, shuffle =False)
#%%
from sklearn.svm import LinearSVC

# initialise the SVM classifier
classifier = LinearSVC()

# train the classifier
classifier.fit(X_train, y_train)
#%%
# Make predictions on the test set using the trained classifier
preds = classifier.predict(X_test)
print("Actual Ratings(Stars): ",end = "")
display(y_test[:5])
print("Predicted Ratings: ",end = "")
print(preds[:5])
#%%
```

```
X_null, X_full_test, y_null, y_full_test = train_test_split(vectors, stars[1],
test_size=0.2, random_state=42, shuffle = False)

# Make predictions on the full test set using the trained classifier
predict_all = classifier.predict(X_full_test)
#%%
# Convert the predicted ratings to a list
predicted_stars = list(predict_all)
print("Actual Ratings(Stars): ")
print(y_full_test[-5:])

# Display the predicted ratings for the last 5 samples in the full test set
print("\nPredicted Ratings: ", end="")
print(predicted_stars[-5:])

#%%
print("\nOriginal Reviews (with user bias)")
display(reviews_dataset.tail(10))

print("\nUnbiased Reviews (with predicted rating using user's review text)")
unbiased_reviews_dataset = reviews_dataset

# dropping actual ratings(stars) by user
unbiased_reviews_dataset = unbiased_reviews_dataset.drop('stars', 1)

# adding the unbiased predicted rating
unbiased_reviews_dataset['stars'] = predicted_stars
```

```
display(unbiased_reviews_dataset.tail(10))


#%%
# Calculate accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, preds))
#%%
# Calculate the precision_score and recall_scor
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
print ('Precision: ' + str(precision_score(y_test, preds, average='weighted')))
print ('Recall: ' + str(recall_score(y_test, preds, average='weighted')))
#%%
from sklearn.metrics import classification_report
print(classification_report(y_test, preds))
#%%
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
```

```
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
#%%
from sklearn import metrics
names = ['1','2','3','4','5']

# Compute confusion matrix
cnf_matrix = metrics.confusion_matrix(y_test, preds)
np.set_printoptions(precision=2)
```

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
#%%
# making binary classes
sentiments = []
for star in stars[1]:
    if star <= 3:
        sentiments.append('negative')
    if star > 3:
        sentiments.append('positive')

print(len(sentiments))
#%%
# Split the TF-IDF vectors and corresponding binary sentiment classes (sentiments) into training and testing sets
X2_train, X2_test, y2_train, y2_test = train_test_split(vectors, sentiments, test_size=0.20, random_state=42)
```

```
#%%
# Initialize a SVM for binary sentiment classification
classifier2 = LinearSVC()

# Train the classifier on the training data for binary sentiment
classifier2.fit(X2_train, y2_train)
#%%
#Make predictions
preds2 = classifier2.predict(X2_test)
print("Actual Class:    ",end = "")
print(y2_test[:10])
print("\nPredicted Class: ",end = "")
print(list(preds2[:10]))
#%%
#Calculate accuracy
print(accuracy_score(y2_test, preds2))
#%%
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
print ('Precision: ' + str(precision_score(y2_test, preds2, average='weighted')))
print ('Recall: ' + str(recall_score(y2_test, preds2, average='weighted')))
#%%
print(classification_report(y2_test, preds2))
#%%
print(metrics.confusion_matrix(y2_test, preds2))

#%%
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming preds_scaled contains the predicted labels for the test set

# Compute confusion matrix
cm = confusion_matrix(y2_test, preds2)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
#%%
# Compute accuracy
accuracy = accuracy_score(y2_test, preds2)
print(f'Accuracy: {accuracy:.2%}')

# Display classification report
print('Classification Report:')
print(classification_report(y2_test, preds2))
#%%
```

9. References

[1] Josepaulosa, "GitHub - josepaulosa/NLP_Sentiment_Analysis: Sentiment analysis of the restaurant reviews from YELP dataset using BoW, TF-IDF, Word2Vec, Doc2Vec, Glove and BERT.," *GitHub*. https://github.com/josepaulosa/NLP_Sentiment_Analysis/tree/main

[2] Skshashankkumar, "Sentiment-Analysis-Using-Transformers-PyTorch/Sentiment_Classification_Using_Transformers.ipynb at master · skshashankkumar41/Sentiment-Analysis-Using-Transformers-PyTorch," *GitHub*. https://github.com/skshashankkumar41/Sentiment-Analysis-Using-Transformers-PyTorch/blob/master/Sentiment_Classification_Using_Transformers.ipynb

[3] Suzanaiacob, "Sentiment analysis of the Yelp reviews data," *Kaggle*, Mar. 07, 2019. https://www.kaggle.com/code/suzanaiacob/sentiment-analysis-of-the-yelp-reviews-data

[4] Rafaljanwojcik, "GitHub - rafaljanwojcik/Unsupervised-Sentiment-Analysis: How to extract sentiment from opinions without any labels," *GitHub*. https://github.com/rafaljanwojcik/Unsupervised-Sentiment-Analysis

[5] Omkarsabnis, "Sentiment analysis on the Yelp Reviews Dataset," *Kaggle*, Jun. 08, 2018. https://www.kaggle.com/code/omkarsabnis/sentiment-analysis-on-the-yelp-reviews-dataset

[6] Barelydedicated, "Yelp review predictions using HuggingFace (BERT)," *Kaggle*, Dec. 30, 2019.

https://www.kaggle.com/code/barelydedicated/yelp-review-predictions-using-huggingface-bert/notebook

[7] “Yelp Dataset.” https://www.yelp.com/dataset/documentation/main

[8] SuyashSonawane, “GitHub - SuyashSonawane/sentimental_analysis-api: To run and check the webapp go to the link,” *GitHub*. https://github.com/SuyashSonawane/sentimental_analysis-api

[9] Skshashankkumar, “Sentiment-Analysis-Using-Transformers-PyTorch/Sentiment_Classification_Using_Transformers.ipynb at master · skshashankkumar41/Sentiment-Analysis-Using-Transformers-PyTorch,” *GitHub*. https://github.com/skshashankkumar41/Sentiment-Analysis-Using-Transformers-PyTorch/blob/master/Sentiment_Classification_Using_Transformers.ipynb

[10] dailyLi, “GitHub - dailyLi/yelp_da: NLP Sentiment Analysis with Yelp Review Data Set,” *GitHub*. https://github.com/dailyLi/yelp_da

[11]Karantyagi, “GitHub - karantyagi/Restaurant-Recommendations-with-Yelp: 🍕Recommend new restaurants to Yelp users, using ratings predicted from reviews.,” *GitHub*. https://github.com/karantyagi/Restaurant-Recommendations-with-Yelp

[12] Dehaoterryzhang, “Yelp_Sentiment_Analysis/code at master · dehaoterryzhang/Yelp_Sentiment_Analysis,” *GitHub*. https://github.com/dehaoterryzhang/Yelp_Sentiment_Analysis/tree/master/code