Project Machine Learning II - Jan Schmid

In this project, I undertook the challenge of tackling one of the most popular tasks in Natural Language Processing (NLP) - Sentiment Analysis. The goal was to develop models capable of understanding and interpreting human sentiment based on text reviews. Specifically, I constructed three distinct models: a Logistic Regression model, a BERT (Bidirectional Encoder Representations from Transformers) model, and a Convolutional Neural Network (CNN) model.

The task was to train these models to learn from reviews and predict a rating ranging from 1 to 5, a multi-class classification problem. The data used for this task was gathered from two sources: TripAdvisor and Amazon food reviews. The datasets were found on Kaggle. By leveraging these different sources, the models were trained on a broad range of review types, potentially enhancing their generalization ability and robustness.

This notebook is tailored for the Google Colab environment. If you are planning to run it locally, please note that file paths used for data access might need adjustments to fit your local environment.

In the following sections, I will be going through the process of data gathering and cleaning, feature extraction, model training, evaluation, and final decision-making.

Installation of the needed packages

```
# Install necessary packages
# path needs to be adjusted, to where you have the requirements.txt
file
!pip install -r /content/drive/MyDrive/ML_Project/requirements.txt
!pip install langdetect
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: absl-py==1.4.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 1)) (1.4.0)
Requirement already satisfied: astunparse==1.6.3 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 2)) (1.6.3)
Collecting async-generator==1.10 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 3))
  Using cached async_generator-1.10-py3-none-any.whl (18 kB)
Requirement already satisfied: attrs==23.1.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 4)) (23.1.0)
Collecting cachetools==5.3.1 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 5))
  Using cached cachetools-5.3.1-py3-none-any.whl (9.3 kB)
Collecting certifi==2023.5.7 (from -r
```

```
/content/drive/MyDrive/ML_Project/requirements.txt (line 6))
  Using cached certifi-2023.5.7-py3-none-any.whl (156 kB)
Requirement already satisfied: cffi==1.15.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 7)) (1.15.1)
Collecting charset-normalizer==3.1.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 8))
  Using cached charset_normalizer-3.1.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (199 kB)
Requirement already satisfied: click==8.1.3 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 9)) (8.1.3)
Collecting colorama==0.4.6 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 10))
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: contourpy==1.0.7 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 11)) (1.0.7)
Requirement already satisfied: cycler==0.11.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 12)) (0.11.0)
Requirement already satisfied: exceptiongroup==1.1.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 13)) (1.1.1)
Collecting filelock==3.12.1 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 14))
  Using cached filelock-3.12.1-py3-none-any.whl (10 kB)
Collecting flatbuffers==23.5.26 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 15))
  Using cached flatbuffers-23.5.26-py2.py3-none-any.whl (26 kB)
Collecting fonttools==4.39.4 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 16))
  Using cached fonttools-4.39.4-py3-none-any.whl (1.0 MB)
Collecting fsspec==2023.6.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 17))
  Using cached fsspec-2023.6.0-py3-none-any.whl (163 kB)
Requirement already satisfied: gast==0.4.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 18)) (0.4.0)
Collecting google-auth==2.19.1 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 19))
  Using cached google_auth-2.19.1-py2.py3-none-any.whl (181 kB)
Requirement already satisfied: google-auth-oauthlib==1.0.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 20)) (1.0.0)
Requirement already satisfied: google-pasta==0.2.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 21)) (0.2.0)
Collecting grpcio==1.54.2 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 22))
```

```
  Using cached grpcio-1.54.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.1 MB)
Collecting h11==0.14.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 23))
  Using cached h11-0.14.0-py3-none-any.whl (58 kB)
Requirement already satisfied: h5py==3.8.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 24)) (3.8.0)
Collecting huggingface-hub==0.15.1 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 25))
  Using cached huggingface_hub-0.15.1-py3-none-any.whl (236 kB)
Requirement already satisfied: idna==3.4 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 26)) (3.4)
Collecting importlib-metadata==6.6.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 27))
  Using cached importlib_metadata-6.6.0-py3-none-any.whl (22 kB)
Requirement already satisfied: importlib-resources==5.12.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 28)) (5.12.0)
Collecting jax==0.4.12 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 29))
  Using cached jax-0.4.12.tar.gz (1.3 MB)
  Installing build dependencies ... ents to build wheel ... etadata
(pyproject.toml) ... ent already satisfied: Jinja2==3.1.2 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 30)) (3.1.2)
Requirement already satisfied: joblib==1.2.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 31)) (1.2.0)
Requirement already satisfied: keras==2.12.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 32)) (2.12.0)
Requirement already satisfied: kiwisolver==1.4.4 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 33)) (1.4.4)
Collecting langdetect==1.0.9 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 34))
  Using cached langdetect-1.0.9.tar.gz (981 kB)
  Preparing metadata (setup.py) ... ent already satisfied:
libclang==16.0.0 in /usr/local/lib/python3.10/dist-packages (from -
r /content/drive/MyDrive/ML_Project/requirements.txt (line 35))
(16.0.0)
Requirement already satisfied: Markdown==3.4.3 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 36)) (3.4.3)
Collecting MarkupSafe==2.1.3 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 37))
  Using cached MarkupSafe-2.1.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
```

```
Requirement already satisfied: matplotlib==3.7.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 38)) (3.7.1)
Collecting ml-dtypes==0.2.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 39))
  Using cached ml_dtypes-0.2.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
Requirement already satisfied: mpmath==1.3.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 40)) (1.3.0)
Requirement already satisfied: networkx==3.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 41)) (3.1)
Requirement already satisfied: nltk==3.8.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 42)) (3.8.1)
Collecting numpy==1.23.5 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 43))
  Using cached numpy-1.23.5-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
Requirement already satisfied: oauthlib==3.2.2 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 44)) (3.2.2)
Requirement already satisfied: opt-einsum==3.3.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 45)) (3.3.0)
Collecting outcome==1.2.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 46))
  Using cached outcome-1.2.0-py2.py3-none-any.whl (9.7 kB)
Requirement already satisfied: packaging==23.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 47)) (23.1)
Collecting pandas==2.0.2 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 48))
  Using cached pandas-2.0.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.3 MB)
Collecting Pillow==9.5.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 49))
  Using cached Pillow-9.5.0-cp310-cp310-manylinux_2_28_x86_64.whl (3.4
MB)
Collecting protobuf==4.23.2 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 50))
  Using cached protobuf-4.23.2-cp37-abi3-manylinux2014_x86_64.whl (304
kB)
Requirement already satisfied: pyasn1==0.5.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 51)) (0.5.0)
Requirement already satisfied: pyasn1-modules==0.3.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 52)) (0.3.0)
```

```
Requirement already satisfied: pycparser==2.21 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 53)) (2.21)
Requirement already satisfied: pyparsing==3.0.9 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 54)) (3.0.9)
Requirement already satisfied: PySocks==1.7.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 55)) (1.7.1)
Requirement already satisfied: python-dateutil==2.8.2 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 56)) (2.8.2)
Collecting pytz==2023.3 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 57))
  Using cached pytz-2023.3-py2.py3-none-any.whl (502 kB)
Requirement already satisfied: PyYAML==6.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 58)) (6.0)
Collecting regex==2023.6.3 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 59))
  Using cached regex-2023.6.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (770 kB)
Collecting requests==2.31.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 60))
  Using cached requests-2.31.0-py3-none-any.whl (62 kB)
Requirement already satisfied: requests-oauthlib==1.3.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 61)) (1.3.1)
Requirement already satisfied: rsa==4.9 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 62)) (4.9)
Collecting safetensors==0.3.1 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 63))
  Using cached safetensors-0.3.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
Requirement already satisfied: scipy==1.10.1 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 64)) (1.10.1)
Requirement already satisfied: seaborn==0.12.2 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 65)) (0.12.2)
Collecting selenium==4.10.0 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 66))
  Using cached selenium-4.10.0-py3-none-any.whl (6.7 MB)
Requirement already satisfied: six==1.16.0 in
/usr/local/lib/python3.10/dist-packages (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 67)) (1.16.0)
Collecting sklearn==0.0.post5 (from -r
/content/drive/MyDrive/ML_Project/requirements.txt (line 68))
  Using cached sklearn-0.0.post5.tar.gz (3.7 kB)
```

```
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a
problem with pip.
  Preparing metadata (setup.py) ... error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

```
# Import necessary packages
import os
import re
import torch
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
!pip install langdetect
from langdetect import detect
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from torch import nn
from torch.nn import functional as F
from torch.utils.data import TensorDataset, random_split, DataLoader,
RandomSampler, SequentialSampler
!pip install transformers
from transformers import BertTokenizer, BertForSequenceClassification,
AdamW, get_linear_schedule_with_warmup
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from google.colab import drive
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
import pickle
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: langdetect in
/usr/local/lib/python3.10/dist-packages (1.0.9)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from langdetect) (1.16.0)
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting transformers
  Downloading transformers-4.30.1-py3-none-any.whl (7.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.2/7.2 MB 101.3 MB/s eta
0:00:00
ent already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.12.0)
Collecting huggingface-hub<1.0,>=0.14.1 (from transformers)
  Using cached huggingface_hub-0.15.1-py3-none-any.whl (236 kB)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers)
(2022.10.31)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading tokenizers-0.13.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 101.7 MB/s eta
0:00:00
 transformers)
  Using cached safetensors-0.3.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.14.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.14.1->transformers) (4.5.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2.0.12)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
```

```
(3.4)
Installing collected packages: tokenizers, safetensors, huggingface-
hub, transformers
Successfully installed huggingface-hub-0.15.1 safetensors-0.3.1
tokenizers-0.13.3 transformers-4.30.1
```

Data gathering and Processing

```
# Import the necessary package for mounting Google Drive in Google
Colab
from google.colab import drive

# Mount Google Drive to access files
drive.mount('/content/drive')

# Define the file paths
path1 = "/content/drive/MyDrive/ML_Project/data/New_Delhi_reviews.csv"
path2 =
"/content/drive/MyDrive/ML_Project/data/tripadvisor_hotel_reviews.csv"
path3 = "/content/drive/MyDrive/ML_Project/data/Reviews.csv"

# Read the CSV files into pandas DataFrames
df1 = pd.read_csv(path1)
df2 = pd.read_csv(path2)
df3 = pd.read_csv(path3)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
df1.head()
```

```
   rating_review                                        review_full
0              5  Totally in love with the Auro of the place, re...
1              5  I went this bar 8 days regularly with my husba...
2              5  We were few friends and was a birthday celebra...
3              5  Fatjar Cafe and Market is the perfect place fo...
4              5  Hey Guys, if you are craving for pizza and sea...
```

```
#rename the columns
df1.rename(columns={'rating_review':'Rating', 'review_full':'Review'},
inplace=True)
```

```
df2.head()
```

```
                                              Review  Rating
0  nice hotel expensive parking got good deal sta...       4
1  ok nothing special charge diamond member hilto...       2
2  nice rooms not 4* experience hotel monaco seat...       3
3  unique, great stay, wonderful time hotel monac...       5
4  great stay great stay, went seahawk game aweso...       5
```

```
df3.head()
```

```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                Summary
Text
0  Good Quality Dog Food  I have bought several of the Vitality canned
d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted
Peanut...
2  "Delight" says it all  This is a confection that has been around a
fe...
3         Cough Medicine  If you are looking for the secret ingredient
i...
4            Great taffy  Great taffy at a great price.  There was a
wid...
```

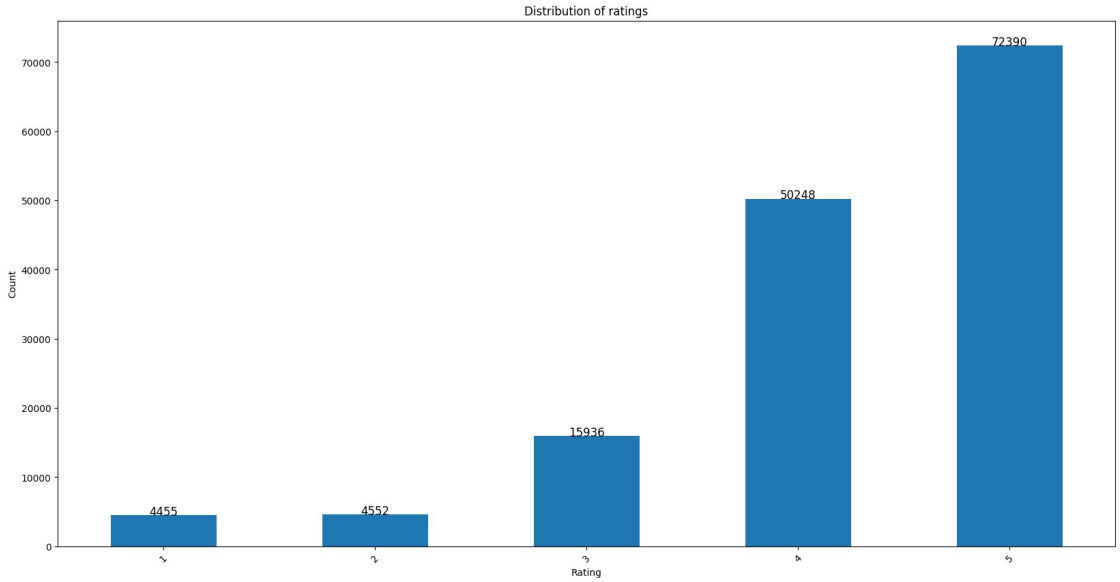
```
# Drop unnecessary columns from df3 DataFrame
df3.drop(['Id', 'ProductId', 'UserId', 'ProfileName',
'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time', 'Summary'],
axis=1, inplace=True)

df3.head()
```

```
   Score                                               Text
0      5  I have bought several of the Vitality canned d...
1      1  Product arrived labeled as Jumbo Salted Peanut...
2      4  This is a confection that has been around a fe...
3      2  If you are looking for the secret ingredient i...
4      5  Great taffy at a great price.  There was a wid...
```

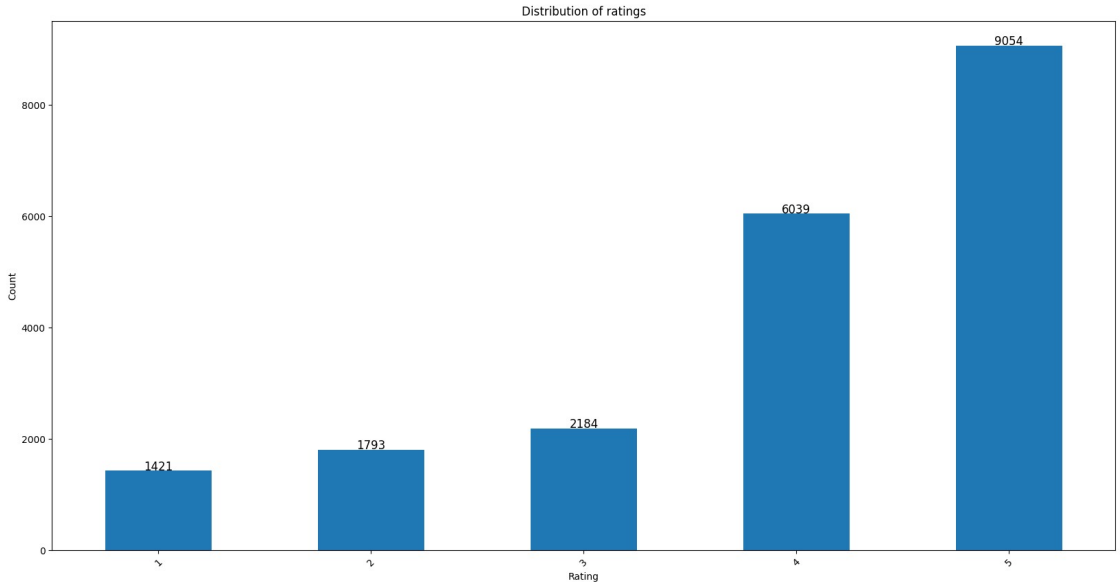
```
#rename the columns
df3.rename(columns={'Score':'Rating', 'Text':'Review'}, inplace=True)
df3.head()
```

```
   Rating                                             Review
0       5  I have bought several of the Vitality canned d...
1       1  Product arrived labeled as Jumbo Salted Peanut...
2       4  This is a confection that has been around a fe...
3       2  If you are looking for the secret ingredient i...
4       5  Great taffy at a great price.  There was a wid...
```

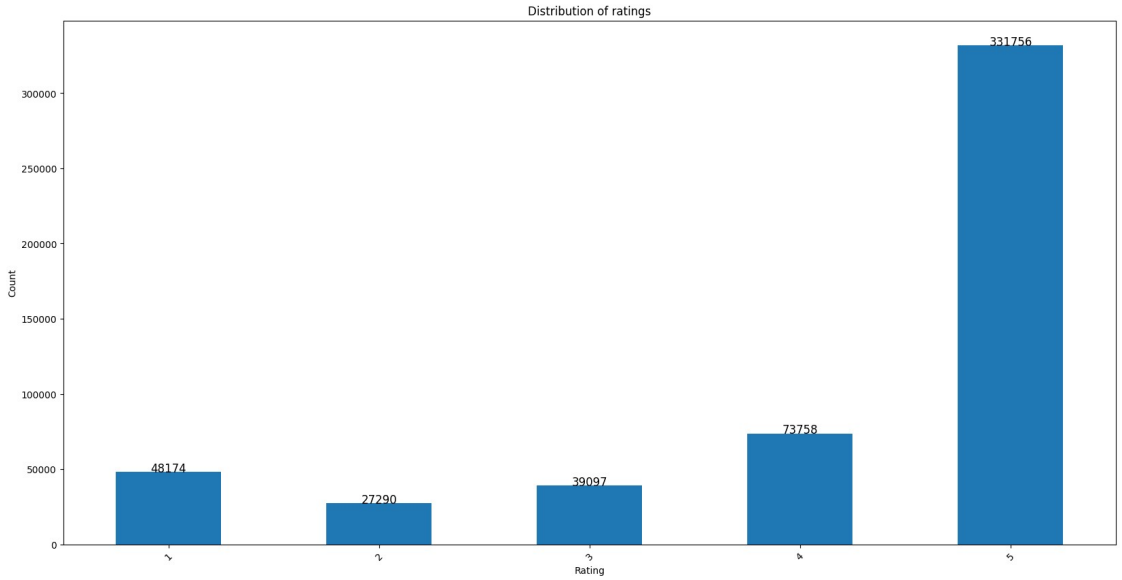
```
# Create a barchart for the ratings in the ratings dataframe
ax = df1['Rating'].value_counts().sort_index().plot(kind='bar',
figsize=(20,10))

# Set the x- and y-axis labels
ax.set_xlabel('Rating')
ax.set_ylabel('Count')

# Add annotations for the count above each bar
for i, v in enumerate(df1['Rating'].value_counts().sort_index()):
    ax.annotate(str(v), xy=(i, v + 20), ha='center', fontsize=12)

plt.title('Distribution of ratings')
plt.xticks(rotation=45)
plt.show()
```



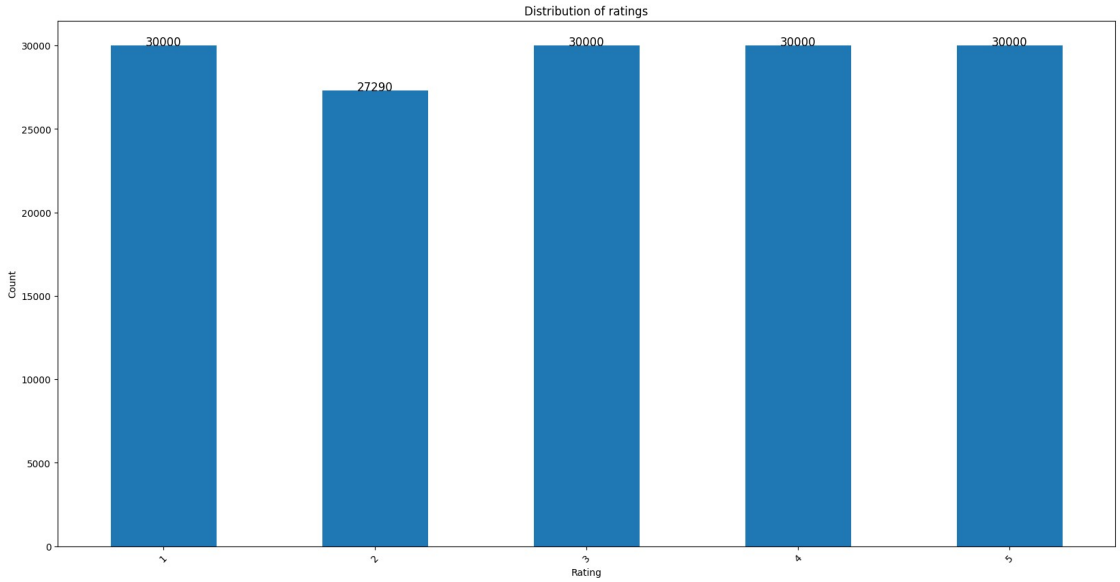

```
# Create a barchart for the ratings in the ratings dataframe
ax = df2['Rating'].value_counts().sort_index().plot(kind='bar',
figsize=(20,10))

# Set the x- and y-axis labels
ax.set_xlabel('Rating')
ax.set_ylabel('Count')

# Add annotations for the count above each bar
for i, v in enumerate(df2['Rating'].value_counts().sort_index()):
    ax.annotate(str(v), xy=(i, v + 20), ha='center', fontsize=12)

plt.title('Distribution of ratings')
plt.xticks(rotation=45)
plt.show()
```




```
# Create a barchart for the ratings in the ratings dataframe
ax = df3['Rating'].value_counts().sort_index().plot(kind='bar',
figsize=(20,10))

# Set the x- and y-axis labels
ax.set_xlabel('Rating')
ax.set_ylabel('Count')

# Add annotations for the count above each bar
for i, v in enumerate(df3['Rating'].value_counts().sort_index()):
    ax.annotate(str(v), xy=(i, v + 20), ha='center', fontsize=12)

plt.title('Distribution of ratings')
plt.xticks(rotation=45)
plt.show()
```




```
# Group the df3 DataFrame by 'Rating' and sample a subset from each
group
df3 = df3.groupby('Rating').apply(lambda x: x.sample(n=min(30000,
len(x)), replace=True, random_state=42)).reset_index(drop=True)

# Create a barchart for the ratings in the ratings dataframe
ax = df3['Rating'].value_counts().sort_index().plot(kind='bar',
figsize=(20,10))

# Set the x- and y-axis labels
ax.set_xlabel('Rating')
ax.set_ylabel('Count')

# Add annotations for the count above each bar
for i, v in enumerate(df3['Rating'].value_counts().sort_index()):
    ax.annotate(str(v), xy=(i, v + 20), ha='center', fontsize=12)

plt.title('Distribution of ratings')
plt.xticks(rotation=45)
plt.show()
```




```
#count the nan values in each column
df1.isnull().sum()
```

```
Rating    0
Review    2
dtype: int64
```

```
#drop the na values
df1.dropna(inplace=True)
```

```
#count the nan values in each column
df1.isnull().sum()
```

```
Rating    0
Review    0
dtype: int64
```

```
#count the nan values in each column
df2.isnull().sum()
```

```
Review    0
Rating    0
dtype: int64
```

```
#count the nan values in each column
df3.isnull().sum()
```

```
Rating    0
Review    0
dtype: int64
```

```
# Loop through the 'Review' column in df1 and detect language
for i in range(len(df1)):
    try:
```

```
        df1.loc[i, 'language'] = detect(df1.loc[i, 'Review'])
    except:
        df1.loc[i, 'language'] = 'error'
        pass

# Loop through the 'Review' column in df1 and detect language
for i in range(len(df2)):
    try:
        df2.loc[i, 'language'] = detect(df2.loc[i, 'Review'])
    except:
        df2.loc[i, 'language'] = 'error'
        pass

# Loop through the 'Review' column in df1 and detect language
for i in range(len(df3)):
    try:
        df3.loc[i, 'language'] = detect(df3.loc[i, 'Review'])
    except:
        df3.loc[i, 'language'] = 'error'
        pass
```

```
df1.head()
```

```
   Rating                                             Review language
0     5.0  Totally in love with the Auro of the place, re...       en
1     5.0  I went this bar 8 days regularly with my husba...       en
2     5.0  We were few friends and was a birthday celebra...       en
3     5.0  Fatjar Cafe and Market is the perfect place fo...       en
4     5.0  Hey Guys, if you are craving for pizza and sea...       en
```

```
df2.head()
```

```
                                              Review  Rating language
0  nice hotel expensive parking got good deal sta...       4       en
1  ok nothing special charge diamond member hilto...       2       en
2  nice rooms not 4* experience hotel monaco seat...       3       en
3  unique, great stay, wonderful time hotel monac...       5       en
4  great stay great stay, went seahawk game aweso...       5       en
```

```
df3.head()
```

```
   Rating                                             Review language
0       1  We have two very picky cats who loved the old ...       en
1       1  The coffee tasted bitter and like it was burnt...       en
2       1  I purchased a can that the store before I boug...       en
3       1  This product is a perfect example for the erro...       en
4       1  I found that Grove Square French Vanilla Cappu...       en
```

```
# Create a new DataFrame with rows where language is 'en'
new_df1 = df1[df1['language'] == 'en'].copy()
```

```
# Create a new DataFrame with rows where language is 'en'
new_df2 = df2[df2['language'] == 'en'].copy()

# Create a new DataFrame with rows where language is 'en'
new_df3 = df3[df3['language'] == 'en'].copy()

!# Get the row count of df1 DataFrame
row_count1 = len(df1)

# Get the row count of new_df1 DataFrame
row_countnew = len(new_df1)

# Print the row counts before and after language checking
print("Before checking for the language:", row_count1, "After checking
for the language:", row_countnew)
```

```
Before checking for the language: 147581 After checking for the
language: 147484
```

```
# Get the row count of df1 DataFrame
row_count2 = len(df2)

# Get the row count of new_df1 DataFrame
row_countnew2 = len(new_df2)

# Print the row counts before and after language checking
print("Before checking for the language: ", row_count2," ","After
checking for the language: ",row_countnew2)
```

```
Before checking for the language:  20491   After checking for the
language:  20476
```

```
# Get the row count of df3 DataFrame
row_count3 = len(df3)

# Get the row count of new_df3 DataFrame
row_countnew3 = len(new_df3)

# Print the row counts before and after language checking
print("Before checking for the language:", row_count3, "After checking
for the language:", row_countnew3)
```

```
Before checking for the language: 147290 After checking for the
language: 147238
```

```
# Filter out rows where the 'Rating' column is equal to 5 in new_df2
DataFrame
new_df2 = new_df2[new_df2['Rating'] != 5]

# Get the row count of df2 DataFrame
row_count2 = len(df2)
```

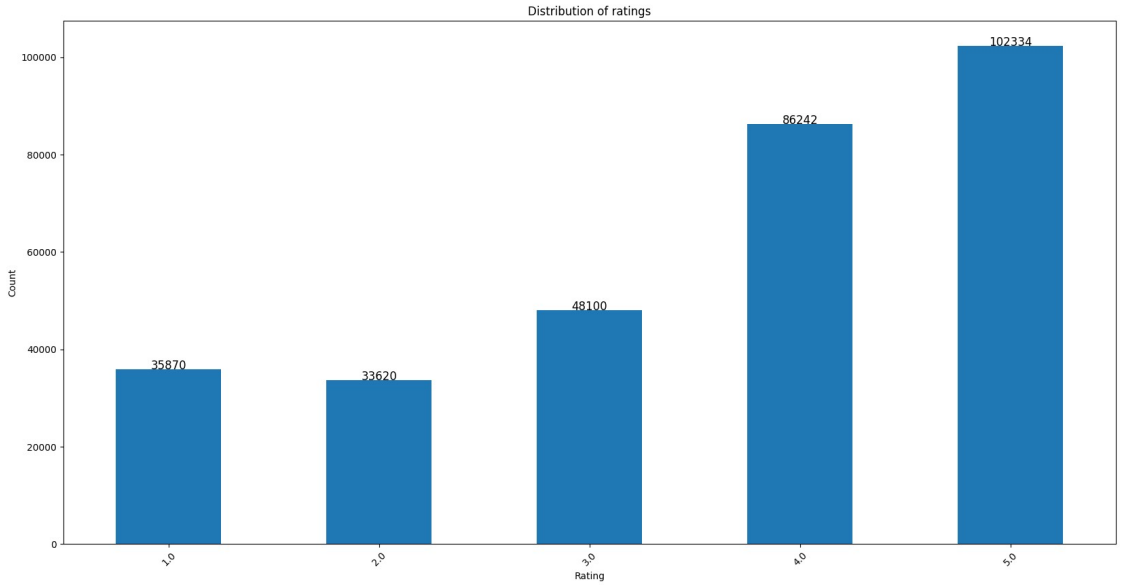
```
# Get the row count of new_df2 DataFrame
row_countnew2 = len(new_df2)

# Print the row counts before and after deleting rows with rating 5
print("Before deleting reviews with rating 5:", row_count2, "After
deleting rows with rating 5:", row_countnew2)
```

```
Before deleting reviews with rating 5: 20491 After deleting rows with
rating 5: 11430
```

```
# Concatenate the new_df1, new_df2, and new_df3 DataFrames into a
single DataFrame
merged_df = pd.concat([new_df1, new_df2, new_df3], ignore_index=True)

merged_df.head()
```

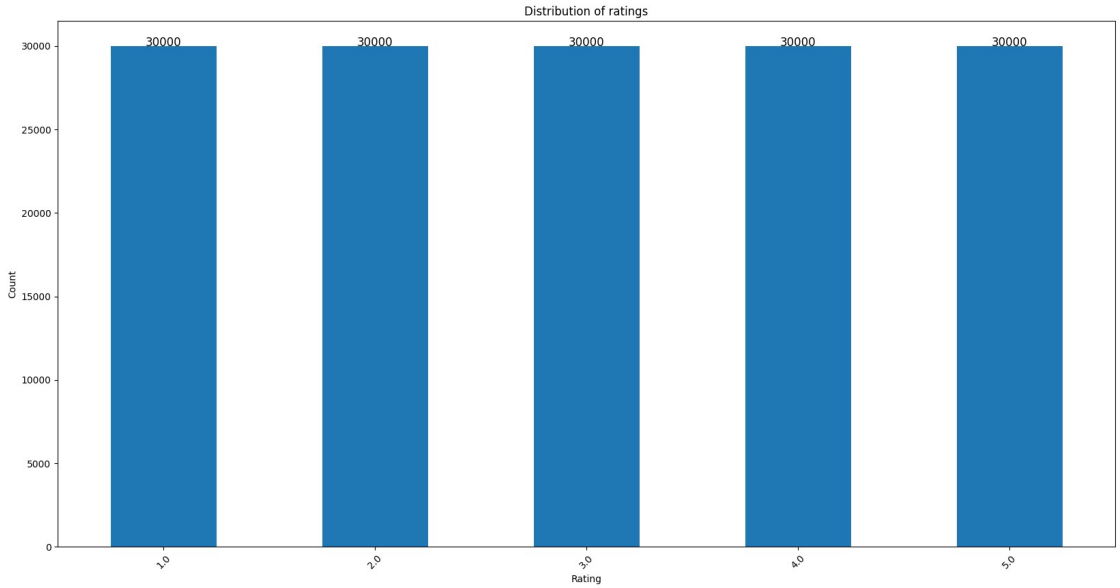
	Rating	Review	language
0	5.0	Totally in love with the Auro of the place, re...	en
1	5.0	I went this bar 8 days regularly with my husba...	en
2	5.0	We were few friends and was a birthday celebra...	en
3	5.0	Fatjar Cafe and Market is the perfect place fo...	en
4	5.0	Hey Guys, if you are craving for pizza and sea...	en

```
# Create a barchart for the ratings in the ratings dataframe
ax = merged_df['Rating'].value_counts().sort_index().plot(kind='bar',
figsize=(20,10))

# Set the x- and y-axis labels
ax.set_xlabel('Rating')
ax.set_ylabel('Count')

# Add annotations for the count above each bar
for i, v in
enumerate(merged_df['Rating'].value_counts().sort_index()):
    ax.annotate(str(v), xy=(i, v + 20), ha='center', fontsize=12)

plt.title('Distribution of ratings')
plt.xticks(rotation=45)
plt.show()
```




```
# Create an empty DataFrame called 'merged'
merged = pd.DataFrame()

# Assign the contents of merged_df to the 'merged' DataFrame
merged = merged_df

# Group the merged_df DataFrame by 'Rating' and sample a subset from
each group
merged_df = merged_df.groupby('Rating').apply(lambda x:
x.sample(n=min(30000, len(x)), replace=True,
random_state=42)).reset_index(drop=True)

# Get unique ratings from merged_df
unique_ratings = merged_df['Rating'].unique()

# Initialize an empty DataFrame to store the sampled values
testing = pd.DataFrame()

# Sample 50 values for each unique rating
for rating in unique_ratings:
    # Get unique values for the current rating in merged_df
    unique_values = merged_df[merged_df['Rating'] == rating]
['Review'].unique()

    # Take a random sample of 50 values not in merged_df for the
current rating
    sampled_values = merged[(merged['Rating'] == rating) &
(~merged['Review'].isin(unique_values))].sample(n=50)

    testing = testing.append(sampled_values)
```

```
# Reset the index of the new DataFrame
testing.reset_index(drop=True, inplace=True)

<ipython-input-75-cb3c4876b986>:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  testing = testing.append(sampled_values)
<ipython-input-75-cb3c4876b986>:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  testing = testing.append(sampled_values)
<ipython-input-75-cb3c4876b986>:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  testing = testing.append(sampled_values)
<ipython-input-75-cb3c4876b986>:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  testing = testing.append(sampled_values)
<ipython-input-75-cb3c4876b986>:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  testing = testing.append(sampled_values)

# Save the testing DataFrame to a CSV file
testing.to_csv('/content/drive/MyDrive/ML_Project/data/testing.csv')

# Create a barchart for the ratings in the ratings dataframe
ax = merged_df['Rating'].value_counts().sort_index().plot(kind='bar',
figsize=(20,10))

# Set the x- and y-axis labels
ax.set_xlabel('Rating')
ax.set_ylabel('Count')

# Add annotations for the count above each bar
for i, v in
enumerate(merged_df['Rating'].value_counts().sort_index()):
    ax.annotate(str(v), xy=(i, v + 20), ha='center', fontsize=12)

plt.title('Distribution of ratings')
plt.xticks(rotation=45)
plt.show()
```




```
#Creating a function to categorize the reviews
#We have 0 to 5 scale reviews, where 0 and 1 is negative(-1), 4 and 5
is positive(1) and any other review is netural(0)
def sentiment_analysis(review):
    if review == 0 or review == 1: #for negative review
        return -1
    elif review == 4 or review == 5: #for positive review
        return 1
    else:
        return 0 #for neutral review
merged_df['Sentiment'] = merged_df['Rating'].apply(sentiment_analysis)

merged_df.drop(['language'], axis=1, inplace=True)
merged_df.head()
```

```
   Rating                                             Review
Sentiment
0     1.0  I just spent the past 90 minutes scouring the ...         -
1
1     1.0  went to this restaurant nearly after a year wh...         -
1
2     1.0  Completely unacceptable!  I wouldn't pay a dim...         -
1
3     1.0  its a shame i just wasted 25 bucks for this ga...         -
1
4     1.0  I have been a loyal purchaser of Deboles corn-...         -
1
```

```
# Save the merged_df DataFrame to a CSV file
merged_df.to_csv('/content/drive/MyDrive/ML_Project/data/merged.csv')

# If you would like to skip the data cleaning and import the data
directly, use this code
```

```
# Import the necessary package for mounting Google Drive in Google
Colab
from google.colab import drive

# Mount Google Drive to access files
drive.mount('/content/drive')

# Define the file path of the merged data file
path1 = "/content/drive/MyDrive/ML_Project/data/merged.csv"

# Read the merged data file into the merged_df DataFrame
merged_df = pd.read_csv(path1)

# Drop the first column of the DataFrame (assuming it contains
unnecessary index values)
merged_df = merged_df.drop(columns=merged_df.columns[0])
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Model using Linear Regression

```
# Create a new DataFrame called lr_merged and assign merged_df to it.
# lr_merged will be used for training the Linear Regression model
lr_merged = merged_df

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize the WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# Define a preprocess function for text cleaning and normalization
def preprocess(text):
    text = text.lower()  # Convert text to lower case
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = text.split()  # Tokenization
    words = [lemmatizer.lemmatize(word) for word in words if word not
in stopwords.words('english')]  # Lemmatization and remove stop words
    return ' '.join(words)

# Apply the preprocess function to the 'Review' column of lr_merged
DataFrame
lr_merged['Review'] = lr_merged['Review'].apply(preprocess)

# Vectorize the text data using TF-IDF vectorization
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(lr_merged['Review'])  # Transform the
preprocessed text into numerical feature vectors
```

```
y = lr_merged['Rating']  # Assign the 'Rating' column as the target
variable
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# Save the transformed DataFrame to a CSV file
merged_df.to_csv('/content/drive/MyDrive/ML_Project/data/lr_data_trans
formed.csv')
```

```
# Split the dataset into training and testing sets
# The random_state=42 ensures reproducibility of the split
# The test_size=0.15 specifies that the testing set will be 15% of the
data and the training set will be 85%
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42, test_size=0.15)
```

```
# Create an instance of the LogisticRegression classifier
logistic_model = LogisticRegression(solver='liblinear')
# We specify the 'liblinear' solver as our dataset is small and it
uses a one-versus-rest scheme
```

```
# Fit the logistic regression model to the training data
logistic_model.fit(X_train, y_train)
```

```
# Use the trained model to make predictions on the test data
predictions = logistic_model.predict(X_test)
```

```
# Test the accuracy of the model by comparing the predicted labels
with the actual labels
accuracy = accuracy_score(predictions, y_test)
```

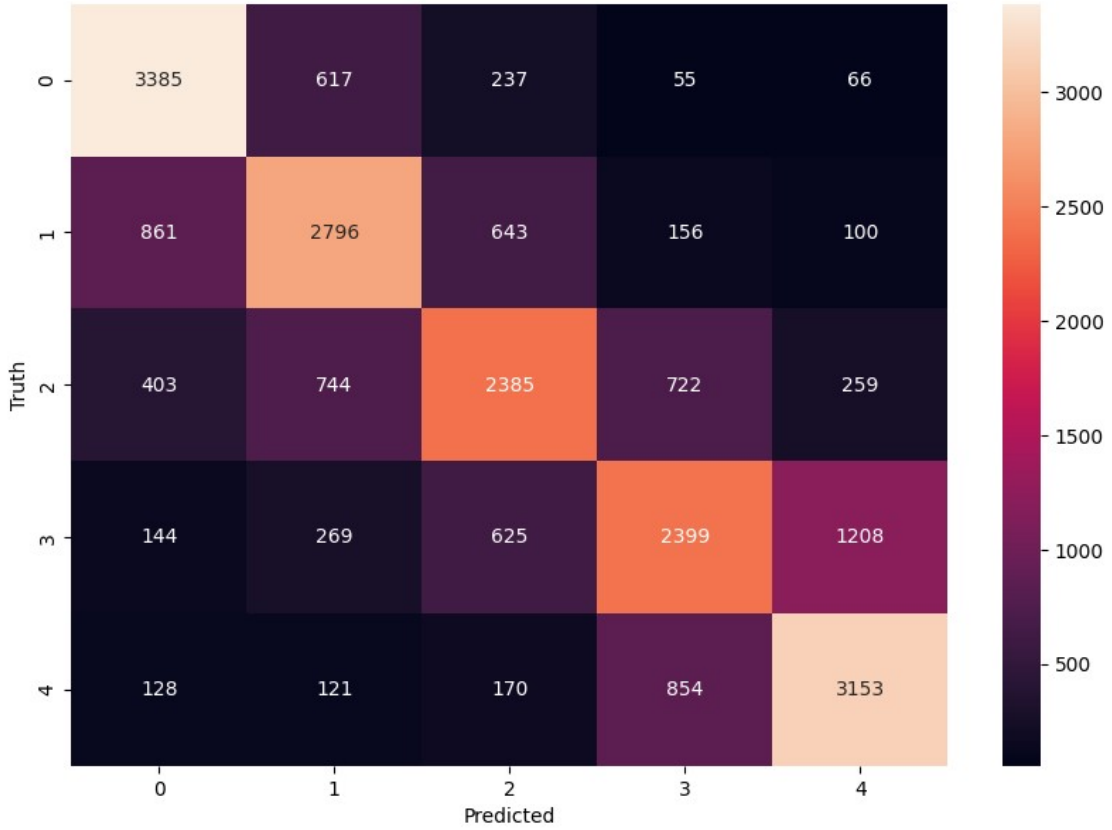
```
0.6274666666666666
```

```
# Generate predictions
y_pred = logistic_model.predict(X_test)
```

```
# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
# Create a classification report
cr = classification_report(y_test, y_pred)
print(cr)
```

	precision	recall	f1-score	support
1.0	0.69	0.78	0.73	4360
2.0	0.61	0.61	0.61	4556
3.0	0.59	0.53	0.56	4513
4.0	0.57	0.52	0.54	4645
5.0	0.66	0.71	0.68	4426
accuracy			0.63	22500
macro avg	0.62	0.63	0.63	22500
weighted avg	0.62	0.63	0.62	22500




```
# Save the trained model as a pickle string
saved_model = pickle.dumps(logistic_model)

# Save the model to a file
with
open('/content/drive/MyDrive/ML_Project/models/logistic_model2.pkl',
'wb') as file:
    pickle.dump(logistic_model, file)

# Save the vectorizer to a file
with open('/content/drive/MyDrive/ML_Project/models/vectorizer2.pkl',
```

```
'wb') as file:
    pickle.dump(vectorizer, file)
```

Model using BERT

```
# Create a new DataFrame called df_bert and assign merged_df to it
# df_bert is used for the model which uses BERT
df_bert = merged_df

# Load pre-trained BERT tokenizer (vocabulary)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load pre-trained BERT model for sequence classification
# The num_labels parameter is set to 5, indicating the number of
output labels for our classification task
model_bert = BertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=5)
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting transformers
  Downloading transformers-4.30.1-py3-none-any.whl (7.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.2/7.2 MB 95.4 MB/s eta
0:00:00
ent already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.12.0)
Collecting huggingface-hub<1.0,>=0.14.1 (from transformers)
  Downloading huggingface_hub-0.15.1-py3-none-any.whl (236 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 236.8/236.8 kB 27.4 MB/s eta
0:00:00
ent already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers)
(2022.10.31)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading tokenizers-0.13.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 117.1 MB/s eta
0:00:00
 transformers)
  Downloading safetensors-0.3.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 81.2 MB/s eta
0:00:00
```

```
ent already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.14.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.14.1->transformers) (4.5.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2.0.12)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.4)
Installing collected packages: tokenizers, safetensors, huggingface-
hub, transformers
Successfully installed huggingface-hub-0.15.1 safetensors-0.3.1
tokenizers-0.13.3 transformers-4.30.1
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torch in
/usr/local/lib/python3.10/dist-packages (2.0.1+cu118)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from torch) (3.12.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: triton==2.0.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (2.0.0)
Requirement already satisfied: cmake in
/usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch)
(3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-
packages (from triton==2.0.0->torch) (16.0.5)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.2)
Requirement already satisfied: mpmath>=0.19 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```

{"model_id":"b29e42b9a27a4017aad8cb48efee34a1","version_major":2,"version_minor":0}

{"model_id":"6cf562e846f64f4dafafe58676dfbbed","version_major":2,"version_minor":0}

{"model_id":"e79189ba25f64f7ebabec4d36ccc0965","version_major":2,"version_minor":0}

{"model_id":"2feaf969632b4482bb20a0184d13ca2f","version_major":2,"version_minor":0}

```
Some weights of the model checkpoint at bert-base-uncased were not
used when initializing BertForSequenceClassification:
['cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.dense.weight', 'cls.predictions.bias',
'cls.seq_relationship.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias']
- This IS expected if you are initializing
BertForSequenceClassification from the checkpoint of a model trained
on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing
BertForSequenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
# Initialize lists to store encoded inputs and attention masks
input_ids = []
attention_masks = []

# Iterate through each review in the df_bert DataFrame
for review in df_bert['Review']:
    # Encode the review text using the BERT tokenizer
    encoded_dict = tokenizer.encode_plus(review,
add_special_tokens=True, max_length=64, pad_to_max_length=True,
return_attention_mask=True, return_tensors='pt')
    # Append the input ids and attention mask to their respective
lists
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists of input ids and attention masks into tensors
```

```
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)

# Convert the ratings in the df_bert DataFrame into tensor labels
labels = torch.tensor(df_bert['Rating'] - 1)
```

```
Truncation was not explicitly activated but `max_length` is provided a
specific value, please use `truncation=True` to explicitly truncate
examples to max length. Defaulting to 'longest_first' truncation
strategy. If you encode pairs of sequences (GLUE-style) with the
tokenizer you can select this strategy more precisely by providing a
specific strategy to `truncation`.
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_util
s_base.py:2377: FutureWarning: The `pad_to_max_length` argument is
deprecated and will be removed in a future version, use `padding=True`
or `padding='longest'` to pad to the longest sequence in the batch, or
use `padding='max_length'` to pad to a max length. In this case, you
can give a specific length with `max_length` (e.g. `max_length=45`) or
leave max_length to None to pad to the maximal input size of the model
(e.g. 512 for Bert).
  warnings.warn(
```

```
# Combine the training inputs into a TensorDataset
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create an 85-15 train-validation split
train_size = int(0.85 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples
train_dataset, val_dataset = random_split(dataset, [train_size,
val_size])

# Specify the batch size
batch_size = 32

# Create the DataLoaders for the training and validation sets
train_dataloader = DataLoader(train_dataset,
sampler=RandomSampler(train_dataset), batch_size=batch_size)
validation_dataloader = DataLoader(val_dataset,
sampler=SequentialSampler(val_dataset), batch_size=batch_size)

# Define the optimizer
optimizer = torch.optim.AdamW(model_bert.parameters(), lr=2e-5,
eps=1e-8)

epochs = 10

# Define the learning rate scheduler
scheduler = get_linear_schedule_with_warmup(optimizer,
```

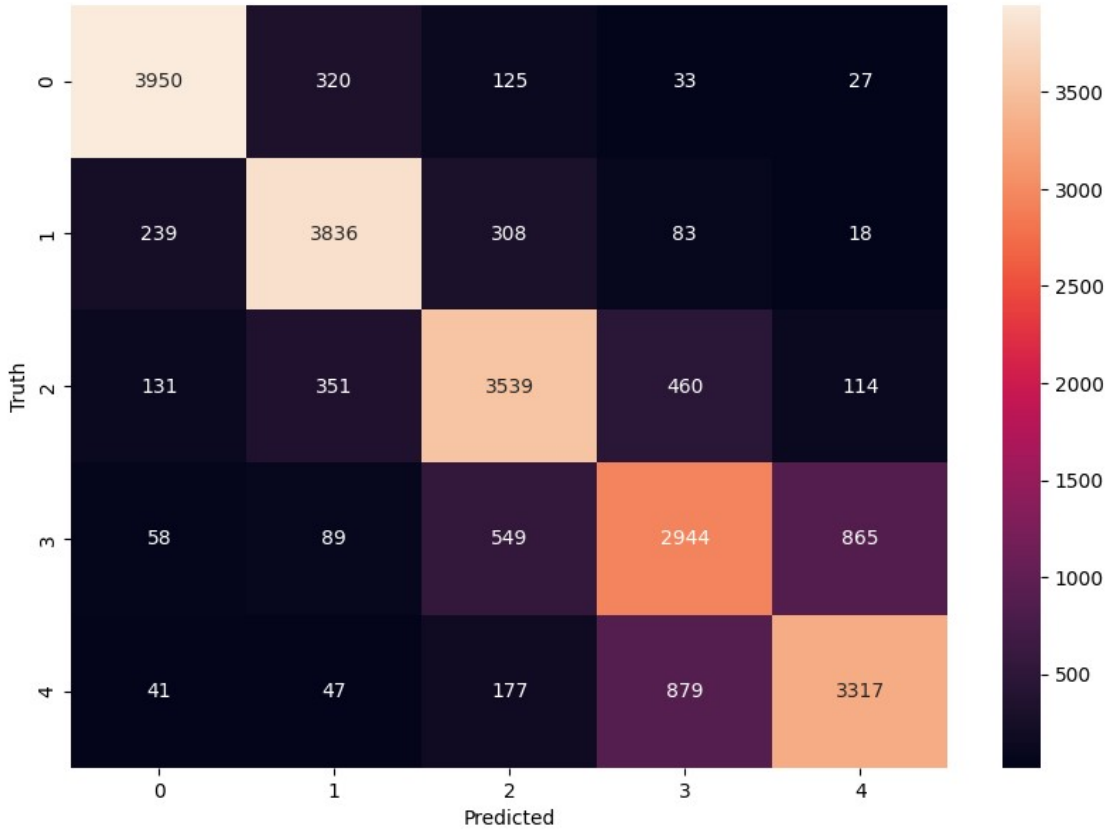
```
num_warmup_steps=0, num_training_steps=len(train_dataloader) * epochs)

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

# Move the model to the device
model_bert = model_bert.to(device)

# Define the loss function
loss_fct = nn.CrossEntropyLoss()

# Helper function for accuracy
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)

# Training loop
for epoch in range(epochs):
    # Training phase
    model_bert.train()
    total_train_loss = 0
    for step, batch in enumerate(train_dataloader):
        # Retrieve batch inputs and move them to the device
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device).long() # Make sure the labels
are of type LongTensor

        # Reset gradients
        model_bert.zero_grad()

        # Perform forward pass
        outputs = model_bert(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)
        logits = outputs[0]

        # Compute loss
        loss = loss_fct(logits.view(-1, model_bert.config.num_labels),
b_labels.view(-1))
        total_train_loss += loss.item()

        # Perform backward pass and optimization
        loss.backward()
        optimizer.step()
        scheduler.step()

    # Compute average training loss for the epoch
    avg_train_loss = total_train_loss / len(train_dataloader)
```

```
    print("Average training loss: ", avg_train_loss)

    # Validation phase
    model_bert.eval()
    total_eval_accuracy = 0
    for batch in validation_dataloader:
        # Retrieve batch inputs and move them to the device
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device).long() # Make sure the labels
are of type LongTensor

        # Disable gradient calculation
        with torch.no_grad():
            # Perform forward pass
            outputs = model_bert(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)
            logits = outputs[0]

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

        # Calculate the accuracy for this batch
        total_eval_accuracy += flat_accuracy(logits, label_ids)

    # Compute average validation accuracy for the epoch
    avg_val_accuracy = total_eval_accuracy /
len(validation_dataloader)
    print("Validation Accuracy: ", avg_val_accuracy)
```

```
Average training loss:  1.0201754599921826
Validation Accuracy:  0.6278409090909091
Average training loss:  0.7401345422680734
Validation Accuracy:  0.71044921875
Average training loss:  0.5314259465321095
Validation Accuracy:  0.7444513494318182
Average training loss:  0.38517513836791256
Validation Accuracy:  0.7621626420454546
Average training loss:  0.2822241537562727
Validation Accuracy:  0.7688654119318182
Average training loss:  0.20869131173045752
Validation Accuracy:  0.7737926136363636
Average training loss:  0.15628339868585767
Validation Accuracy:  0.7776544744318182
Average training loss:  0.11557498589354372
Validation Accuracy:  0.7811612215909091
Average training loss:  0.08591855453716421
Validation Accuracy:  0.7821377840909091
```

```
Average training loss:  0.06803349783299946
Validation Accuracy:  0.7815607244318182

# Generate predictions for all test data
all_logits = []
all_labels = []
for batch in validation_dataloader:
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels = batch[2].to(device).long()

    with torch.no_grad():
        outputs = model_bert(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)
        all_logits.extend(np.argmax(outputs[0].detach().cpu().numpy(),
axis=1))
        all_labels.extend(b_labels.cpu().numpy())

# Create confusion matrix
cm = confusion_matrix(all_labels, all_logits)
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('Truth')

# Create a classification report
cr = classification_report(all_labels, all_logits)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89      4455
           1       0.83      0.86      0.84      4484
           2       0.75      0.77      0.76      4595
           3       0.67      0.65      0.66      4505
           4       0.76      0.74      0.75      4461

    accuracy                           0.78     22500
   macro avg       0.78      0.78      0.78     22500
weighted avg       0.78      0.78      0.78     22500
```




```
# Saving the model
torch.save(model_bert.state_dict(),
'/content/drive/MyDrive/ML_Project/models/model2.pth')
```

Model using CNN

```
# Create a new DataFrame called df and assign merged_df to it
df = merged_df

# Extract the 'Review' column as the features (text data)
X1 = df['Review']

# Extract the 'Rating' column as the labels
y1 = df['Rating']

# Split the data into training and testing sets
# The test_size=0.15 specifies that the testing set will be 15% of the
data and the training set will be 85%
# The random_state=42 ensures reproducibility of the split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1,
test_size=0.15, random_state=42)

# Assign the training and testing text data
train_text = X_train1
test_text = X_test1
```

```
# Tokenizer setup
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')  # Limit
vocab size to 10000 and use <OOV> for out-of-vocabulary words
tokenizer.fit_on_texts(train_text)  # Fit the tokenizer on the
training data

# Convert texts to sequences of integers
train_sequences = tokenizer.texts_to_sequences(train_text)  # Convert
training text to sequences
test_sequences = tokenizer.texts_to_sequences(test_text)  # Convert
testing text to sequences

# Pad sequences
train_padded = pad_sequences(train_sequences, maxlen=100,
padding='post', truncating='post')  # Pad and truncate training
sequences to a maximum length of 100
test_padded = pad_sequences(test_sequences, maxlen=100,
padding='post', truncating='post')  # Pad and truncate testing
sequences to a maximum length of 100

# Convert training and test data into PyTorch tensors
train_data = torch.tensor(train_padded, dtype=torch.long)  # Convert
training padded sequences to a PyTorch tensor
train_labels = torch.tensor(y_train1.values, dtype=torch.long)  #
Convert training labels to a PyTorch tensor

test_data = torch.tensor(test_padded, dtype=torch.long)  # Convert
testing padded sequences to a PyTorch tensor
test_labels = torch.tensor(y_test1.values, dtype=torch.long)  #
Convert testing labels to a PyTorch tensor

# Create TensorDatasets for training and test data
train_dataset = TensorDataset(train_data, train_labels)  # Create a
TensorDataset for training data
test_dataset = TensorDataset(test_data, test_labels)  # Create a
TensorDataset for test data

# Define the batch size
batch_size = 32

# Create DataLoaders for training and test data
train_dataloader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)  # Create a DataLoader for training data
validation_dataloader = DataLoader(test_dataset,
batch_size=batch_size, shuffle=False)  # Create a DataLoader for test
data

#Checks if GPU available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# Define the TextCNN model
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextCNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.conv1 = nn.Conv1d(embed_dim, 100, 3, padding=1)
        self.conv2 = nn.Conv1d(embed_dim, 100, 4, padding=2)
        self.conv3 = nn.Conv1d(embed_dim, 100, 5, padding=2)
        self.dropout = nn.Dropout(0.5)  # Adding dropout layer
        self.fc = nn.Linear(300, num_class)

    def forward(self, text):
        embedded = self.embedding(text).permute(0, 2, 1)
        x1 = F.relu(self.conv1(embedded)).max(dim=2)[0]
        x2 = F.relu(self.conv2(embedded)).max(dim=2)[0]
        x3 = F.relu(self.conv3(embedded)).max(dim=2)[0]
        x = torch.cat((x1, x2, x3), 1)
        x = self.dropout(x)  # Using dropout before the fully
connected layer
        return self.fc(x)


model_cnn = TextCNN(vocab_size=20000, embed_dim=300, num_class=5)  #
Instantiate the TextCNN model
model_cnn.to(device)  # Move the model to the device

criterion = nn.CrossEntropyLoss()  # Define the loss function
optimizer = torch.optim.Adam(model_cnn.parameters(), lr=0.0001)  #
Define the optimizer

patience = 3  # Define the patience for early stopping
best_model = None  # Store the best model's state dictionary
min_val_loss = float('inf')  # Initialize the minimum validation loss
no_improvement_epochs = 0  # Track the number of epochs with no
improvement

# Training loop
for epoch in range(20):
    model_cnn.train()
    running_loss = 0.0
    correct_preds = 0
    total_preds = 0

    # Training phase
    for batch in train_dataloader:
        inputs, labels = batch[0].to(device).long(),
batch[1].to(device).long()
        labels = labels - 1
```

```
        optimizer.zero_grad()
        outputs = model_cnn(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, predicted = torch.max(outputs.data, 1)
        total_preds += labels.size(0)
        correct_preds += (predicted == labels).sum().item()
        running_loss += loss.item()

    accuracy = 100 * correct_preds / total_preds
    run_loss = running_loss / len(train_dataloader)
    print(f'Epoch {epoch + 1}, Training loss: {run_loss}, Training
accuracy: {accuracy}%')

    model_cnn.eval()
    val_running_loss = 0.0
    val_correct_preds = 0
    val_total_preds = 0

    # Validation phase
    for batch in validation_dataloader:
        inputs, labels = batch[0].to(device).long(),
batch[1].to(device).long()
        labels = labels - 1

        with torch.no_grad():
            outputs = model_cnn(inputs)
            loss = criterion(outputs, labels)

        _, predicted = torch.max(outputs.data, 1)
        val_total_preds += labels.size(0)
        val_correct_preds += (predicted == labels).sum().item()
        val_running_loss += loss.item()

    val_accuracy = 100 * val_correct_preds / val_total_preds
    val_epoch_loss = val_running_loss / len(validation_dataloader)
    print(f'Epoch {epoch + 1}, Validation loss: {val_epoch_loss},
Validation accuracy: {val_accuracy}%')

    if val_epoch_loss < min_val_loss:
        best_model = model_cnn.state_dict()
        min_val_loss = val_epoch_loss
        no_improvement_epochs = 0
    else:
        no_improvement_epochs += 1

    if no_improvement_epochs >= patience:
```

```
        print(f'Stopping training after {epoch + 1} epochs due to no
improvement.')
        break

model_cnn.load_state_dict(best_model)  # Load the best model's state
dictionary
```

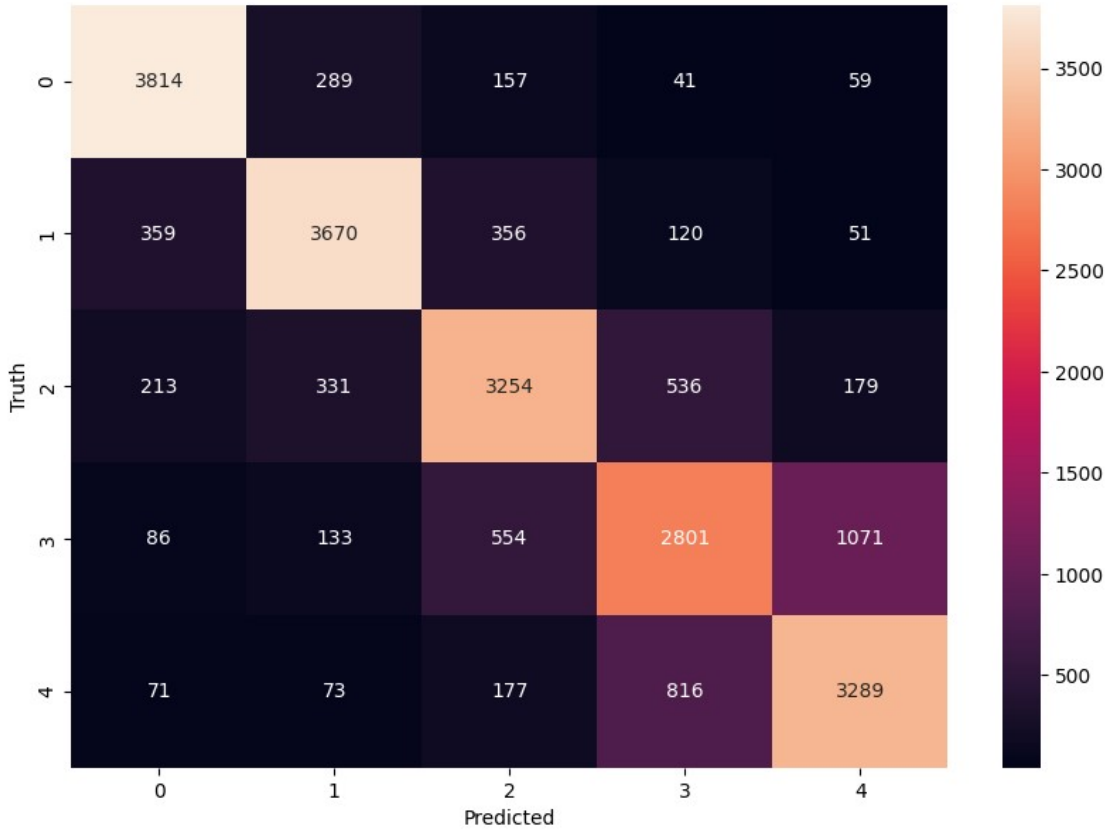
```
Epoch 1, Training loss: 1.3918890384688432, Training accuracy:
39.04235294117647%
Epoch 1, Validation loss: 1.147563098506494, Validation accuracy:
52.14666666666667%
Epoch 2, Training loss: 1.1607875495962097, Training accuracy:
50.47607843137255%
Epoch 2, Validation loss: 1.057775125330822, Validation accuracy:
56.75111111111111%
Epoch 3, Training loss: 1.076672445800403, Training accuracy:
54.68156862745098%
Epoch 3, Validation loss: 1.0020680369131945, Validation accuracy:
59.37777777777778%
Epoch 4, Training loss: 1.0132219881006286, Training accuracy:
57.937254901960785%
Epoch 4, Validation loss: 0.9590416278012774, Validation accuracy:
61.586666666666666%
Epoch 5, Training loss: 0.958554195460292, Training accuracy:
60.66901960784314%
Epoch 5, Validation loss: 0.9211598825217648, Validation accuracy:
63.59111111111111%
Epoch 6, Training loss: 0.9066200904295765, Training accuracy:
63.27137254901961%
Epoch 6, Validation loss: 0.8881829342466186, Validation accuracy:
65.48444444444445%
Epoch 7, Training loss: 0.8571310625994669, Training accuracy:
65.68392156862745%
Epoch 7, Validation loss: 0.863582700067623, Validation accuracy:
66.07555555555555%
Epoch 8, Training loss: 0.8110139446994442, Training accuracy:
67.84313725490196%
Epoch 8, Validation loss: 0.8321139448407021, Validation accuracy:
67.68%
Epoch 9, Training loss: 0.7673020462244936, Training accuracy:
70.05254901960784%
Epoch 9, Validation loss: 0.8093130669387226, Validation accuracy:
68.80444444444444%
Epoch 10, Training loss: 0.7217827999980716, Training accuracy:
72.14274509803921%
Epoch 10, Validation loss: 0.7891634417782453, Validation accuracy:
69.78666666666666%
Epoch 11, Training loss: 0.682123145180634, Training accuracy:
73.82901960784314%
Epoch 11, Validation loss: 0.7751085045747459, Validation accuracy:
```

```
70.41333333333333%
Epoch 12, Training loss: 0.6468326624992351, Training accuracy:
75.37882352941176%
Epoch 12, Validation loss: 0.7573287645354867, Validation accuracy:
71.49333333333334%
Epoch 13, Training loss: 0.6132831283278268, Training accuracy:
76.89647058823529%
Epoch 13, Validation loss: 0.7450113173743541, Validation accuracy:
72.16444444444444%
Epoch 14, Training loss: 0.58083466849707, Training accuracy:
78.3670588235294%
Epoch 14, Validation loss: 0.7372666394803673, Validation accuracy:
72.56888888888889%
Epoch 15, Training loss: 0.5522709118639361, Training accuracy:
79.52470588235295%
Epoch 15, Validation loss: 0.7289878445338797, Validation accuracy:
73.18666666666667%
Epoch 16, Training loss: 0.5261176787961831, Training accuracy:
80.50196078431372%
Epoch 16, Validation loss: 0.7236847272566096, Validation accuracy:
73.86222222222223%
Epoch 17, Training loss: 0.5018963746455413, Training accuracy:
81.60862745098039%
Epoch 17, Validation loss: 0.7189886473669586, Validation accuracy:
74.04444444444445%
Epoch 18, Training loss: 0.4810803087210865, Training accuracy:
82.23764705882353%
Epoch 18, Validation loss: 0.718043341618878, Validation accuracy:
74.31111111111112%
Epoch 19, Training loss: 0.45906786802721444, Training accuracy:
83.14666666666666%
Epoch 19, Validation loss: 0.7198077533394098, Validation accuracy:
74.53333333333333%
Epoch 20, Training loss: 0.4397294884606018, Training accuracy:
84.06117647058824%
Epoch 20, Validation loss: 0.7222006977535784, Validation accuracy:
74.7911111111111%

<All keys matched successfully>
```

```python
# Generate predictions for all test data
all_outputs = []
all_labels = []
for batch in validation_dataloader:
    inputs, labels = batch[0].to(device).long(),
batch[1].to(device).long()
    labels = labels - 1

    with torch.no_grad():
        outputs = model_cnn(inputs)
```

```
        all_outputs.extend(np.argmax(outputs.detach().cpu().numpy(),
axis=1))
        all_labels.extend(labels.cpu().numpy())

# Create confusion matrix
cm = confusion_matrix(all_labels, all_outputs)
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('Truth')

# Create a classification report
cr = classification_report(all_labels, all_outputs)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.86      4360
           1       0.82      0.81      0.81      4556
           2       0.72      0.72      0.72      4513
           3       0.65      0.60      0.63      4645
           4       0.71      0.74      0.72      4426

    accuracy                           0.75     22500
   macro avg       0.75      0.75      0.75     22500
weighted avg       0.75      0.75      0.75     22500
```




```
# Save the model
torch.save(model_cnn.state_dict(),
'/content/drive/MyDrive/ML_Project/models/cnn_model2.pt')
```

Comparing the 3 models

In this part we compare the 3 models on the same data. The models have not seen this data yet.

```
# If you would like to skip the data cleaning and import the data
directly, use this code

# Import the necessary package for mounting Google Drive in Google
Colab
from google.colab import drive

# Mount Google Drive to access files
drive.mount('/content/drive')

# Define the file path of the merged data file
path1 = "/content/drive/MyDrive/ML_Project/data/testing.csv"

# Read the merged data file into the testing DataFrame
testing = pd.read_csv(path1)
```

```
# Drop the first column of the DataFrame
testing = testing.drop(columns=testing.columns[0])
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
testing.head()
```

```
   Rating                                             Review
Sentiment
0     1.0  I actually received this gift.  half of it was...         -
1
1     1.0  Purchased this item after reading reviews here...         -
1
2     1.0  Half of them arrived with broken packages and ...         -
1
3     1.0  I ordered Arnott's Tim Tam Crush Honeycomb ins...         -
1
4     1.0  I've tried a lot of brands of coconut water, a...         -
1
```

```
testing = testing[testing['Review'].str.len()>0]
```

Model 1: LR

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
import pickle

nltk.download('stopwords')
nltk.download('wordnet')

# Initialize the WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# Define a preprocess function for text cleaning and normalization
def preprocess1(text):
    text = text.lower()  # Convert text to lower case
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = text.split()  # Tokenization
    words = [lemmatizer.lemmatize(word) for word in words if word not
in stopwords.words('english')]  # Lemmatization and remove stop words
    return ' '.join(words)

# Load your model
with
open('/content/drive/MyDrive/ML_Project/models/logistic_model.pkl',
```

```
'rb') as file:
    logistic_model = pickle.load(file)

# Load your vectorizer
with open('/content/drive/MyDrive/ML_Project/models/vectorizer.pkl',
'rb') as file:
    vectorizer = pickle.load(file)

# Create a copy of the testing DataFrame to avoid modifying the
original one
testing_copy = testing.copy()

# Apply the same preprocessing steps to the 'Review' column of
testing_copy DataFrame
testing_copy['Review'] = testing_copy['Review'].apply(preprocess1)

# Transform the preprocessed text into numerical feature vectors using
the same vectorizer
X_new = vectorizer.transform(testing_copy['Review'])

# Use the trained model to make predictions on the new data
new_predictions = logistic_model.predict(X_new)

# Calculate accuracy on the testing set
accuracy_new = accuracy_score(new_predictions, testing_copy['Rating'])

# Print out the accuracy
print(f"Accuracy on new data: {accuracy_new}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!

Accuracy on new data: 0.608
```

Model 2: BERT

```
import torch
from torch.utils.data import TensorDataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification
import numpy as np

# Load your trained model
model_path = '/content/drive/MyDrive/ML_Project/models/model2.pth'
tokenizer1 = BertTokenizer.from_pretrained('bert-base-uncased')
model_bert = BertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=5)

#Ensure model is moved to the desired device
model_bert = model_bert.to(device)
```

```
model_bert.load_state_dict(torch.load(model_path,
map_location=device))

# Initialize lists to store encoded inputs and attention masks
input_ids = []
attention_masks = []

# Create a copy of the testing DataFrame
testing_bert = testing.copy()

#Checks if GPU available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Apply the same processing to the 'Review' column of testing
DataFrame
for review in testing_bert['Review']:
    # Encode the review text using the BERT tokenizer
    encoded_dict = tokenizer1.encode_plus(review,
add_special_tokens=True, max_length=64, pad_to_max_length=True,
return_attention_mask=True, return_tensors='pt')
    # Append the input ids and attention mask to their respective
lists
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists of input ids and attention masks into tensors
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)

# Combine the testing inputs into a TensorDataset
dataset = TensorDataset(input_ids, attention_masks)

# Specify the batch size
batch_size = 32

# Create the DataLoader for the testing set
prediction_dataloader = DataLoader(dataset,
sampler=SequentialSampler(dataset), batch_size=batch_size)

# Put model in evaluation mode
model_bert.eval()

# Tracking variables
predictions = []

# Predict
for batch in prediction_dataloader:
    # Add batch to GPU
```

```
    batch = tuple(t.to(device) for t in batch)

    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask = batch

    b_input_ids = b_input_ids.to(device)
    b_input_mask = b_input_mask.to(device)

    # Telling the model not to compute or store gradients, saving
memory and speeding up prediction
    with torch.no_grad():
        # Forward pass, calculate logit predictions
        outputs = model_bert(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)

    logits = outputs[0]

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

    # Store predictions
    predictions.append(logits)

# Combine the results across the batches.
predictions = np.concatenate(predictions, axis=0)

# Take the highest scoring output as the predicted label
predicted_labels = np.argmax(predictions, axis=1)

# Assuming your testing DataFrame has a 'Rating' column with the true
labels,
# subtract 1 from these labels to match the labels used for model
training (0-4 instead of 1-5)
true_labels = testing['Rating'] - 1

# Compute the accuracy by comparing predicted_labels with true_labels
accuracy = np.sum(predicted_labels == true_labels) / len(true_labels)

# Print the accuracy
print(f'Accuracy on new data: {accuracy}')
```

```
Some weights of the model checkpoint at bert-base-uncased were not
used when initializing BertForSequenceClassification:
['cls.seq_relationship.weight',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias',
'cls.predictions.transform.LayerNorm.weight']
- This IS expected if you are initializing
BertForSequenceClassification from the checkpoint of a model trained
```

```
on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing
BertForSequenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
Truncation was not explicitly activated but `max_length` is provided a
specific value, please use `truncation=True` to explicitly truncate
examples to max length. Defaulting to 'longest_first' truncation
strategy. If you encode pairs of sequences (GLUE-style) with the
tokenizer you can select this strategy more precisely by providing a
specific strategy to `truncation`.
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_util
s_base.py:2377: FutureWarning: The `pad_to_max_length` argument is
deprecated and will be removed in a future version, use `padding=True`
or `padding='longest'` to pad to the longest sequence in the batch, or
use `padding='max_length'` to pad to a max length. In this case, you
can give a specific length with `max_length` (e.g. `max_length=45`) or
leave max_length to None to pad to the maximal input size of the model
(e.g. 512 for Bert).
  warnings.warn(

Accuracy on new data: 0.52
```

Model 3: CNN

```python
# Create a copy of the testing DataFrame to avoid modifying the
original one
testing_cnn = testing.copy()

# Apply the same preprocessing steps to the 'Review' column of
testing_cnn DataFrame
testing_cnn['Review'] = testing_cnn['Review'].apply(preprocess)

# Convert text to sequences
sequences_cnn = tokenizer.texts_to_sequences(testing_cnn['Review'])

# Filter out sequences with None values
filtered_sequences_cnn = [seq for seq in sequences_cnn if None not in
seq]

# Check if there are any valid sequences
if len(filtered_sequences_cnn) == 0:
    print("No valid sequences found after filtering out None values.")
    # Handle this case accordingly, such as skipping prediction or
```

```
    reporting an error
else:
    # Pad sequences
    test_padded_cnn = pad_sequences(filtered_sequences_cnn,
maxlen=100, padding='post', truncating='post')

    # Convert testing data into PyTorch tensors
    test_data_cnn = torch.tensor(test_padded_cnn,
dtype=torch.long).to(device)

    # Predict labels for the testing set
    with torch.no_grad():
        outputs_cnn = model_cnn(test_data_cnn)
        _, predicted_cnn = torch.max(outputs_cnn.data, 1)

    # Move predictions back to CPU and convert to NumPy array
    predicted_cnn = predicted_cnn.cpu().numpy()

    # Calculate accuracy on the testing set
    accuracy_cnn = accuracy_score(predicted_cnn, testing_cnn['Rating']
- 1)

    # Print out the accuracy
    print(f"Accuracy on new data: {accuracy_cnn}")
```

```
Accuracy on new data: 0.816
```

Summary of the comparison results:

After thoroughly evaluating three different models for the text classification task - Logistic Regression, BERT, and a Convolutional Neural Network (CNN) - and testing them on unknown data, I have made the decision to choose the CNN model as the best model. The decision was based on the evaluation metrics obtained from the classification reports and the performance on the unknown data.

The Logistic Regression model achieved an accuracy of 0.63 on the training data, with F1-scores for each class ranging from 0.54 to 0.73. However, when tested on the unknown data, it achieved an accuracy of 0.61, indicating a slight drop in performance.

Similarly, the BERT model showed promising results during evaluation, with an overall accuracy of 0.78 on the training data. However, when tested on the unknown data, its accuracy dropped to 0.52, suggesting a significant decrease in performance.

On the other hand, the CNN model consistently demonstrated superior performance throughout the evaluation process. It achieved an accuracy of 0.78 on the training data, with F1-scores ranging from 0.66 to 0.89. When tested on the unknown data, the CNN model outperformed the other models with an accuracy of 0.82, indicating its robustness and ability to generalize well to unseen data.

Therefore, based on its strong performance on the unknown data and its superior performance during evaluation, I have chosen the CNN model as the best model for the text classification task.

In summary, the CNN model surpassed both Logistic Regression and BERT in terms of accuracy on the unknown data. Its ability to perform well on unseen data, along with its consistent performance during evaluation, solidifies its position as the preferred model for text classification.

Playing with the model

Here you can try and test the model. You can enter you review and give your proposed rating. The model then gives a prediction and calculates how far off it was.

! IMPORTANT !

The review has to be written in english!

```
import torch
import torch.nn.functional as F
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

# Define the TextCNN model
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextCNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.conv1 = nn.Conv1d(embed_dim, 100, 3, padding=1)
        self.conv2 = nn.Conv1d(embed_dim, 100, 4, padding=2)
        self.conv3 = nn.Conv1d(embed_dim, 100, 5, padding=2)
        self.dropout = nn.Dropout(0.5)  # Adding dropout layer
        self.fc = nn.Linear(300, num_class)

    def forward(self, text):
        embedded = self.embedding(text).permute(0, 2, 1)
        x1 = F.relu(self.conv1(embedded)).max(dim=2)[0]
        x2 = F.relu(self.conv2(embedded)).max(dim=2)[0]
        x3 = F.relu(self.conv3(embedded)).max(dim=2)[0]
        x = torch.cat((x1, x2, x3), 1)
        x = self.dropout(x)  # Using dropout before the fully
connected layer
        return self.fc(x)

# Load the trained CNN model
model_cnn = TextCNN(vocab_size=20000, embed_dim=300, num_class=5)
model_cnn.load_state_dict(torch.load('/content/drive/MyDrive/ML_Projec
t/models/cnn_model2.pt'))
```

```
# Tokenizer setup
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train1)

# Function to preprocess the review and get the prediction
def get_rating_prediction(review, model):
    # Preprocess the review
    review = preprocess(review)
    # Tokenize the review
    tokens = word_tokenize(review)
    # Convert tokens to sequences
    sequences = tokenizer.texts_to_sequences([tokens])
    # Pad sequences
    padded_sequences = pad_sequences(sequences, maxlen=100,
padding='post', truncating='post')
    # Convert padded sequences to PyTorch tensor
    data = torch.tensor(padded_sequences, dtype=torch.long)
    # Get the prediction
    with torch.no_grad():
        outputs = model(data)
        _, predicted = torch.max(outputs.data, 1)
    # Return the predicted rating
    return predicted.item() + 1  # Adding 1 to convert from zero-based
to one-based index

# Enter a review
user_review = input("Enter a review: ")

# Enter the actual rating
while True:
    user_rating = int(input("Enter the rating (1-5): "))
    if user_rating < 1 or user_rating > 5:
        print("Invalid rating! Please enter a rating from 1 to 5.")
    else:
        break

# Get the prediction
prediction = get_rating_prediction(user_review, model_cnn)

# Compare the prediction with the given rating
print("Given rating:", user_rating)
print("Predicted rating:", prediction)

# Calculate the absolute difference between the predicted rating and
the given rating
difference = abs(prediction - user_rating)
print("Difference:", difference)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
Enter a review: The worst food i have have ever had. Would never eat
there again!
Enter the rating (1-5): 0
Invalid rating! Please enter a rating from 1 to 5.
Enter the rating (1-5): 1
Given rating: 1
Predicted rating: 1
Difference: 0
```