CIS 9760

Big Data Technologies

Data Engineering Project

Yunus Khan

khan.yunus@baruchmail.cuny.edu

Milestone 1: Proposal

Dataset Description

The dataset for this milestone consists of 5.5 million records of IMDb movie and TV show reviews, sourced from Kaggle. It includes 1.7 million unique users and over 450,000 shows, offering diverse perspectives across various content types. Each review contains metadata such as the reviewer's identity, the movie name, the review summary, the review details, the rating (out of 10), and whether the review contains spoilers. The dataset also includes information about how many users found the review helpful. This dataset is publicly available and provides a rich source of textual and numerical data for analysis and prediction tasks.

Dataset URL: IMDb Reviews Dataset on Kaggle

Dataset Attributes (Columns)

Attribute	Description
review_id	Unique identifier for each review.
reviewer	Public username of the reviewer.
movie	Name of the movie or TV show.
rating	Rating given by the user (out of 10).
review_summary	Short summary of the review.
review_date	Date when the review was posted.
spoiler_tag	Binary indicator (1 = spoiler, 0 = non-spoiler).
review_detail	Full text of the review.
helpful	Number of users who found the review helpful.

Prediction Task

Instead of predicting the exact rating, this milestone reframes the task as a binary classification problem: determining whether a review is positive (rating ≥ 5) or negative (rating < 5). The target label (label) is set to 1 for positive reviews and 0 for negative reviews.

Model Selection

For this classification task, we employ Logistic Regression due to its interpretability and efficiency on large datasets. We also compare performance with an alternative such as Random Forest to ensure robust model selection and optimize classification accuracy.

Expected Outcome

This milestone will deliver a binary classification model that identifies positive reviews from textual summaries and metadata. The results can help streaming platforms or marketplaces automatically surface favorable feedback, enhance recommendation engines, and streamline content moderation processes.

Milestone 2: Data Acquisition

In this stage I will use a Virtual Machine (VM) instance in GCP Compute Engine and write SSH scripts to Download the dataset into the VM. I will create a bucket inside GCP Cloud Storage and create folder as follows, landing/, cleaned/, trusted/, code/ and models/. Finally, I will upload the dataset from the VM to the bucket in the landing/ folder. Step by step code is shown in Appendix A. Below are screenshots of the bucket in Cloud Storage.

Milestone 3: EDA and Data Cleaning

Exploratory Data Analysis

The Exploratory Data Analysis (EDA) conducted on the IMDb reviews dataset provided a thorough examination of the dataset's structure and key statistics. This part only summarizes for 1/6 files of the dataset, Analysis to be continued to understand further in next stages. The preliminary code in Appendix B includes analysis for all files in the dataset.

The dataset comprises 5.5 M records and 11 columns, containing information such as `review_id`, `reviewer`, `movie`, `rating`, and `review_summary`. Among the key findings, it was observed that the `rating` column had significant missing values, which should be addressed through imputation or removal of rows. Additionally, there were duplicate records, which will need to be removed for data integrity.

The `review_summary_count` column, which represents the word count in the review summary, has a wide range from 0 to 570 words, with an average of 5.12 words.

The EDA also highlighted the distribution of key columns. For instance, the `rating` column has a mean value of 6.7 and a standard deviation of 3.1, indicating a spread of ratings around the middle of the scale, ranging from 1 to 10. Most reviews do not contain spoilers, as evidenced by the `spoiler_tag` column, which has a mean value of 0.19. The `review_date` column spans from April 1, 2014, to September 9, 2020, reflecting a diverse range of review dates. The `movie` column contains 164,204 unique movie names, showcasing a wide variety of films being reviewed by users.

Visualisaion

The categorical columns were analyzed using visualizations such as count plots for `spoiler_tag` and `review_summary_count`, revealing imbalances like the majority of reviews not containing spoilers. Numeric columns such as `rating`, `helpful_upvotes`, and `review_summary_count` were visualized using histograms, helping to uncover distribution patterns. Some visualizations are shown below for the first file in the dataset.

Distribution of spoiler_tag

Distribution of review_summary_count
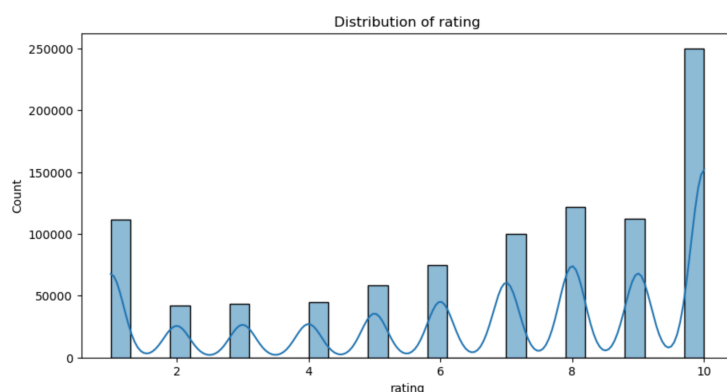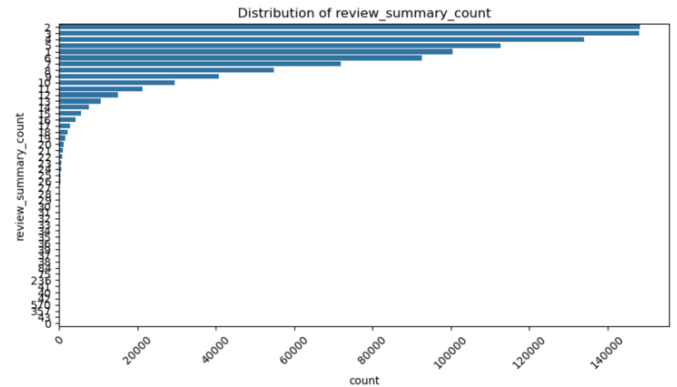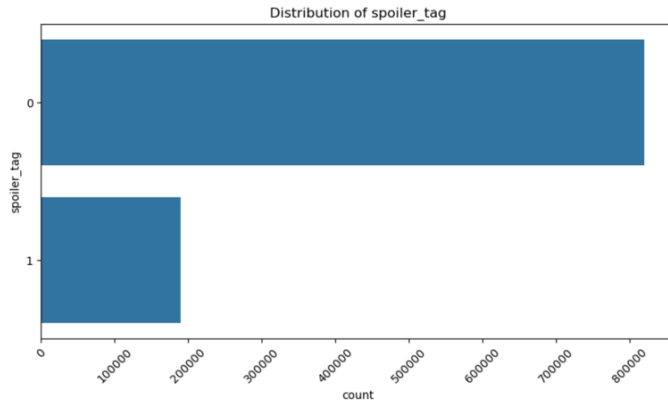
Cleaning

Before any modeling or feature transformations, raw JSON review files were ingested from the Google Cloud Storage **landing** zone and subjected to a multi-step cleaning pipeline to ensure data quality and consistency:

1. **Data Ingestion:** Connected to GCS via the Python storage.Client API, enumerated all blobs under landing/, and read each JSON file into a Pandas DataFrame for initial processing.
2. **Splitting and Casting:** Decomposed the composite helpful column into separate integer fields helpful_upvotes and helpful_total_votes. Irrelevant fields were dropped, and remaining columns were cast to appropriate data types (e.g., strings to Pandas string, numeric fields to Int64, dates to datetime).
3. **Error Handling & Filtering:** Coerced invalid review_date entries to NaT and removed any resulting null rows. Applied a regular expression filter to retain only true movie reviews (titles ending in a four-digit year). Ratings outside the valid 1–10 range were excluded.
4. **Deduplication & Completeness:** Dropped duplicate records to avoid bias, then eliminated any rows containing remaining null values across selected columns.
5. **Parquet Serialization:** Wrote the fully cleaned and typed DataFrame back to GCS in Parquet format under the /cleaned/ folder, creating a partitioned, schema-enforced dataset ready for downstream feature engineering.

Results:

Number of records after cleaning: **3,803,589**

Figure below shows the data frame snippet after cleaning.

```
Processing landing/part-01.json
```

	movie	rating	review_summary	review_detail	review_date	spoiler_tag	helpful_upvotes	helpful_total_votes
4	The Droving (2020)	2.00	An honest review	Here's the truth. There's not much to this mov...	2020-05-03	0	26	41
5	All About Eve (1950)	10.00	Amazing	Having seen this film for the first time today...	2020-05-03	0	0	1
6	Runaway Train (1985)	7.00	Impressive action scenes!	The movie had some very impressive scenes. Esp...	2020-05-03	0	0	1
8	The Half of It (I) (2020)	4.00	Needed the other half of the movie to cover up...	I see that Netflix has a teenage/kids audience...	2020-05-03	0	16	26
10	Closure (I) (2018)	9.00	Fun and intriguing	This is a fun and intriguing mystery. The acti...	2020-05-03	0	2	2
11	Unstoppable (2010)	8.00	Excellent last film of legendary Director Tony...	A suspenseful thrilling adventure about a loos...	2020-05-03	0	3	4
13	Beastie Boys Story (2020)	3.00	The apology tour	A lot of excuses and apologizing for their pol...	2020-05-03	0	8	20

Files saved in cleaning folder as parquet files.

	Name	Size	Type	Created ⑦
☐	📄 part-01.parquet	402 MB		Apr 21, 2025, 1:47:26 PM
☐	📄 part-02.parquet	515.7 MB		Apr 21, 2025, 1:48:26 PM
☐	📄 part-03.parquet	565.5 MB		Apr 21, 2025, 1:49:38 PM
☐	📄 part-04.parquet	536.3 MB		Apr 21, 2025, 1:50:44 PM
☐	📄 part-05.parquet	792.2 MB		Apr 21, 2025, 1:52:11 PM
☐	📄 part-06.parquet	190 MB		Apr 21, 2025, 1:52:40 PM

Buckets > my-bigdata-project-ky > cleaned 📋

Create folder   Upload ▾   Transfer data ▾   Other services ▾

Filter by name prefix only ▾   ≡ Filter   Filter objects and folders

Milestone 4: Feature Engineering and Modeling

Feature Engineering

This milestone emphasizes feature engineering as the foundation for building an effective machine learning model. we worked with a large Parquet dataset by adding more worker nodes and adjusting TF-IDF settings to prevent memory errors, chose TF-IDF over slower Python UDFs for text processing, turned the 1–10 ratings into a simple positive/negative label (5 or above = positive), filtered out very short reviews to cut down on noise, created a helpfulness ratio and scaled all numeric features so no single number dominated, picked logistic regression because it trains quickly and its results are easy to understand, and wrapped everything into a single cloud shell command so the whole process can run automatically.

The goal of this model is to predict whether a user will give a movie a positive rating (defined as 5 stars or higher). This binary classification approach enables the platform to:

- Automatically flag positive reviews for recommendation systems.
- Identify useful user sentiment even in the absence of a full rating.

The prediction task is framed as a binary classification problem, where the target variable (label) is set to 1 if the user gives a rating ≥ 5 and 0 otherwise.

Results

Prediction Result: **0.8425** (AUC Score), Sample Predictions first 20 is shown below.

Review Summary	Label	Prediction
Fun homemade documentary about the Appalachian Trail	1.0	1.0
Bad	0.0	0.0
Bad MOOOOVIEEEEEEEEEEEE	0.0	0.0
Incredibly Boring	0.0	0.0
More than worest	0.0	1.0
Pay attention	0.0	0.0
don't waste your time for this cheap sh....	0.0	0.0
Dissapointed K & Zombie movies fan	0.0	1.0
Lame Waste Of Time	0.0	0.0
Tom and Jerry again	0.0	1.0
Absolutely ridiculous	0.0	0.0
Don't waste your time on this movie	0.0	0.0
Dumb and Dumber	0.0	0.0
Sad to see	0.0	1.0
Stupid from start to finish	0.0	0.0
A demonstration of what not to do in case there is a real zombie apocalypse.	0.0	1.0
#IdentityRouting	1.0	1.0

Bland Zombie Thriller.	1.0	1.0
Not bad but could be way better	1.0	1.0
One more Zombies movie	1.0	1.0

Features Used

Column Name	Type	Role	Feature Engineering Treatment
`review_summary`	string	Text Feature	Tokenization → Stopword Removal → TF-IDF
`review_detail`	string	Filter Logic	Used for filtering based on length/word count
`review_detail_wordcount`	numeric	Feature	Used directly as numerical input
`helpful_upvotes`	numeric	Feature	Used directly; cast to double
`helpful_total_votes`	numeric	Feature	Used directly; cast to double
`spoiler_tag`	numeric	Feature	Used directly (binary flag)
`review_date`	date	Metadata	Extracted `review_year`, `review_month`, `review_yearmonth`
`rating`	float	Label Source	Converted to binary classification label

Process Overview

1. **Data Loading**: Parquet files were read from the `/cleaned` folder in GCS.
2. **Filtering**: Rows missing key columns (`rating`, `review_detail`, or `review_summary`) were removed. Short reviews were filtered out.
3. **Feature Engineering**:
   - Tokenized and cleaned the `review_summary` field.
   - Applied `HashingTF` and `IDF` to extract `text_features`.
   - Engineered numerical features from metadata.
4. **Vectorization**: Combined text + numerical features into a single `features` column using `VectorAssembler`.
5. **Modeling**: Used `LogisticRegression` to predict if the rating is 5 or more.
6. **Evaluation**: Evaluated on test data using AUC (Area Under ROC Curve).
7. **Saving Outputs**: Feature-engineered data, predictions, and model were saved to `/trusted` and `/models` folders.

Output Files

The following files were saved as part of the pipeline execution:

- `trusted/feature_engineered_data.parquet`: Cleaned and engineered features
- `trusted/predictions.parquet`: Prediction results on the test set
- `models/LogisticRegression_model`: Trained binary classification model

Screenshots of the file structure in GCS:

Screenshot of trusted/ Folder

Screenshot of models/ Folder

Codes saved to code/ folder

Milestone 5: Visualization

In this phase of the project, we focus on generating meaningful visualizations to interpret and evaluate the prediction results of our logistic regression model built on IMDb movie review data. The primary goal is to uncover patterns in sentiment classification, explore feature importance, and derive insights from the cleaned and feature-engineered dataset. Using tools such as PySpark for modeling and Matplotlib/Seaborn for visualization, we present a comprehensive view of model behavior, review patterns, and rating trends over time. These visualizations not only validate model performance but also offer interpretability crucial for drawing data-driven conclusions.

Confusion Matrix
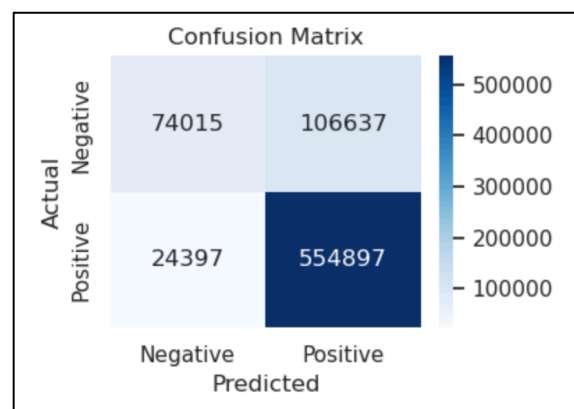
The confusion matrix provides a clear breakdown of how well the model performs in terms of classification accuracy. The model correctly classified 74,015 negative reviews and 554,897 positive reviews. However, it also misclassified 106,637 negative reviews as positive (false positives), and 24,397 positive reviews as negative (false negatives), suggesting a moderate bias toward predicting positive sentiment.

ROC Curve
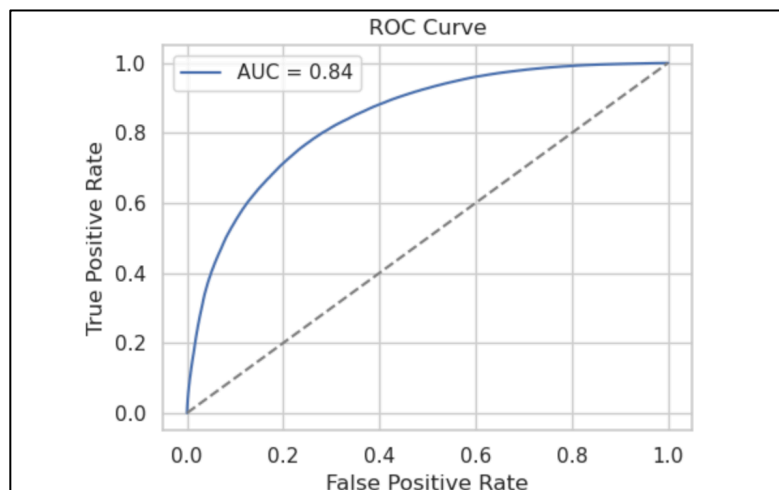
This ROC curve illustrates the model's ability to separate positive and negative reviews using probability scores. The Area Under the Curve (AUC) is 0.84, indicating strong discriminative performance. Unlike other plots based on hard class labels, this curve reflects the full range of prediction probabilities and better aligns with the model's evaluation during testing. The curve rises sharply and stays above the diagonal baseline, confirming that the model effectively distinguishes between classes across various thresholds.

Top 10 Most Reviewed Movies

Top 10 Most Reviewed Movies

Movie Title	Number of Reviews
Avengers: Endgame (2019)	8641
Dil Bechara (2020)	7733
The Shawshank Redemption (1994)	7701
Captain Marvel (2019)	7011
The Dark Knight (2008)	6584
小丑 (2019)	6435
Star Wars: Episode VIII - The Last Jedi (2017)	6311
Mrs. Serial Killer (2020)	5347
Wonder Woman 1984 (2020)	5238
STAR WARS／最後の決闘 (2019)	5036
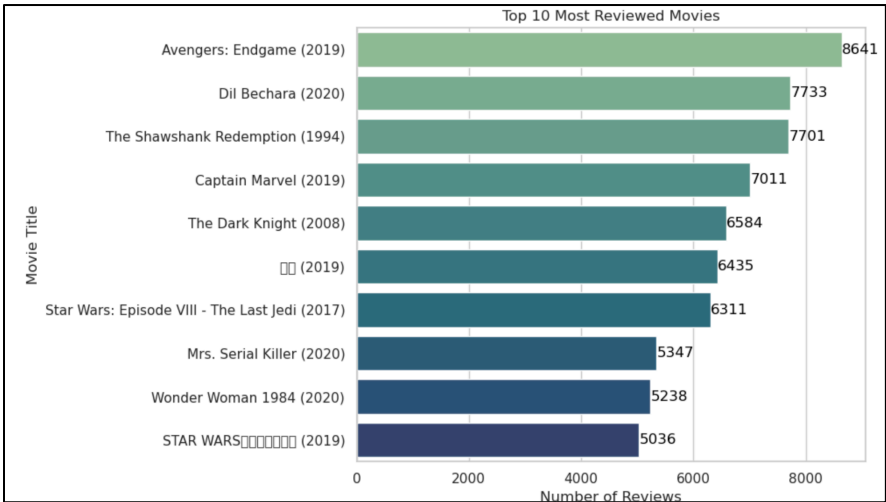
This horizontal bar chart shows the top 10 movies with the most user reviews in the dataset. Popular and widely released films like *Avengers: Endgame* and *The Shawshank Redemption* dominate the list, highlighting the influence of blockbuster status and cultural relevance on review volume.

Average Rating Over Time
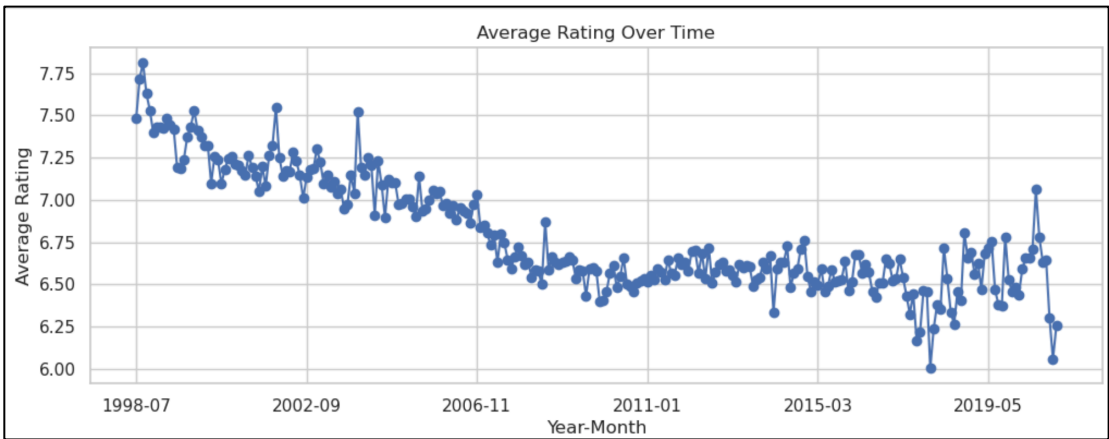
This line chart displays the trend of average IMDb ratings over time, grouped by year and month. A gradual decline is visible across the years, possibly reflecting changing audience expectations, review behavior, or shifts in the types of content produced over time.

Count of Predicted Sentiments

Count of Predicted Sentiments
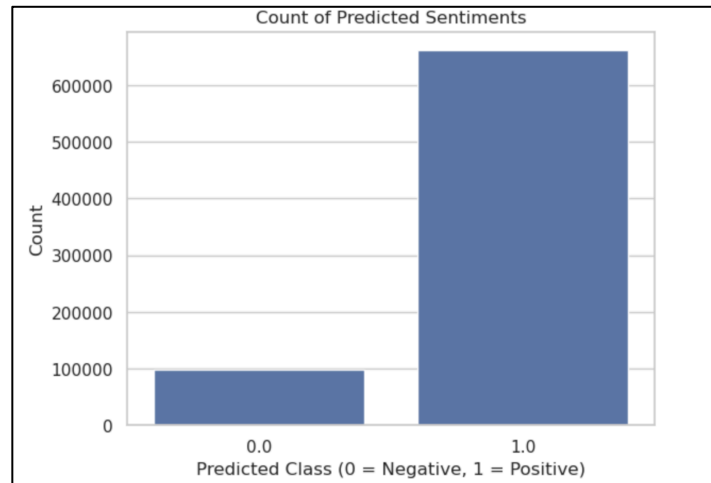
This bar chart shows the distribution of model predictions between negative (0) and positive (1) sentiments. The model predicts a significantly higher number of reviews as positive, suggesting either an actual class imbalance in the dataset or a prediction bias in favor of positive reviews.

Average Rating by Predicted Sentiment

Average Rating by Predicted Sentiment
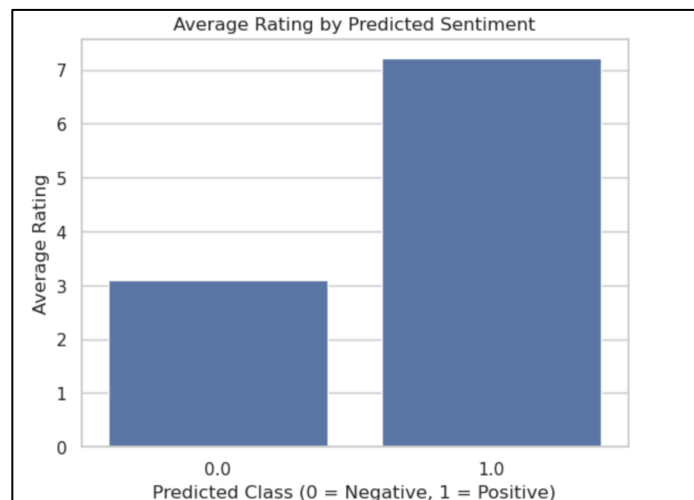
This visualization compares the average actual IMDb ratings for reviews classified as negative or positive. The model's predicted sentiment aligns with true rating trends, as positively predicted reviews tend to have higher average scores.
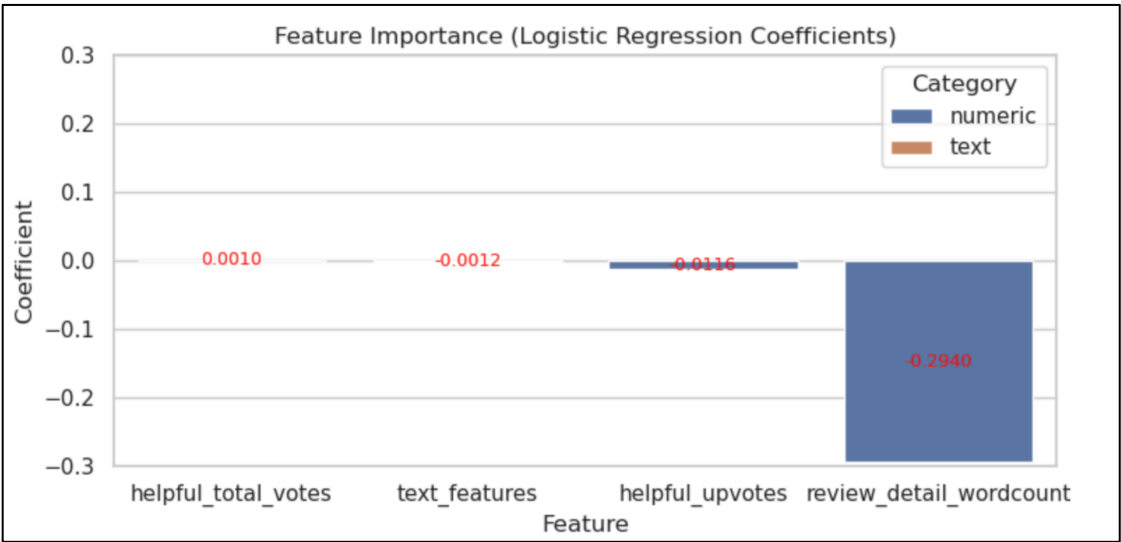
This chart visualizes the coefficients of the logistic regression model, showing which features had the most influence on classification. `review_detail_wordcount` had the strongest negative influence, indicating that longer reviews are associated with negative sentiment. Other features like `helpful_upvotes` and `text_features` had smaller effects, suggesting weaker contributions to the model's decision.

Visualization folder

A new folder was created to save all the visualizations in order to automate the visualization pipeline

Milestone 6: Summary and Conclusions

This milestone brings together all components of the project, showcasing a complete end-to-end data engineering pipeline on the IMDb movie review dataset. We began by acquiring and cleaning over 5 million reviews, transforming raw JSON data into a reliable, structured format. Feature engineering included text vectorization with TF-IDF and the creation of numerical metadata features. A logistic regression model was trained using PySpark and evaluated through cross-validation and test data scoring.

One of the primary challenges was managing the scale of data within memory and compute limits. Cross-validation with grid search initially caused kernel crashes due to resource constraints, which was resolved by optimizing cluster settings and scaling up compute nodes in Dataproc. Another difficulty was ensuring that evaluation metrics were consistent; AUC discrepancies between different evaluation methods led to identifying that raw probabilities, not hard predictions, should be used for ROC curve generation. Also, formatting and aligning visual outputs across Spark, Pandas required careful orchestration to ensure automation and reproducibility.

Model evaluation revealed an AUC of 0.84 on the test set, indicating strong discriminative performance. Multiple visualizations; including ROC curves, confusion matrix, feature importance, review volume over time, and sentiment distribution provided insights into model behavior and data structure. All visualizations were programmatically saved and uploaded to Google Cloud Storage to support automated reporting and reproducibility.

Overall, this milestone demonstrates how cloud-native tools like GCP, Dataproc, and PySpark can be effectively orchestrated to process large-scale, semi-structured data and deliver actionable machine learning insights. Future improvements could include using richer text representations experimenting with ensemble models, or applying stratified sampling to address class imbalance.

The project can be found on GitHub. https://github.com/Yani-k/IMDb_Reviews_DE_ML

<div align="center">**Appendix A**</div>

Code for Data Accusation (Milestone 2)

Downloading the dataset to VM:

```
curl -L -o imdb-review-dataset.zip
https://www.kaggle.com/api/v1/datasets/download/ebiswas/imdb-review-dataset
```

Installing zip unzip program:

```
sudo apt install zip
```

Unzipping the dataset:

```
unzip imdb-review-dataset.zip
```

Setup google cloud authentication:

```
gcloud auth login
```

Create bucket in Cloud Storage:

```
gcloud storage buckets create gs://my-bigdata-project-ky --
project=cis-9760-bdt-project-yunus --default-storage-class=STANDARD --
location=us-central1 --uniform-bucket-level-access
```

Upload the unzipped contents of the dataset to the bucket inside the landing/ folder

```
gcloud storage cp *.json gs://my-bigdata-project-ky/landing/
```

Note: the dataset only has json files therefore we use *.json to upload all the the json files.

Create the remaining folders in the bucket:

```
gcloud storage cp /dev/null gs://my-bigdata-project-ky/trusted/
gcloud storage cp /dev/null gs://my-bigdata-project-ky/code/
gcloud storage cp /dev/null gs://my-bigdata-project-ky/models/
```

Appendix B

Code for EDA

```python
def main_imdb():
    # This function processes all JSON files in the landing
directory of the GCS bucket.
    bucket_name = 'my-bigdata-project-ky'  # Ensure this is
correct for your GCS bucket

    storage_client = storage.Client()
    blobs = storage_client.list_blobs(bucket_name,
prefix="landing/")

    # Get all JSON files
    json_blobs = [blob for blob in blobs if
blob.name.endswith('.json')]

    for blob in json_blobs:
        print(f"Processing file {blob.name} with size
{blob.size} bytes created on {blob.time_created}")

        # Read in the file
        df = pd.read_json(StringIO(blob.download_as_text()))

        # Updated categorical and numeric column lists
        categorical_columns_list =
["spoiler_tag","review_summary_count","review_date"]
        numeric_columns_list = ["rating","helpful_upvotes",
"helpful_total_votes"]

        # Handle 'helpful' column if it exists
        if "helpful" in df.columns:
            df[["helpful_upvotes", "helpful_total_votes"]] =
df["helpful"].apply(
                lambda x: pd.Series(x) if isinstance(x, list)
and len(x) == 2 else pd.Series([None, None])
            )
            df.drop(columns=["helpful"], inplace=True)

        # Count the number of words in 'review_summary' column
        if "review_summary" in df.columns:
            df["review_summary_count"] =
df["review_summary"].apply(lambda x: len(str(x).split()) if
pd.notnull(x) else 0)

        # Perform EDA
```

```python
        eda_results = perform_EDA(df, blob.name)
        numeric_summary_df = perform_EDA_numeric(df, blob.name)
        categorical_summary_df = perform_EDA_categorical(df,
blob.name, categorical_columns_list)

        print(numeric_summary_df.head(24))
        print(categorical_summary_df.head(12))

        # Generate visualizations for categorical columns
        for col in categorical_columns_list:
            plt.figure(figsize=(10, 5))
            sns.countplot(y=df[col],
order=df[col].value_counts().index)
            plt.title(f"Distribution of {col}")
            plt.xticks(rotation=45)
            plt.show()

        # Generate visualizations for numeric columns
        for col in numeric_columns_list:
            plt.figure(figsize=(10, 5))
            sns.histplot(df[col].dropna(), bins=30, kde=True)
            plt.title(f"Distribution of {col}")
            plt.show()
```

Appendix C

Code for Data Cleaning

```
#Author: Yunus Khan
#IMDb Movie Reviews Cleaning
#------------------------------------------------
# Import the storage module
from google.cloud import storage
import pandas as pd
from io import StringIO

# Define your bucket and folder paths
bucket_name = 'my-bigdata-project-ky'
landing_folder = f"{bucket_name}/landing/"
cleaned_folder = f"{bucket_name}/cleaned/"

# List of columns to keep
selected_columns = ["movie", "rating",
"review_summary","review_detail",
                    "review_date",
"spoiler_tag","helpful_upvotes","helpful_total_votes"]

# Connect to Google Cloud Storage
storage_client = storage.Client()
blobs = storage_client.list_blobs(bucket_name,
prefix="landing/")

# Process each JSON file
for blob in blobs:
    if blob.name.endswith('.json'):
        print(f"Processing {blob.name}")

        # Read the JSON file into a DataFrame
        df = pd.read_json(StringIO(blob.download_as_text()))


        # Split helpful column and drop
        df[["helpful_upvotes", "helpful_total_votes"]] =
pd.DataFrame(
                df["helpful"].tolist(),
                index=df.index)

        df.drop(columns=["helpful"], inplace=True)

        df = df[selected_columns]
```

```python
        # Convert columns to appropriate data types

        df['movie'] = df['movie'].astype('string')
        df['spoiler_tag'] = df['spoiler_tag'].astype('Int64')
        df['review_summary'] =
df['review_summary'].astype('string')
        df['review_detail'] =
df['review_detail'].astype('string')

        df["review_date"] = pd.to_datetime(df["review_date"],
errors="coerce")
        df.dropna(subset=["review_date"], inplace=True)  #
optional: remove failed dates
        df["review_date"] = df["review_date"].dt.strftime("%Y-
%m-%d")

        df["helpful_upvotes"] =
pd.to_numeric(df["helpful_upvotes"],
errors='coerce').astype('Int64')
        df["helpful_total_votes"] =
pd.to_numeric(df["helpful_total_votes"],
errors='coerce').astype('Int64')


        # Drop rows with any missing values
        df.dropna(inplace=True)

        # Filtering only movies as the dataset has TV shows in
the movie column
        df= df[df['movie'].str.contains(r"\(\d{4}\)$",
regex=True)]

        # Filter out invalid ratings
        df = df[df['rating'].between(1.0, 10.0)]

        # Remove duplicate
        df.drop_duplicates(inplace=True)

        display(df.head(10))

        #Save the cleaned data as a Parquet file
        cleaned_filename = blob.name.replace("landing/",
"cleaned/").replace(".json", ".parquet")
        df.to_parquet(f"gs://{bucket_name}/{cleaned_filename}",
index=False)

        print(f"Saved cleaned file to {cleaned_filename}")
```

Appendix D

```
# Khan Mohammad Yunus -- Feature Engineering --5/8/2025

# -----------------------------------------
# Import required PySpark ML modules
# -----------------------------------------
from pyspark.ml.feature import RegexTokenizer, StopWordsRemover,
HashingTF, IDF, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import to_date, year, month, length,
size, split, col, when
from pyspark.ml import Pipeline

# -----------------------------------------
# Load cleaned data from cleaned/ folder
# -----------------------------------------
sdf = spark.read.parquet("gs://my-bigdata-project-ky/cleaned/")

# -----------------------------------------
# Feature Engineering
# -----------------------------------------

# Convert string date to proper date type
sdf = sdf.withColumn("review_date", to_date(col("review_date"),
"yyyy-MM-dd"))

# Filter nulls and apply custom logic
sdf = sdf.filter((col("rating").isNotNull()) &
(col("review_detail").isNotNull()) &
(col("review_summary").isNotNull()))
sdf = sdf.withColumn("review_detail_wordcount",
size(split(col("review_detail"), " ")))
sdf = sdf.filter(length(col("review_detail")) > 10)
sdf = sdf.filter(col("review_detail_wordcount") > 5)

# Create year/month/yearmonth columns
sdf = sdf.withColumn("review_year", year(col("review_date")))
sdf = sdf.withColumn("review_month", month(col("review_date")))
sdf = sdf.withColumn("review_yearmonth",
sdf["review_date"].cast("timestamp").cast("string").substr(0,
7))
```

```python
# Cast numerical columns
sdf = sdf.withColumn("helpful_upvotes",
col("helpful_upvotes").cast("double"))
sdf = sdf.withColumn("helpful_total_votes",
col("helpful_total_votes").cast("double"))

# Create binary label column
sdf = sdf.withColumn("label", when((col("rating") >= 5),
1.0).otherwise(0.0))

# -----------------------------------------
# Text Processing for review_summary
# -----------------------------------------
tokenizer = RegexTokenizer(inputCol="review_summary",
outputCol="tokens", pattern="\\W")
remover = StopWordsRemover(inputCol="tokens",
outputCol="filtered")
tf = HashingTF(inputCol="filtered", outputCol="text_tf",
numFeatures=5000)
idf = IDF(inputCol="text_tf", outputCol="text_features")

# -----------------------------------------
# Assemble Final Features
# -----------------------------------------
numerical_features = ["helpful_upvotes", "helpful_total_votes",
"review_detail_wordcount", "spoiler_tag"]
final_assembler = VectorAssembler(inputCols=["text_features"] +
numerical_features, outputCol="features")

# -----------------------------------------
# Logistic Regression Model
# -----------------------------------------
lr = LogisticRegression(featuresCol="features",
labelCol="label")

# -----------------------------------------
# Pipeline Assembly
# -----------------------------------------
pipeline = Pipeline(stages=[
    tokenizer,
    remover,
    tf,
    idf,
    final_assembler,
    lr
])
```

```python
# -------------------------------------------
# Train-Test Split
# -------------------------------------------
train_data, test_data = sdf.randomSplit([0.8, 0.2], seed=42)

# -------------------------------------------
# Cross-Validation and Grid Search
# -------------------------------------------
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Create parameter grid
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

# Define evaluator
evaluator = BinaryClassificationEvaluator(labelCol="label",
metricName="areaUnderROC")

# Set up CrossValidator
crossval = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=3,
    parallelism=4  # Match with your number of cores
)

# Fit the cross-validated model
cvModel = crossval.fit(train_data)

# -------------------------------------------
# Predict & Evaluate
# -------------------------------------------
predictions = cvModel.transform(test_data)
predictions.select("review_summary", "label",
"prediction").show(15, truncate=False)

roc_auc = evaluator.evaluate(predictions)
print(f"Area Under ROC Curve (AUC): {roc_auc:.4f}")

# ------------------------------------------------
# Save Outputs to trusted/ and model/ folders
# ------------------------------------------------
```

```
# Save feature-engineered training data
sdf.write.mode("overwrite").parquet("gs://my-bigdata-project-
ky/trusted/feature_engineered_data.parquet")

# Save predicted data
predictions.write.mode("overwrite").parquet("gs://my-bigdata-
project-ky/trusted/predictions.parquet")

# Save model
cvModel.write().overwrite().save("gs://my-bigdata-project-
ky/models/LogisticRegression_model")
```

```
#----Automation script
gcloud dataproc jobs submit pyspark --cluster cluster-a8fe --
region us-central1 gs://my-bigdata-project-
ky/code/feature_engineering.py
```

Appendix E

```
#Data Visualization and Feature Importance----Yunus----5/9/2025

# This notebook includes visualizations and feature importance
analysis for the IMDb movie review prediction model.

# ------------------
# Imports
# ------------------
from pyspark.sql import SparkSession
from pyspark.ml.tuning import CrossValidatorModel
from pyspark.ml.feature import VectorAssembler
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.cloud import storage
import os

def upload_plot_to_gcs(local_path, gcs_path):
    client = storage.Client()
    bucket = client.bucket('my-bigdata-project-ky')
    blob = bucket.blob(gcs_path)
    blob.upload_from_filename(local_path)
    os.remove(local_path)  # Clean up local copy

# ------------------
# Setup
# ------------------
sns.set(style="whitegrid")
spark = SparkSession.builder.getOrCreate()

# Load Spark predictions and convert to pandas
predictions = spark.read.parquet("gs://my-bigdata-project-
ky/trusted/predictions.parquet")
predictions_pd = predictions.select("rating", "prediction",
"label", "helpful_upvotes", "helpful_total_votes").toPandas()

# Load cleaned data and select relevant columns
cleaned_df = spark.read.parquet("gs://my-bigdata-project-
ky/cleaned/")
cleaned_df = cleaned_df.select("rating", "helpful_upvotes",
"helpful_total_votes").toPandas()

# Load CrossValidatorModel and extract best Logistic Regression
stage
```

```python
cv_model = CrossValidatorModel.load("gs://my-bigdata-project-
ky/models/LogisticRegression_model")
lr_model = cv_model.bestModel.stages[-1]  # Last stage in
pipeline is Logistic Regression

# Define input feature names manually (used in training)
# Extract text feature importance (only one vector feature —
overall weight)
all_coefficients = lr_model.coefficients.toArray()

text_feature_importance = pd.DataFrame({
    "feature": ["text_features"],
    "coefficient": [all_coefficients[0]]
})

# Define numeric feature names and their coefficients
# Extract numeric feature coefficients
numeric_feature_names = ["helpful_upvotes",
"helpful_total_votes", "review_detail_wordcount"]
numeric_coefficients = all_coefficients[-3:]
numeric_feature_importance = pd.DataFrame({
    "feature": numeric_feature_names,
    "coefficient": numeric_coefficients
})

importance_df = pd.concat([text_feature_importance,
numeric_feature_importance])
importance_df["category"] =
importance_df["feature"].apply(lambda x: "text" if x ==
"text_features" else "numeric")
importance_df = importance_df.sort_values(by="coefficient",
ascending=False)




# 1. Prediction Class Count
# This bar chart provides a simple overview of how the model's
predictions are distributed.
# It's useful to quickly check if the model is biased towards a
class or is balanced in sentiment prediction.

plt.figure()
sns.countplot(x="prediction", data=predictions_pd)
plt.title("Count of Predicted Sentiments")
plt.xlabel("Predicted Class (0 = Negative, 1 = Positive)")
plt.ylabel("Count")
```

```python
local_path = "prediction_count.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path,
"visualisations/prediction_count.png")
plt.show()



# 2. Average Rating by Predicted Sentiment
# This bar plot compares average actual ratings for each
predicted class.
# It's a sanity check to ensure predictions align with real-
world user ratings.
plt.figure()
avg_rating_pred =
predictions_pd.groupby("prediction")["rating"].mean().reset_ind
ex()
sns.barplot(x="prediction", y="rating", data=avg_rating_pred)
plt.title("Average Rating by Predicted Sentiment")
plt.xlabel("Predicted Class (0 = Negative, 1 = Positive)")
plt.ylabel("Average Rating")
local_path = "avg_rating_by_prediction.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path,
"visualisations/avg_rating_by_prediction.png")
plt.show()

# 3. Feature Importance Bar Chart
# Visualizes feature weights learned by the model to interpret
variable influence.

importance_df_rounded = importance_df.copy()
importance_df_rounded["coefficient"] =
importance_df_rounded["coefficient"].round(4)
importance_df_rounded =
importance_df_rounded.sort_values(by="coefficient",
ascending=True).reset_index(drop=True)

plt.figure(figsize=(8, 4))
ax = sns.barplot(y="coefficient", x="feature",
data=importance_df_rounded, hue="category", dodge=False)
ax.invert_xaxis()  # Reverse axis to highlight most important
feature

# Add coefficient values inside bars
for i, coef in enumerate(importance_df_rounded["coefficient"]):
    ax.text(i, coef / 2, f"{coef:.4f}", ha='center',
va='center', fontsize=9, color='red')
```

```python
plt.title("Feature Importance (Logistic Regression
Coefficients)")
plt.xlabel("Feature")
plt.ylabel("Coefficient")
plt.legend(title="Category")
plt.ylim(-0.3, 0.3)
plt.tight_layout()
local_path = "feature_importance.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path,
"visualisations/feature_importance.png")
plt.show()



# 4. Top 10 Most Reviewed Movies (with Count Labels)
# A horizontal bar chart highlighting the most reviewed movie
titles.
# This shows which titles attracted the most attention and may
reflect popularity or controversy.

sdf_top_movies = spark.read.parquet("gs://my-bigdata-project-
ky/cleaned/")
sdf_top_movies_pd = sdf_top_movies.select("movie").toPandas()
top_movies = sdf_top_movies_pd["movie"].value_counts().head(10)
plt.figure(figsize=(10, 6))
sns.barplot(x=top_movies.values, y=top_movies.index,
palette="crest")
plt.title("Top 10 Most Reviewed Movies")
plt.xlabel("Number of Reviews")
plt.ylabel("Movie Title")
for i, v in enumerate(top_movies.values):
    plt.text(v + 1, i, str(v), color='black', va='center')
plt.tight_layout()
local_path = "top10_reviewed_movies.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path,
"visualisations/top10_reviewed_movies.png")
plt.show()

# 5. Average Rating by Review Month (Trend)
# This line chart reveals trends in viewer sentiment over time.
# Useful for observing whether ratings improve or decline
during certain periods.
sdf_monthly = spark.read.parquet("gs://my-bigdata-project-
ky/trusted/feature_engineered_data.parquet")
```

```python
sdf_monthly_pd = sdf_monthly.select("review_yearmonth",
"rating").toPandas()
monthly_avg =
sdf_monthly_pd.groupby("review_yearmonth")["rating"].mean().sor
t_index()
plt.figure(figsize=(12, 4))
monthly_avg.plot(marker='o')
plt.title("Average Rating Over Time")
plt.xlabel("Year-Month")
plt.ylabel("Average Rating")
local_path = "rating_trend_over_time.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path,
"visualisations/rating_trend_over_time.png")
plt.show()

# 6. Confusion Matrix
from sklearn.metrics import confusion_matrix
import numpy as np
conf_mat = confusion_matrix(predictions_pd['label'],
predictions_pd['prediction'])
plt.figure(figsize=(4, 3))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues",
xticklabels=["Negative", "Positive"], yticklabels=["Negative",
"Positive"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
local_path = "confusion_matrix.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path,
"visualisations/confusion_matrix.png")
plt.show()

# 7. ROC Curve
from sklearn.metrics import roc_curve, roc_auc_score
preds_pd = predictions.select("label",
"probability").toPandas()
preds_pd["score"] = preds_pd["probability"].apply(lambda x:
float(x[1]))
fpr, tpr, thresholds = roc_curve(preds_pd["label"],
preds_pd["score"])
roc_auc = roc_auc_score(preds_pd["label"], preds_pd["score"])
plt.figure(figsize=(5, 4))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
```

```
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.tight_layout()
local_path = "roc_curve.png"
plt.savefig(local_path)
upload_plot_to_gcs(local_path, "visualisations/roc_curve.png")
plt.show()
```

```
#----Visualization Automation script

gcloud dataproc jobs submit pyspark --cluster cluster-a8fe --
region us-central1 gs://my-bigdata-project-
ky/code/visualisation.py
```