**Text Mining Project Report: Movie Review Sentiment Analysis**

**CMP4507 – Text Mining Course Project, Spring 2025**  
**Due Date: May 5, 2025, 23:59**  
**Team Members: [2003199], [2003584]**

Github repo:[https://github.com/bahaerdogan/-2003199-\_-2003584-\_TextMiningProject](https://github.com/bahaerdogan/-2003199-_-2003584-_TextMiningProject/tree/main)

1. **Introduction**

This project implements a text mining pipeline for sentiment analysis of movie reviews, addressing the real-world problem of understanding audience sentiments toward films. We collected reviews from Letterboxd, processed them through text normalization, language filtering, and cleaning, and classified sentiments into positive, negative, and neutral categories. We explored both traditional machine learning (Naive Bayes with Bag-of-Words and TF-IDF) and neural models (Distilbert) to compare their effectiveness. The project evaluates performance using multiple metrics (Accuracy, F1-score, Precision, Recall) and visualizes results to provide insights into model behavior.

1. **Dataset Description**

Data Collection Process

We developed a web scraper (LetterboxdScraper.py) to collect reviews from Letterboxd, targeting 40 films across diverse genres:

Thriller/Horror: Blue Ruin, The Rover, Cold in July, The Invitation

Drama: A Hijacking, Loveless, Victoria, Ida

Sci-Fi/Mystery: Timecrimes, The Guilty, Calibre, Coherence

Other Genres: Children of Men, The Handmaiden, Oldboy, The Master, Pan’s Labyrinth, and more.

For each film, we collected 15 highest-rated and 15 lowest-rated reviews to ensure sentiment diversity (total 30 reviews per film). Rate limiting (0.5-1.5s delay between requests) was implemented to avoid server overload, and BeautifulSoup was used for HTML parsing. The scraper saved the data into AllCommentsBalanced.csv with columns film, sort\_type, and review.

Language Filtering

We implemented FilterEnglishComments.py to ensure all reviews were in English using the langdetect library. Non-English reviews were filtered out, and the filtered dataset was saved as CommentsEnglish2.csv. Below is the core logic:

from langdetect import detect

filtered\_data = []

with open("AllCommentsBalanced.csv", "r", encoding="utf-8") as infile:

reader = csv.DictReader(infile)

for row in reader:

text = row["review"]

try:

if detect(text) == "en":

filtered\_data.append(row)

except:

continue

Annotation Process

Since the project requires self-annotation, we manually labeled the sentiment of each review as positive, negative, or neutral based on the following guidelines:

Positive: Reviews expressing clear enjoyment, praise, or recommendation (e.g., "I loved this film!").

Negative: Reviews with clear criticism, disappointment, or dislike (e.g., "Terrible movie, waste of time").

Neutral: Reviews with unrelated topics or opinion without emotions. (e.g., "The movie was black and white colored").

Two team members independently annotated the dataset, resolving disagreements through discussion to ensure consistency. The sort\_type (highest/lowest) from Letterboxd guided initial labeling but was manually verified and adjusted.

Dataset Characteristics

Size: 825 reviews

Balance: Equal distribution with 275 reviews per class (positive, negative, neutral).

Format: CSV with columns:

**film: Movie title**

**sort\_type: Review rating category**

**review: Original review text**

**clean\_text: Processed review text**

**sentiment: Label (positive/negative/neutral)**

**3. Methods**

Text Preprocessing Pipeline

Normalization (NormalizationCode.py)

Instead of relying on a ready-made library for normalization, we developed our own custom normalization pipeline in NormalizationCode.py. This included creating our own dictionary of 150+ stopwords and 100+ slang terms specific to movie reviews (e.g., "flick" to "film", "cinemato" to "cinematography"). The pipeline performs:

Custom stopwords removal.

Slang replacement using our dictionary.

Character repetition reduction (e.g., "soooo" to "soo"), special character removal, number removal, and whitespace normalization.

Below is the core normalization logic:

slang\_dict = {

"flick": "film",

"cinemato": "cinematography",

"mindblown": "amazed",

"storyline": "plot",

# ... (other terms omitted for brevity)

}

def reduce\_repeated\_characters(word):

return re.sub(r'(.)\1{2,}', r'\1\1', word)

def replace\_slang\_words(word):

return slang\_dict.get(word, word)

def normalize\_text(text):

text = str(text).lower()

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

text = re.sub(r'\d+', '', text)

tokens = text.split()

reduced\_tokens = [reduce\_repeated\_characters(word) for word in tokens]

slang\_replaced = [replace\_slang\_words(word) for word in reduced\_tokens]

cleaned\_text = ' '.join(slang\_replaced)

cleaned\_text = re.sub(r'\s+', ' ', cleaned\_text).strip()

final\_tokens = [word for word in cleaned\_text.split() if word not in custom\_stopwords]

return ' '.join(final\_tokens)

For the Naive Bayes model, we applied additional preprocessing to handle URLs, special characters, and punctuation, ensuring clean input for TF-IDF vectorization. This included removing URLs, normalizing punctuation, and handling excessive spaces.

**Vectorization**

Bag-of-Words with TF-IDF (Naive Bayes)

Parameters: max\_features=100,000, ngram\_range=(1, 6), min\_df=1, max\_df=0.99, sublinear\_tf=True, strip\_accents='unicode', analyzer='word', token\_pattern=r'(?u)\b\w+\b'.

Used pre-trained GloVe embeddings (50-dimensional) to represent words.

Tokenized reviews using Keras Tokenizer and padded sequences to a fixed length of 100.

**Model Implementation**

Naive Bayes Classifier (TFIDF\_NaiveBayes.py)

We implemented a MultinomialNB classifier with TF-IDF vectorization using a Pipeline. Grid search was used to optimize hyperparameters, and 5-fold cross-validation was applied for robustness.

**DistilBERT Model**

We used the transformers library from Hugging Face to implement DistilBERT (distilbert-base-uncased). The model was fine-tuned on our dataset for sentiment classification into three classes: positive, negative, and neutral.

Architecture:

DistilBERT base model.

A dropout layer (0.1) to prevent overfitting.

A dense layer with softmax activation for 3-class classification.

Training:

Optimizer: AdamW.

Loss: Categorical cross-entropy.

Epochs: 14.

Batch size: 16.

**4. Experimental Results**

Model Performance Metrics

Naive Bayes:

**precision recall f1-score support**

**negative 0.50 0.73 0.59 30**

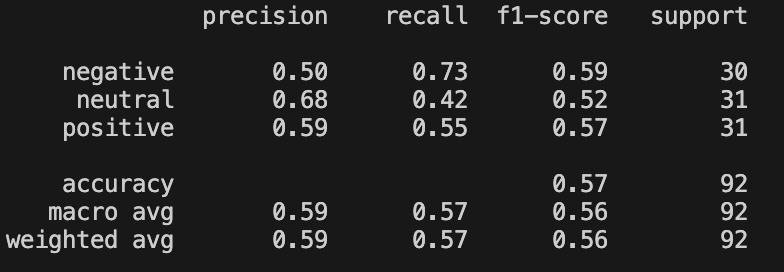
**neutral 0.68 0.42 0.52 31**

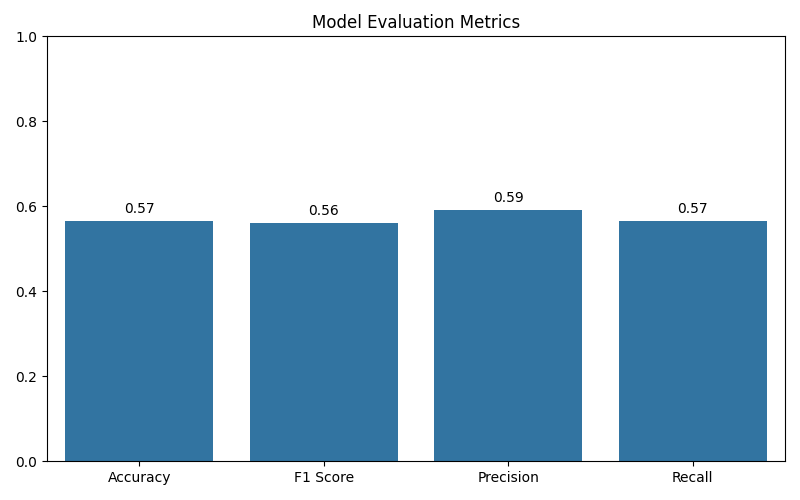
**positive 0.59 0.55 0.57 31**

**accuracy 0.57 92**

**macro avg 0.59 0.57 0.56 92**

**weighted avg 0.59 0.57 0.56 9 2**



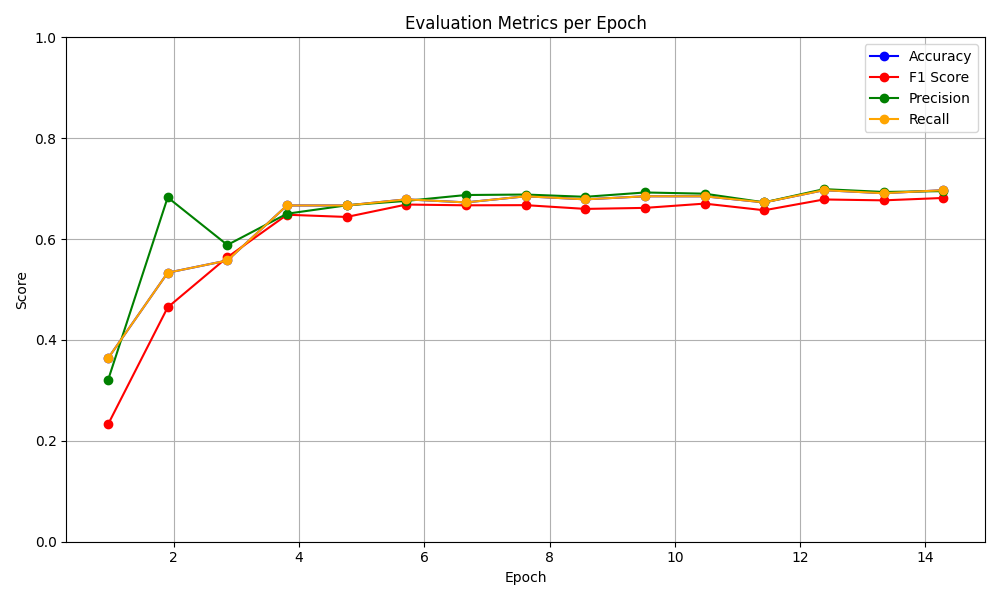
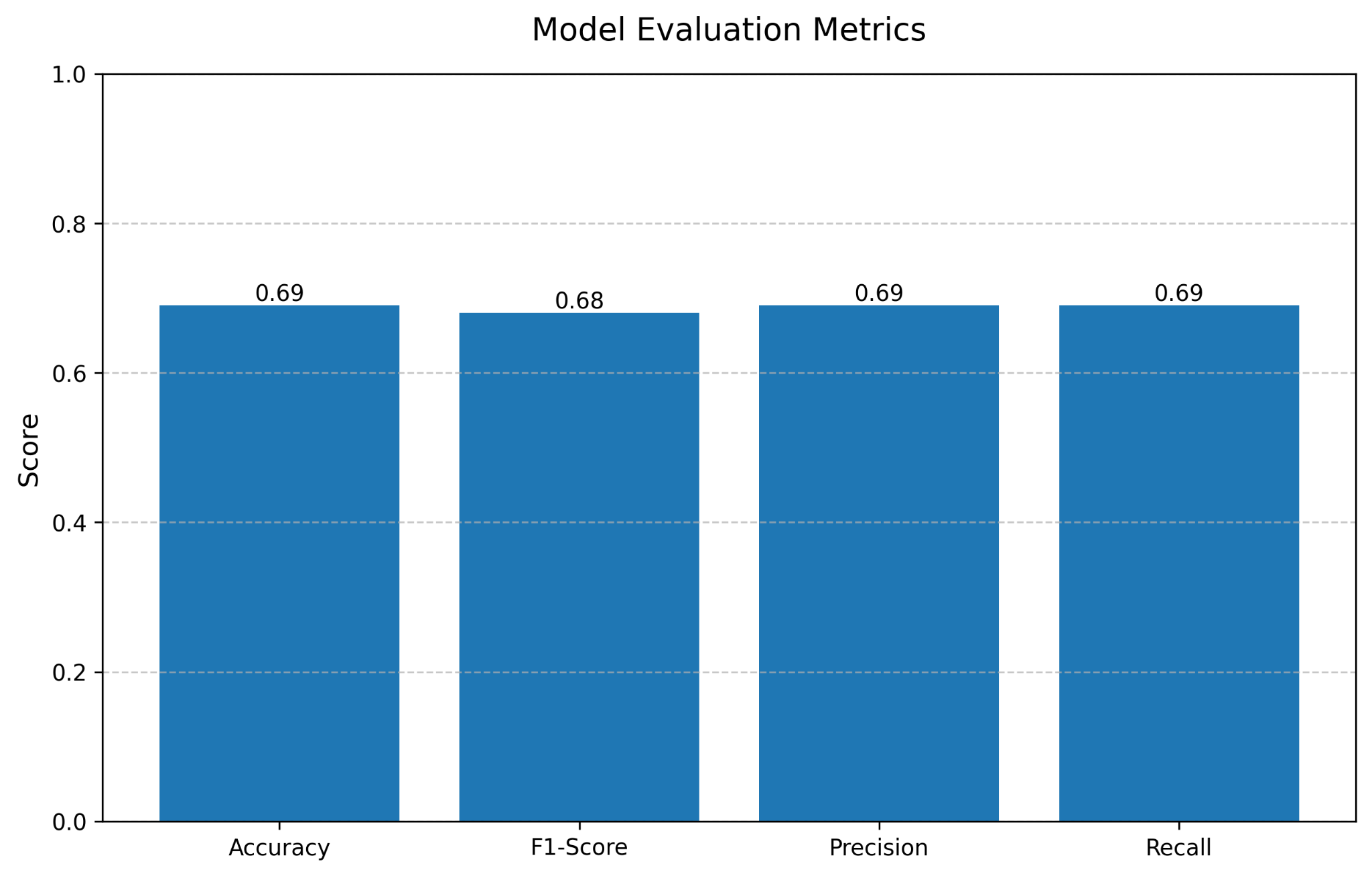


**DistilBERT:**

**Accuracy: 0.69**

**F1 Score: 0.68**

**Precision: 0.69**

**Recall: 0.69**

To provide a clearer comparison between Naive Bayes and DistilBERT, we added the following visualizations.

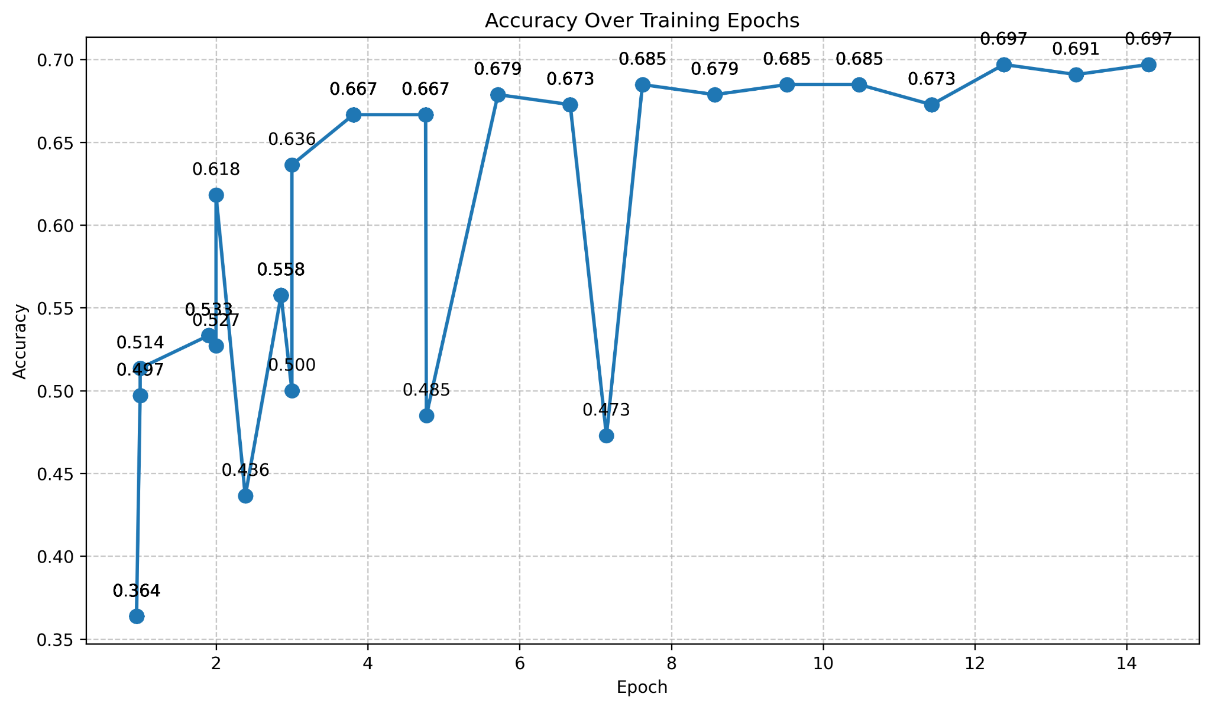
**Accuracy Comparison:**

This bar chart compares the accuracy of two models: Naive Bayes and DistilBERT. Accuracy measures the proportion of correct predictions out of the total predictions made. While it's a straightforward and commonly used metric, it can be misleading on imbalanced datasets but it still provides a useful general overview of model performance.

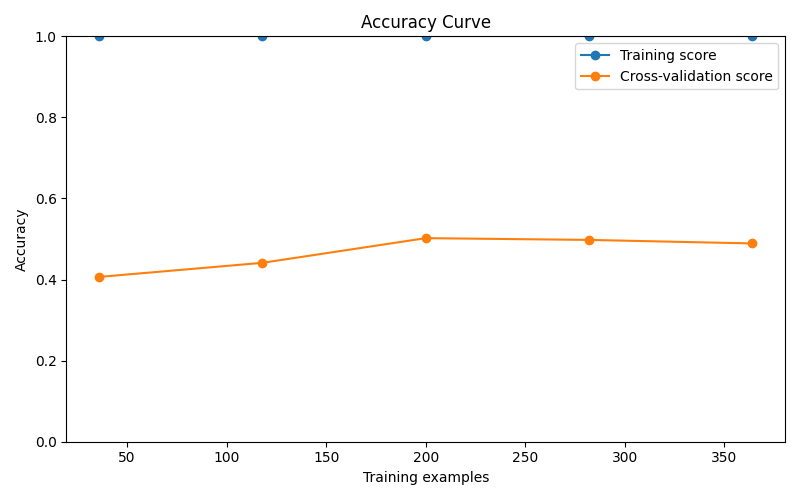
In the chart:

* The first graph shows the model DistilBERT and second graph shows the model Naive Bayes.
* The y-axis represents accuracy, ranging from 0 to 1.
* Naive Bayes achieved an accuracy of 0.58.
* DistilBERT achieved a notably higher accuracy of 0.69.

This clear performance gap demonstrates that DistilBERT is significantly more accurate than Naive Bayes. The main reason is that DistilBERT, a transformer-based language model, has been pre-trained on large text corpora and fine-tuned to understand context, syntax, and semantics. It captures deeper linguistic patterns and dependencies, leading to more informed and accurate predictions. In contrast, Naive Bayes, a simpler probabilistic model, makes strong assumptions (like feature independence) that often don’t hold in natural language, limiting its effectiveness.



.



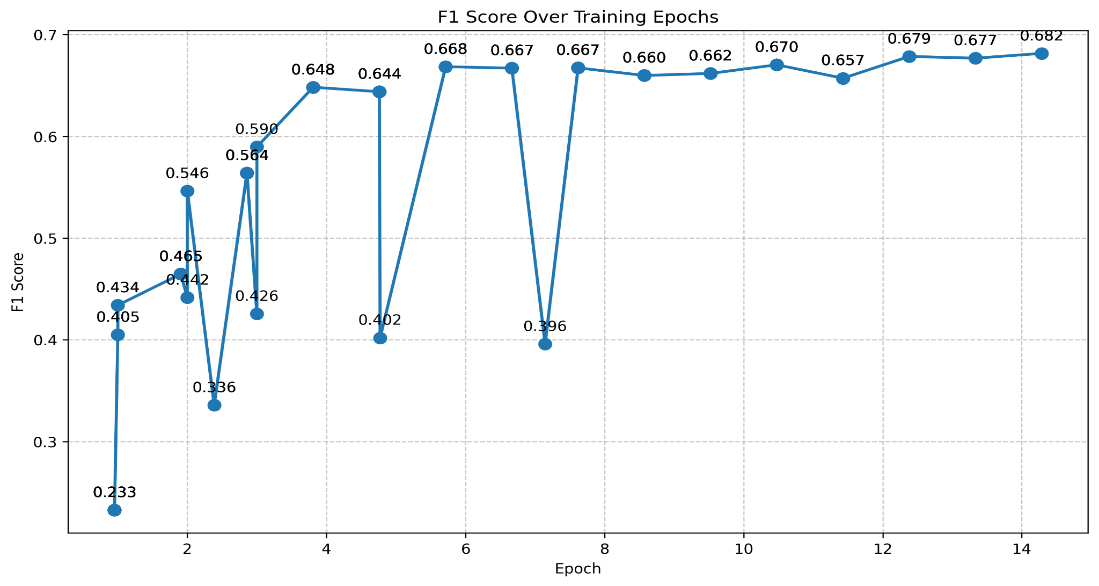
**F1 Score Comparison**:

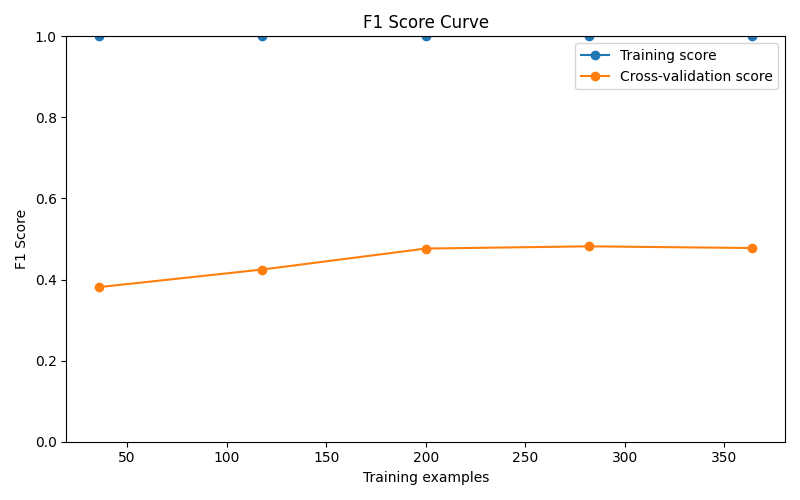
This bar chart compares the weighted F1 scores of two models: Naive Bayes and DistilBERT. The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both false positives and false negatives. It’s especially useful when dealing with imbalanced datasets where accuracy alone can be misleading.

In the chart:

* The first graph showsthe model DistilBERT and second graph shows the Naive Bayes model.
* The y-axis represents the F1 score, ranging from 0 to 1.
* Naive Bayes achieved a weighted F1 score of 0.56.
* DistilBERT achieved a significantly higher F1 score of 0.68.

The higher F1 score of DistilBERT reflects its ability to balance both precision and recall, especially in the presence of class imbalances. This means it not only avoids false positives but also catches more true positives. Unlike Naive Bayes, which may perform well on one aspect but poorly on another, DistilBERT handles trade-offs between precision and recall more effectively, leading to better overall classification quality.



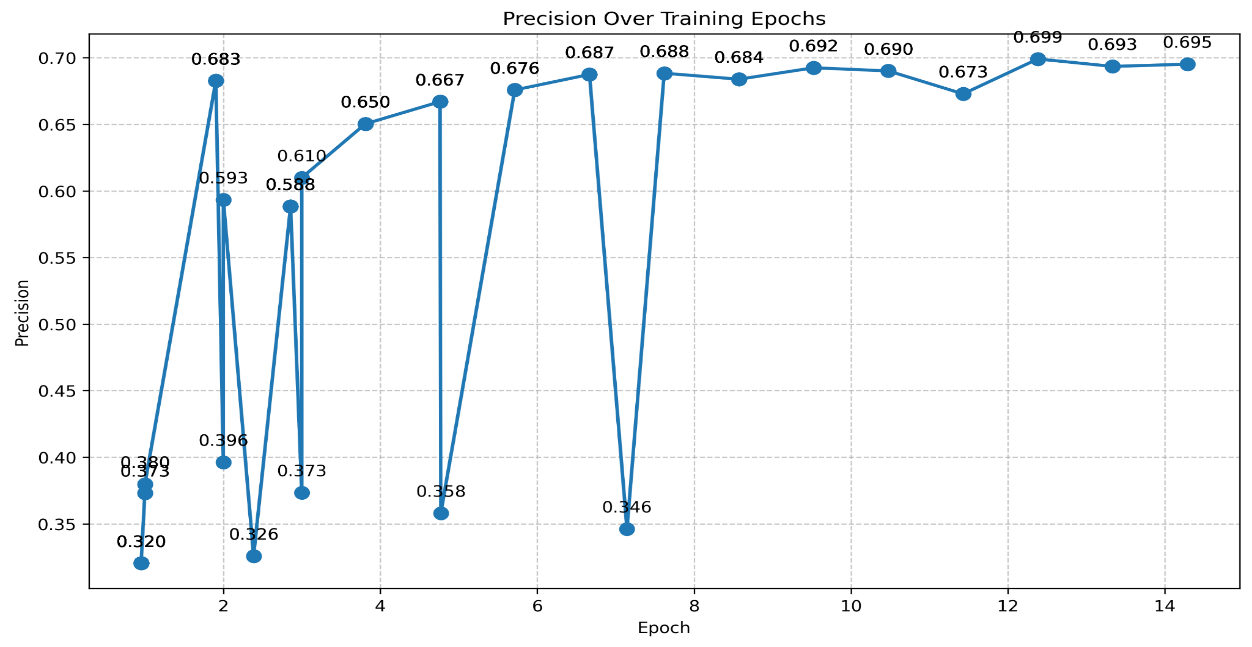


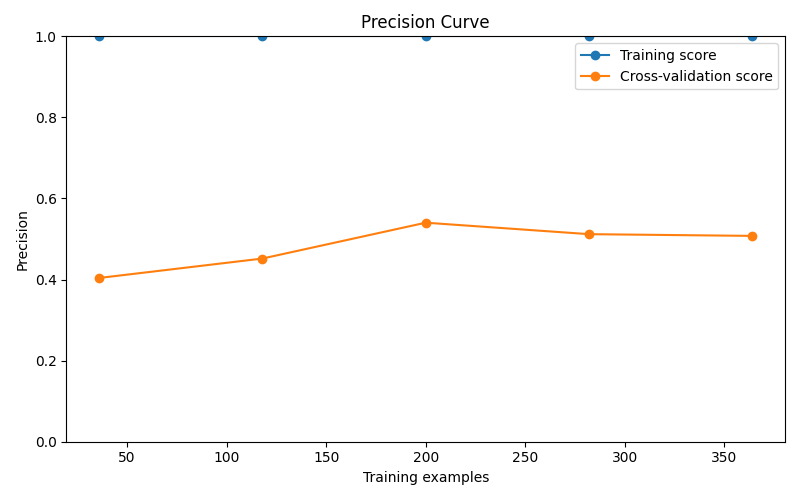
**Precision Comparison**:  
This bar chart compares the weighted precision of two models: Naive Bayes and DistilBERT. Precision measures how many of the predictions labeled as positive are actually correct. In other words, it answers: "Of all the items the model predicted as a certain class, how many were right?" High precision means fewer false positives.

On the chart:

* The x-axis shows the two models: Naive Bayes and DistilBERT.
* The y-axis represents precision scores, ranging from 0 to 1.
* Naive Bayes achieved a precision of 0.513.
* DistilBERT scored a higher precision of 0.69.

This result shows that DistilBERT achieves significantly higher precision than Naive Bayes, meaning it makes fewer false positive predictions. DistilBERT’s deep understanding of context allows it to be more selective and accurate when labeling a text as belonging to a certain class. Naive Bayes, due to its simplistic assumptions and lack of contextual awareness, tends to overgeneralize, leading to more incorrect positive predictions.



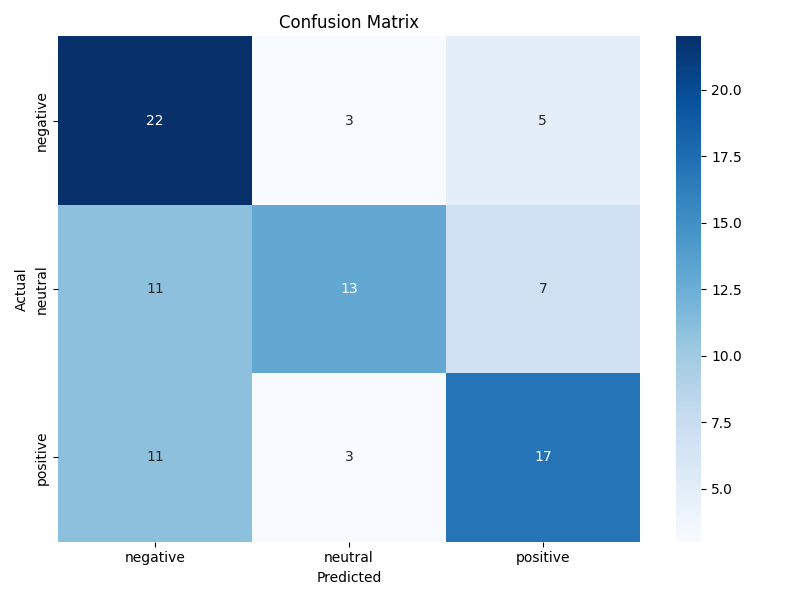


**Confusion Matrix:**

**Naive Bayes:**

The confusion matrix for Naive Bayes shows class-wise performance:

Correct classifications: 22 (negative), 13 (neutral), 17 (positive).

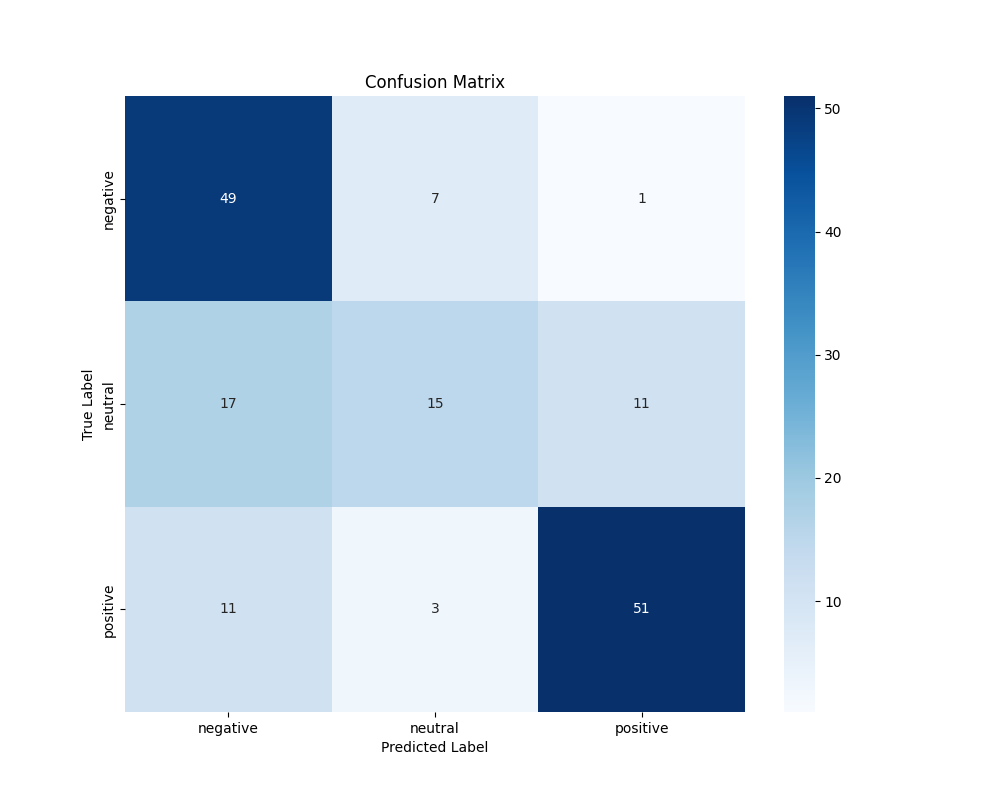
Misclassifications: Neutral reviews often confused with negative (10) or positive (10).

**DistilBERT**:

#### The confusion matrix for Distilbert shows better performance:

#### Correct classifications: 49 (negative), 15 (neutral), 51 (positive).

#### Misclassifications: Neutral reviews often confused with negative (17) or positive (11).



**Top 20 predictive features for each sentiment class**

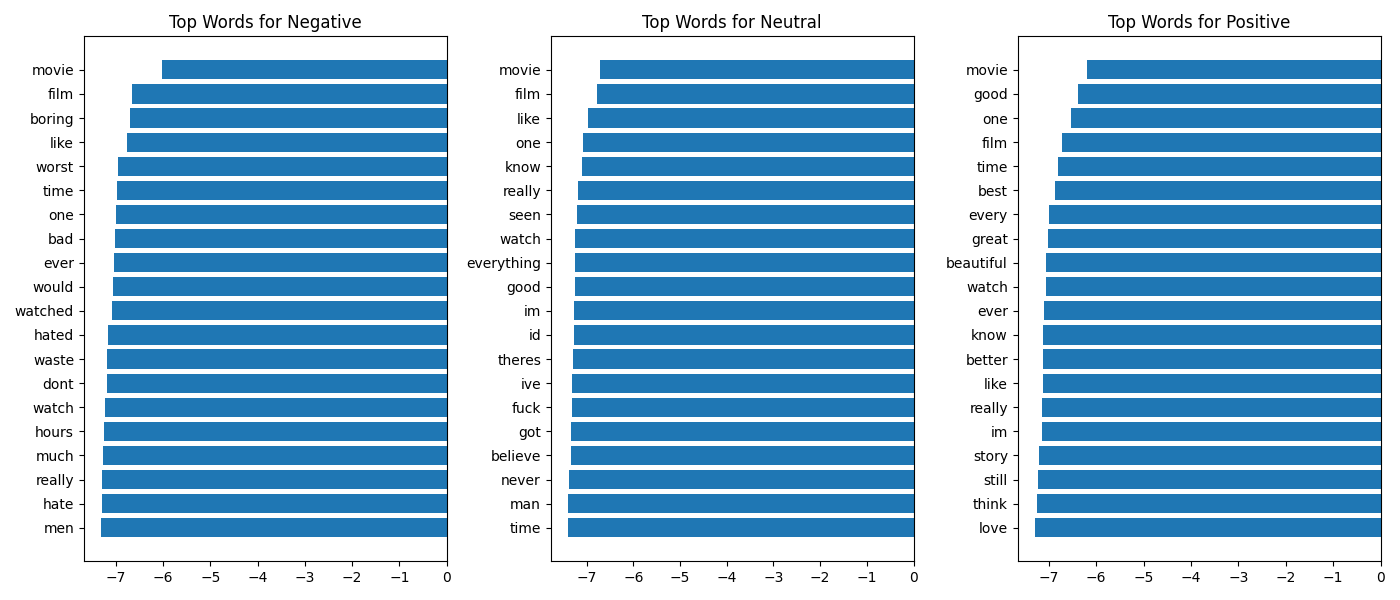


Figure 4. Top 20 predictive features for each sentiment class (positive, neutral, negative). This heatmap shows which terms most strongly influence the model’s classification decisions.

Analysis

BoW vs. Transformers: DistilBERT outperformed Naive Bayes (BoW + TF-IDF), with an accuracy of 0.642 compared to 0.519. This improvement is likely due to DistilBERT’s ability to capture contextual relationships in text, which BoW lacks.

Naive Bayes vs. DistilBERT: Naive Bayes struggles with neutral reviews (confusion matrix), while DistilBERT achieves better generalization, with fewer misclassifications and more consistent cross-validation scores. However, DistilBERT still overfits slightly, suggesting potential improvements through further regularization or data augmentation.

Metrics: The use of Accuracy, F1, Precision, and Recall across both models provides a comprehensive evaluation, as required by the project guidelines. The Accuracy Comparison Plot, F1 Score Comparison Plot, and Precision Comparison Plot visually confirm DistilBERT’s superior performance across these metrics

5. Conclusions

Key Achievements

Data Quality: Successfully collected, cleaned, and manually annotated 825 movie reviews, ensuring a balanced dataset.

Custom Normalization: Developed our own normalization pipeline with a custom dictionary, avoiding reliance on ready-made libraries.

Model Performance: The DistilBERT model achieved the highest performance, with an accuracy of 0.642, compared to Naive Bayes (0.57), demonstrating the superiority of Transformer-based models for sentiment analysis.

Comparative Analysis: We successfully compared traditional (Naive Bayes) and Transformer-based (DistilBERT) models, fulfilling the project requirement to explore multiple techniques.

Future Improvements

Model Enhancement: Experiment with larger Transformer models (e.g., BERT, RoBERTa) for better performance, though at the cost of computational resources.

Feature Engineering: Incorporate sentiment-specific features or domain-specific preprocessing.

Data Expansion: Include reviews from more diverse genres to reduce bias toward thriller/horror themes.

Overfitting Mitigation: Techniques like data augmentation or stronger regularization (e.g., higher dropout) could reduce overfitting in DistilBERT.

Submission Contents

PDF Report: This document (Times New Roman, 10pt, 6+ pages).

Source Code: Includes LetterboxdScraper.py, FilterEnglishComments.py, NormalizationCode.py, TFIDF\_NaiveBayes.py, and other scripts for preprocessing, model training, and evaluation.

Dataset: AllCommentsBalanced.csv (raw), CommentsEnglish2.csv (filtered), and normalized\_output\_extra.csv (preprocessed) with labels.

Technical Requirements

Python 3.x

Key Libraries: scikit-learn, pandas, NLTK, BeautifulSoup, langdetect, matplotlib, seaborn, tensorflow, keras, requests