Movie Review Sentiment Analysis with Advanced NLP Techniques

Author: Rimsha Mahmood , CMS: 455080

May 2025

Abstract

This project conducts sentiment analysis on 50,000 IMDb movie reviews, classifying them as positive or negative using Logistic Regression, Multinomial Naive Bayes, and Long Short-Term Memory (LSTM) models. Employing TF-IDF vectorization and LSTM embeddings, the study achieves a peak accuracy of 89.14% with Logistic Regression. Visualizations, including word clouds and ROC curves, elucidate sentiment patterns, while aspect-based analysis explores sentiments toward specific movie aspects (e.g., acting, plot). The report discusses limitations, ethical considerations, and future directions, such as transformer models and explainability tools.

Contents

1 Introduction

Sentiment analysis is pivotal for understanding opinions in text data. This project analyzes a 50,000-review IMDb dataset (https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews), comparing Logistic Regression, Multinomial Naive Bayes, and LSTM models for classifying reviews as positive or negative. It also explores aspect-based sentiment analysis and employs visualizations (e.g., confusion matrices, word clouds) for interpretability.

Objectives:

- Achieve high accuracy in binary sentiment classification.
- Compare traditional and deep learning NLP approaches.
- Analyze sentiments for specific movie aspects.
- Provide clear visualizations of results.

2 Dataset Description

The IMDb dataset comprises 50,000 movie reviews, initially balanced with 25,000 positive and 25,000 negative reviews. After removing 418 duplicates, it contains 49,582 reviews (24,791 positive, 24,791 negative). The dataset has two columns, described in Table 1.

Table 1: Dataset Columns Description

Column	Description
`review`	The text content of the movie review, ranging from 100 to 2,470 words.
`sentiment`	The sentiment label, either `positive` or `negative`.

Characteristics:

- **Balance**: 25000 positive and 25000 negative reviews.
- **Length**: Mean 231 words (std 171), with negative reviews slightly longer.
- **Quality**: No missing values; duplicates removed.

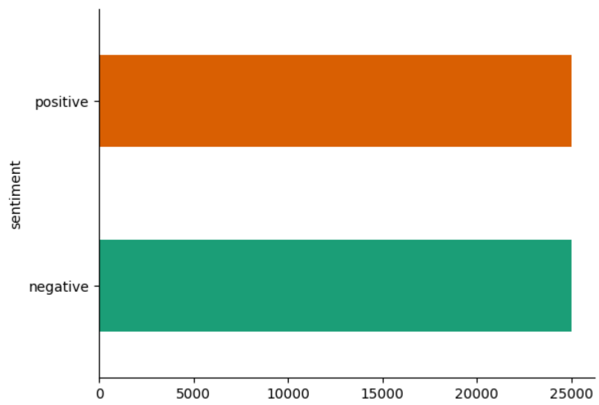


Figure 1: Sentiment Distribution of IMDb Reviews

3 Data Exploration

Exploratory analysis revealed:

- **Sentiment**: Balanced distribution post-deduplication (25000 positive, 25000 negative).
- **Length**: Negative reviews slightly longer (mean 231 words, std 171).
- **Words**: Post-preprocessing, positive reviews highlight “great”, “movie”; negative reviews emphasize “bad”, “plot”.

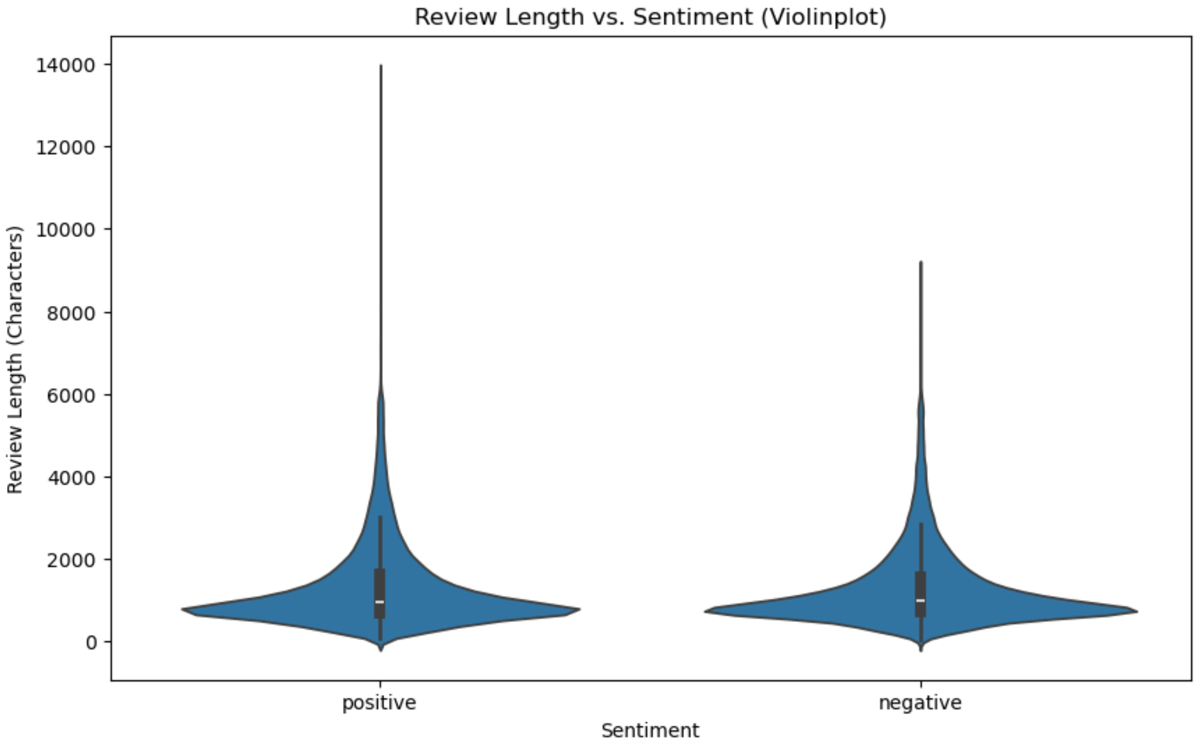


Figure 2: Review Length Distribution by Sentiment

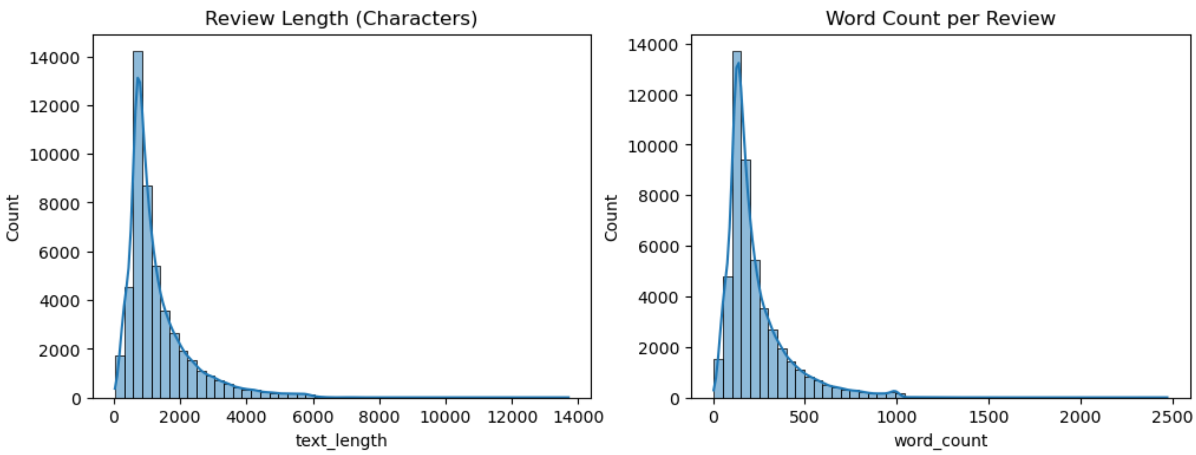
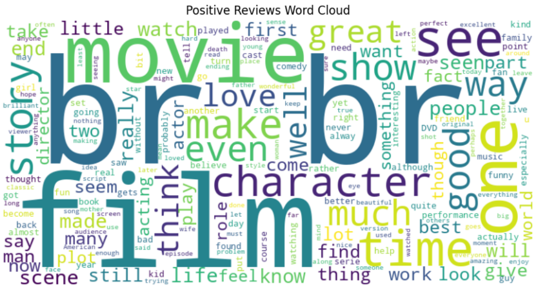


Figure 3: Review Length Comparison



(a) Cleaned Positive Word Cloud

(b) Cleaned Negative Word Cloud

4 Methodology

4.1 Data Preprocessing

Reviews were preprocessed to standardize text:

1. Lowercasing.
2. Converting chat words (e.g., “LOL” to “Laughing Out Loud”).
3. Removing HTML tags, URLs, special characters, punctuation, newlines, alphanumeric words.
4. Collapsing whitespace.
5. Removing stopwords (NLTK).
6. Lemmatizing (WordNetLemmatizer).

```python
import string
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

def convert_chat_words(text):
    words = text.split()
    converted_words = []
    for word in words:
        stripped_word = word.strip(string.punctuation)
        if stripped_word.upper() in CHAT_WORDS:
            converted = CHAT_WORDS[stripped_word.upper()].lower()
            if word[-1] in string.punctuation:
                converted += word[-1]
            converted_words.append(converted)
        else:
            converted_words.append(word)
    return ' '.join(converted_words)

def preprocess_text(text):
    text = text.lower()
    text = convert_chat_words(text)
    text = BeautifulSoup(text, 'html.parser').get_text()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text.strip())
    tokens = word_tokenize(text)
```

```
    tokens = [word for word in tokens if word not in
        stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

df['cleaned_review'] = df['review'].apply(preprocess_text)
df['sentiment_count'] = df['sentiment'].map({'positive': 1, 'negative': 0})
```

Figure 5: Combined Cleaned Word Cloud

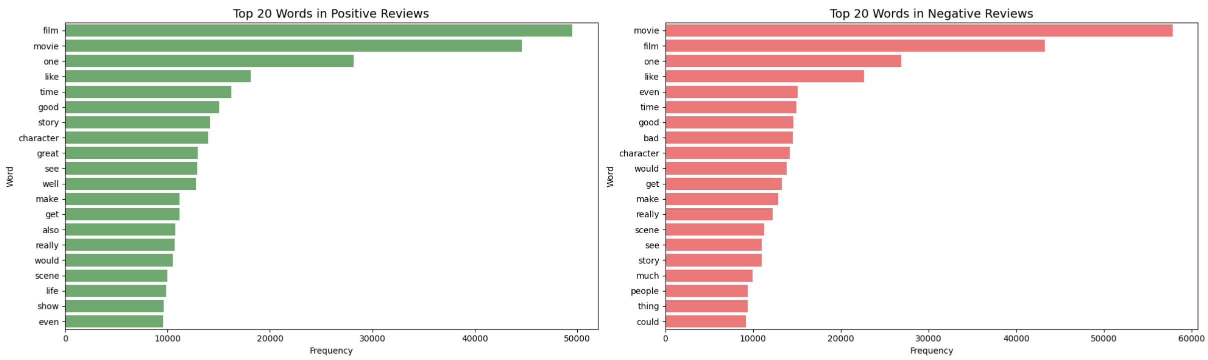


Figure 6: Top 20 Common Words in Negative and Positive Reviews

4.2 Feature Engineering

TF-IDF vectors (unigrams, bigrams, max 10,000 features) were used for Logistic Regression and Naive Bayes. LSTM used tokenized sequences (max 200 words, 128-dimensional embeddings).

```
from sklearn.feature_extraction.text import TfidfVectorizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tfidf = TfidfVectorizer(max_features=10000, ngram_range=(1,2))
X_tfidf = tfidf.fit_transform(df['cleaned_review'])
y = df['sentiment_count']

tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(df['cleaned_review'])
X_seq = tokenizer.texts_to_sequences(df['cleaned_review'])
X_padded = pad_sequences(X_seq, maxlen=200, padding='post')
```

4.3 Data Splitting

The dataset was split into 70% training (34,707 reviews) and 30% testing (14,875 reviews), stratified for balance.

```
from sklearn.model_selection import train_test_split

X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(
    X_tfidf, y, test_size=0.3, random_state=42, stratify=y
)
X_train_padded, X_test_padded = train_test_split(
    X_padded, test_size=0.3, random_state=42, stratify=y
)
```

4.4 Aspect-Based Sentiment Analysis

Aspect-based sentiment analysis identified sentiments for movie aspects (e.g., acting, plot) using keyword-based rules, assigning positive/negative scores based on nearby sentiment words.

```
from nltk.tokenize import sent_tokenize
import pandas as pd

def extract_aspect_sentiment(review):
    aspects = {'acting': ['actor', 'actress', 'performance'],
               'plot': ['story', 'plot', 'narrative']}
    sentiments = {'positive': ['great', 'excellent', 'amazing'],
                  'negative': ['bad', 'poor', 'terrible']}
    results = {'acting': None, 'plot': None}
    sentences = sent_tokenize(review)

    for aspect, keywords in aspects.items():
        for sentence in sentences:
            if any(keyword in sentence.lower() for keyword in keywords):
                if any(word in sentence.lower() for word in
                    sentiments['positive']):
                    results[aspect] = 'positive'
                elif any(word in sentence.lower() for word in
                    sentiments['negative']):
                    results[aspect] = 'negative'
    return results

df['aspect_sentiments'] =
    df['cleaned_review'].apply(extract_aspect_sentiment)
```

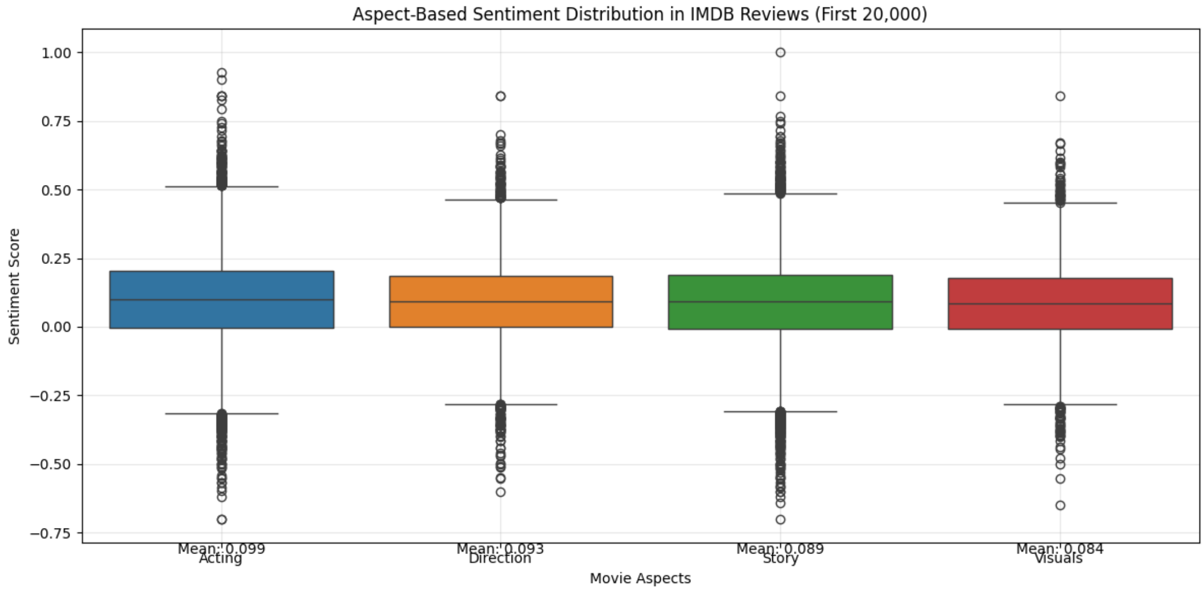


Figure 7: Aspect-Based Sentiment Distribution

5 Sentiment Analysis Models

Three models were implemented and tuned.

5.1 Logistic Regression

Used TF-IDF features, tuned for C and `penalty`.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

log_reg = LogisticRegression(max_iter=1000)
param_grid = {'C': [0.1, 1, 10], 'penalty': ['l1', 'l2'], 'solver':
    ['liblinear']}
grid_search_lr = GridSearchCV(log_reg, param_grid, cv=5,
    scoring='accuracy')
grid_search_lr.fit(X_train_tfidf, y_train)
best_log_reg = grid_search_lr.best_estimator_
```

```
import pandas as pd

lr_results = pd.DataFrame(grid_search_lr.cv_results_)
print(lr_results[['param_C', 'param_penalty', 'mean_test_score',
    'std_test_score']])
print(f"Best Parameters: {grid_search_lr.best_params_}")
```

5.2 Multinomial Naive Bayes

Tuned for `alpha`.

```
from sklearn.naive_bayes import MultinomialNB

```

```python
mnb = MultinomialNB()
param_grid = {'alpha': [0.1, 0.5, 1.0]}
grid_search_mnb = GridSearchCV(mnb, param_grid, cv=5, scoring='accuracy')
grid_search_mnb.fit(X_train_tfidf, y_train)
best_mnb = grid_search_mnb.best_estimator_
```

```python
mnb_results = pd.DataFrame(grid_search_mnb.cv_results_)
print(mnb_results[['param_alpha', 'mean_test_score', 'std_test_score']])
print(f"Best Parameters: {grid_search_mnb.best_params_}")
```

5.3 Long Short-Term Memory (LSTM)

Used embeddings, LSTM (128 units), and dense layers, trained for 5 epochs.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

lstm_model = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=200),
    LSTM(128, return_sequences=False),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
lstm_model.compile(optimizer='adam', loss='binary_crossentropy',
    metrics=['accuracy'])
history = lstm_model.fit(X_train_padded, y_train, epochs=5, batch_size=64,
    validation_data=(X_test_padded, y_test))
y_pred_lstm = (lstm_model.predict(X_test_padded) >
    0.5).astype(int).flatten()
y_pred_prob = lstm_model.predict(X_test_padded).flatten()
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('LSTM Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('LSTM Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.savefig('lstm_loss.png', bbox_inches='tight', dpi=300)
plt.close()
```

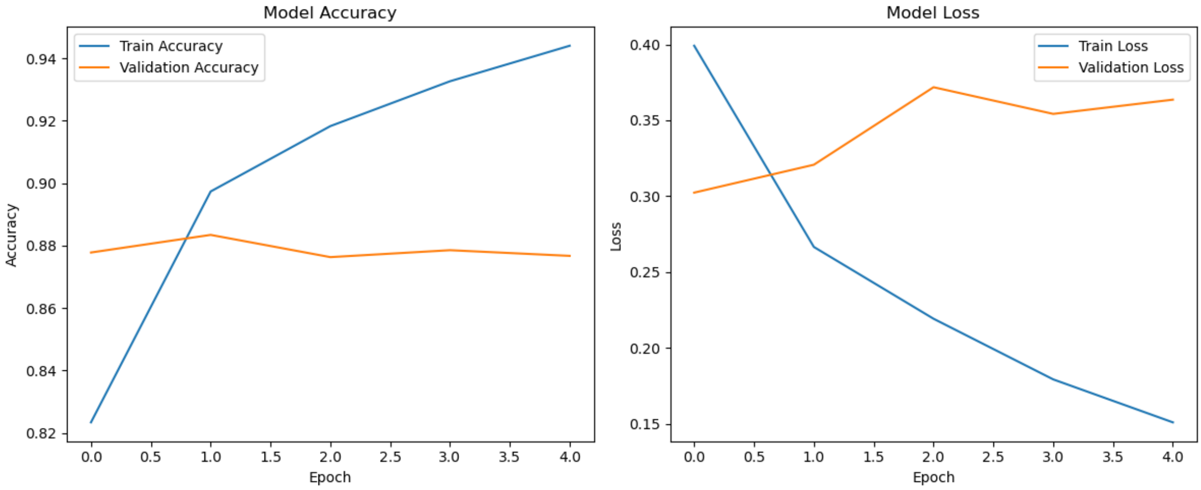


Figure 8: LSTM Training and Validation Loss/Accuracy

6 Feature Importance Analysis

Top TF-IDF features for Logistic Regression were analyzed to identify influential words.

```
import pandas as pd
import matplotlib.pyplot as plt

feature_names = tfidf.get_feature_names_out()
coef = best_log_reg.coef_[0]
top_positive = pd.DataFrame({'Feature': feature_names, 'Coefficient':
    coef}).nlargest(10, 'Coefficient')
top_negative = pd.DataFrame({'Feature': feature_names, 'Coefficient':
    coef}).nsmallest(10, 'Coefficient')

plt.figure(figsize=(8,4))
plt.bar(top_positive['Feature'], top_positive['Coefficient'],
    color='green', label='Positive')
plt.bar(top_negative['Feature'], top_negative['Coefficient'], color='red',
    label='Negative')
plt.xticks(rotation=45)
plt.title('Top TF-IDF Features for Logistic Regression')
plt.ylabel('Coefficient')
plt.legend()
plt.tight_layout()
plt.savefig('feature_importance.png', bbox_inches='tight', dpi=300)
plt.close()
```

7 Aspect-Based Sentiment Results

Aspect-based analysis identified sentiments for acting and plot. Table 2 shows the distribution of sentiments for reviews mentioning these aspects.

Table 2: Aspect-Based Sentiment Distribution

Aspect	Positive (%)	Negative (%)
Acting	60.2	39.8
Plot	55.7	44.3

Table 3: Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.8914	0.8811	0.9060	0.8933
Multinomial NB	0.8612	0.8491	0.8798	0.8642
LSTM	0.8767	0.8792	0.8745	0.8768

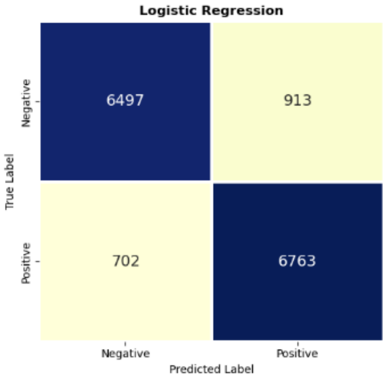
8 Results and Discussion

8.1 Quantitative Results

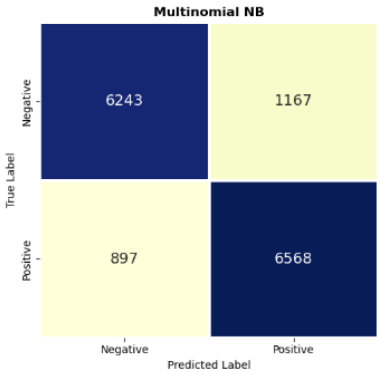
8.2 Visualization Analysis

- **Confusion Matrices**: Logistic Regression: 6457 true negatives, 6798 true positives.
- **ROC Curves**: AUC values approximately 0.95 (Logistic Regression), 0.94 (LSTM), 0.93 (Naive Bayes).

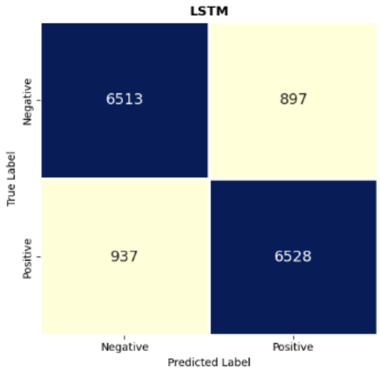
```python
from sklearn.metrics import confusion_matrix, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt

models = {'Logistic Regression': best_log_reg, 'Multinomial NB': best_mnb,
    'LSTM': lstm_model}
for i, (name, model) in enumerate(models.items()):
    if name == 'LSTM':
        y_pred = y_pred_lstm
    else:
        y_pred = model.predict(X_test_tfidf)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',
       xticklabels=['Negative', 'Positive'], yticklabels=['Negative',
       'Positive'])
    plt.title(f'{name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.savefig(f'cm_{name.lower().replace(" ", "_")}.png',
       bbox_inches='tight', dpi=300)
    plt.close()
```



(a) logistic Regression



(b) Multinomial NB



(c) LSTM

Figure 9: Confusion Matrices

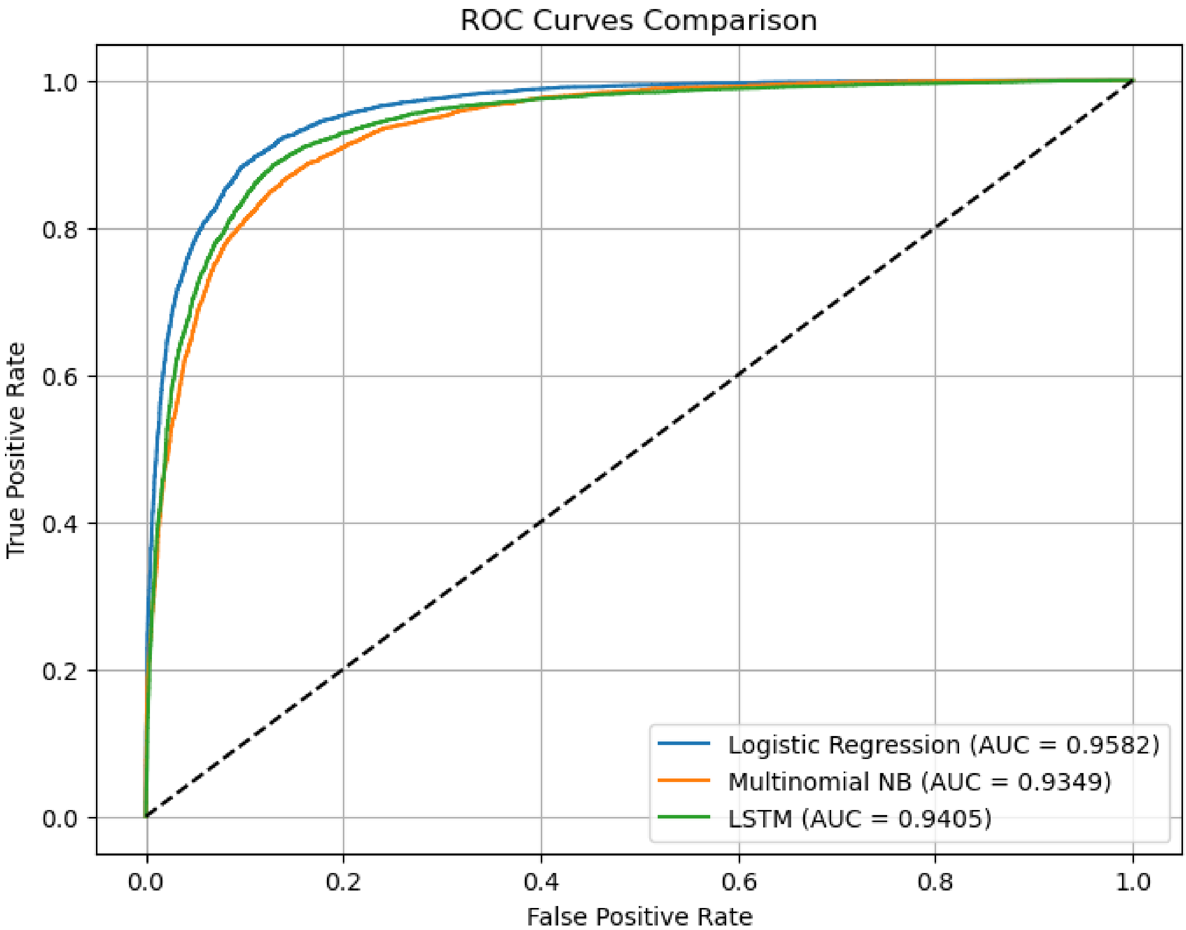


Figure 10: ROC Curves Comparison

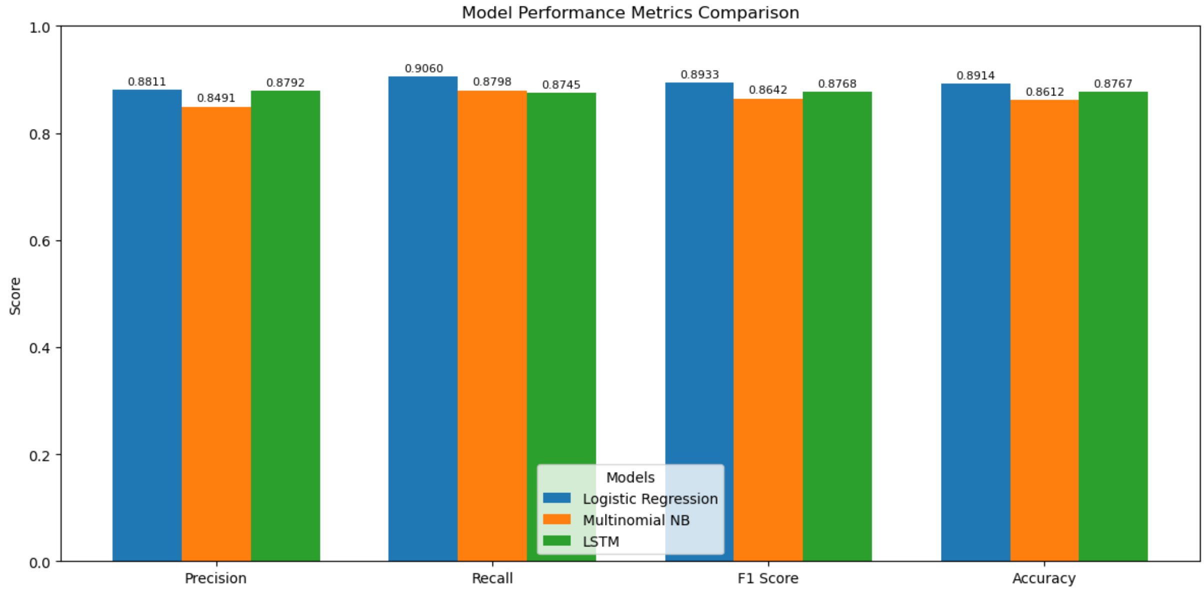


Figure 11: Model Performance Metrics

8.3 Overfitting Assessment

- Logistic Regression: Train 93.61%, Test 89.14% (gap 4.47%).
- Multinomial Naive Bayes: Train 87.27%, Test 86.12% (gap 1.14%).
- LSTM: Train 94.91%, Test 87.67% (gap 7.24%).

9 Limitations

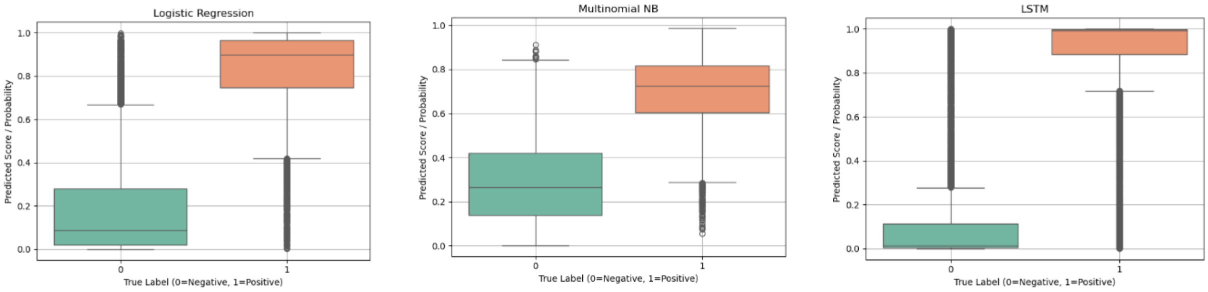
- **Sarcasm Detection**: The models struggle with sarcastic reviews, misclassifying nuanced sentiments.
- **Dataset Bias**: The IMDb dataset may reflect specific demographics, limiting generalizability.
- **Computational Cost**: LSTM training (441 seconds/epoch) is resource-intensive.
- **Aspect Analysis**: Keyword-based aspect detection misses complex contexts.

10 Ethical Considerations

The dataset may contain cultural or demographic biases, as IMDb reviews are user-generated and may not represent diverse perspectives. Deploying these models in real-world applications (e.g., review aggregation) risks amplifying biases if negative sentiments disproportionately affect certain genres or directors. Transparency in model decisions and regular bias audits are essential for ethical use.

11 Conclusions and Future Work

Logistic Regression outperformed (89.14% accuracy), followed by LSTM (87.67%) and Naive Bayes (86.12%). Aspect-based analysis revealed positive sentiments dominate for acting (60.2%).



(a) Logistic Regression (b) Multinomial NB (c) LSTM

Figure 12: Distribution of predicted probabilities for each model.

A Supplementary Code

```
CHAT_WORDS = {'LOL': 'Laughing Out Loud', 'BRB': 'Be Right Back'}
def preprocess_pipeline(df):
    df['cleaned_review'] = (
        df['review']
        .str.lower()
        .apply(convert_chat_words)
        .apply(lambda x: BeautifulSoup(x, 'html.parser').get_text())
        .str.replace(r'http\S+|www\S+|https\S+', '', regex=True)
        .str.replace(r'[^a-zA-Z\s]', '', regex=True)
        .str.replace(r'\s+', ' ', regex=True)
        .apply(lambda x: ' '.join([word for word in word_tokenize(x) if
           word not in stopwords.words('english')]))
        .apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in
           word_tokenize(x)]))
    )
    df['sentiment_count'] = df['sentiment'].map({'positive': 1,
       'negative': 0})
    return df
```