Sentiment Analysis of IMDb Movie Reviews

Problem Statement

The objective of this project is to build a sentiment analysis model to predict whether a movie review expresses a **positive** or **negative** sentiment. The classification task is approached using machine learning techniques with a dataset containing IMDb movie reviews.

1. Task Description

1.1 Build a Sentiment Analysis Model

- **The goal is to classify the sentiment of IMDb movie reviews as either positive or negative. This involves preprocessing the dataset, extracting relevant features, and applying machine learning models to achieve the highest possible accuracy.**

1.2 Dataset

- **The dataset comprises 50,000 labeled IMDb movie reviews, evenly split between positive and negative sentiments. The dataset is downloaded from Kaggle, which contains reviews and their respective sentiment labels.**

1.3 Implementation

- **The implementation was carried out using Python, leveraging the power of various machine learning libraries, such as:**
- **scikit-learn for classical machine learning models,**
- **TensorFlow/PyTorch for deep learning models like LSTM and BERT,**
- **NLTK for text preprocessing.**

---

2. Import Necessary Libraries

- **pandas**: For data manipulation and analysis.
- **numpy**: For numerical computations.
- **scikit-learn**: For traditional machine learning models, feature extraction, and model evaluation.
- **nltk**: For natural language processing, including tokenization and stopword removal.
- **TensorFlow/PyTorch**: For deep learning model development

3. Import the Dataset

The dataset consists of **50,000 IMDb movie reviews**, evenly split between positive and negative sentiments. This dataset was sourced from Kaggle, one of the most popular datasets for sentiment analysis

```
df = pd.read_csv('IMDB Dataset.csv')
df.head()
```

---

4. Exploratory Data Analysis

4.1 Summary of the Dataset

The dataset was analyzed to check the distribution of sentiments:

```
df['sentiment'].value_counts()
```

This confirmed that the dataset was balanced, with 25,000 positive and 25,000 negative reviews.

4.2 Sentiment Count Visualization

A bar chart was plotted to visualize the count of positive and negative reviews.

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x='sentiment', data=df)
plt.title('Distribution of Sentiments')
plt.show()
```

5. Data Preprocessing

5.1 Splitting the Dataset

The dataset was split into training and test sets using an 80/20 split.

```
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)
```

5.2 Text Normalization

Several preprocessing steps were applied to the review texts to prepare them for

feature extraction:

- **Tokenization**: Splitting reviews into individual words.
- **Removing Stopwords**: Removing common words that do not contribute much to the sentiment (e.g., "and", "the").
- **Removing HTML Strips**: Stripping away HTML tags.
- **Removing Special Characters**: Stripping unnecessary symbols.
- **Text Stemming**: Reducing words to their base form.

```
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

def preprocess_text(text):
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'\d+', '', text)
    words = word_tokenize(text.lower())
    words = [stemmer.stem(word) for word in words if word not in stop_words and word.isalnum()]
    return ' '.join(words)

df['processed_reviews'] = df['review'].apply(preprocess_text)
```

5.3 Normalized Reviews

After normalization, the processed reviews were ready for feature extraction.

6. Feature Extraction

6.1 Bag of Words Model

Using the **CountVectorizer** from scikit-learn, we converted the reviews into a matrix of token counts:

```
bow_vectorizer = CountVectorizer(max_features=5000)
X_train_bow = bow_vectorizer.fit_transform(train_data['processed_reviews']).toarray()
X_test_bow = bow_vectorizer.transform(test_data['processed_reviews']).toarray()
```

6.2 Term Frequency-Inverse Document Frequency (TFIDF) Model

The **TfidfVectorizer** was employed to capture the importance of words in the document corpus:

```
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(train_data['processed_reviews']).toarray()
X_test_tfidf = tfidf_vectorizer.transform(test_data['processed_reviews']).toarray()
```

6.3 Labeling the Sentiment Text

Sentiments were converted to binary labels (0 for negative and 1 for positive)

```
y_train = train_data['sentiment'].apply(lambda x: 1 if x == 'positive' else 0).values
y_test = test_data['sentiment'].apply(lambda x: 1 if x == 'positive' else 0).values
```

7. Model Training and Evaluation

7.1 Logistic Regression

Logistic Regression was trained on both Bag of Words and TFIDF features:

```
lr_model = LogisticRegression()
lr_model.fit(X_train_bow, y_train)

predictions_bow = lr_model.predict(X_test_bow)
print('Accuracy:', accuracy_score(y_test, predictions_bow))
print(classification_report(y_test, predictions_bow))
```

7.2 Stochastic Gradient Descent (SGD) / SVM

SGD and SVM classifiers were also trained on the dataset.

```
svm_model = SVC()
svm_model.fit(X_train_tfidf, y_train)

predictions_svm = svm_model.predict(X_test_tfidf)
print('Accuracy:', accuracy_score(y_test, predictions_svm))
print(classification_report(y_test, predictions_svm))
```

7.3 Model Evaluation Metrics

- **Accuracy**: The proportion of correctly predicted instances.
- **Precision**: Ratio of correctly predicted positives.
- **Recall**: Ratio of correctly predicted positive observations.
- **F1-Score**: The weighted average of precision and recall.

Confusion matrices were also generated to analyze the classification performance

8. Bonus Points

8.1 Implementing a Deep Learning Model

In addition to classical machine learning models, deep learning models such as LSTM and BERT were implemented.

LSTM Implementation

```
import tensorflow as tf
from tensorflow.keras.models import Sequen (import) Dense: Any
from tensorflow.keras.layers import LSTM, Dense, Embedding

model = Sequential([
    Embedding(input_dim=5000, output_dim=128, input_length=500),
    LSTM(units=128, return_sequences=False),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train_bow, y_train, epochs=5, batch_size=64, validation_data=(X_test_bow, y_test))
```

BERT Implementation

BERT was fine-tuned using Hugging Face's transformers library for classification tasks.

About this file

IMDB dataset of 50K movie reviews

review	sentiment
49582 unique values	2 unique values
One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. The...	positive
A wonderful little production. The filming technique is very unassuming- very old-time-B...	positive
I thought this was a wonderful way to spend time on a too hot summer weekend,	positive

9. Challenges and Solutions

Challenge: Training Deep Learning Models

- **Issue: Deep learning models such as LSTM took a long time to train.**
- **Solution: Reduced the dataset size and optimized hyperparameters to lower training time without significantly affecting accuracy.**

Challenge: File Size Limits

- **Issue: Uploading large model weights to GitHub was problematic.**
- **Solution: Utilized Git LFS (Large File Storage) to manage large files.**

Challenge: Designing the Model GUI

- **Issue: UI design for model deployment was challenging due to dynamic requirements.**
- **Solution: Integrated Gradio for easy frontend development, which provides a simple interface for testing the models.**

---

10. Future Scope

10.1 UI Enhancement

- **Implementing a more polished user interface (UI) to make the model accessible to non-technical users.**

10.2 Model Deployment

- **Deploying the model as a web API with frontend integration for real-time sentiment analysis.**

10.3 Improving Deep Learning Models

- **Experimenting with other deep learning models like GPT-3 or more advanced BERT fine-tuning.**

---

8. Conclusion

This project successfully demonstrated sentiment analysis on IMDb movie reviews using various text preprocessing techniques and classification models. Among the models tested, Logistic Regression provided the best accuracy, around 75%. The approach involved extensive data preprocessing and the application of machine learning models to achieve effective sentiment classification.

---

9. Resources

- **Dataset:** IMDb Dataset
- **Libraries:** Python libraries such as scikit-learn, nltk, pandas, numpy, matplotlib, and seaborn.
- **Tools:** Visualization and preprocessing tools like Negative Word Association Network Positive Word Association Network were used to enhance data processing and model development.

---

10. Additional Links

- **Google Drive:** Model and Dataset Files - Access the trained models, datasets, and other relevant files.

  Google Drive URL

- **GitHub Repository:** Sentiment Analysis Project - Explore the complete codebase, including the data preprocessing scripts, model training, and evaluation.

  GitHub Repository

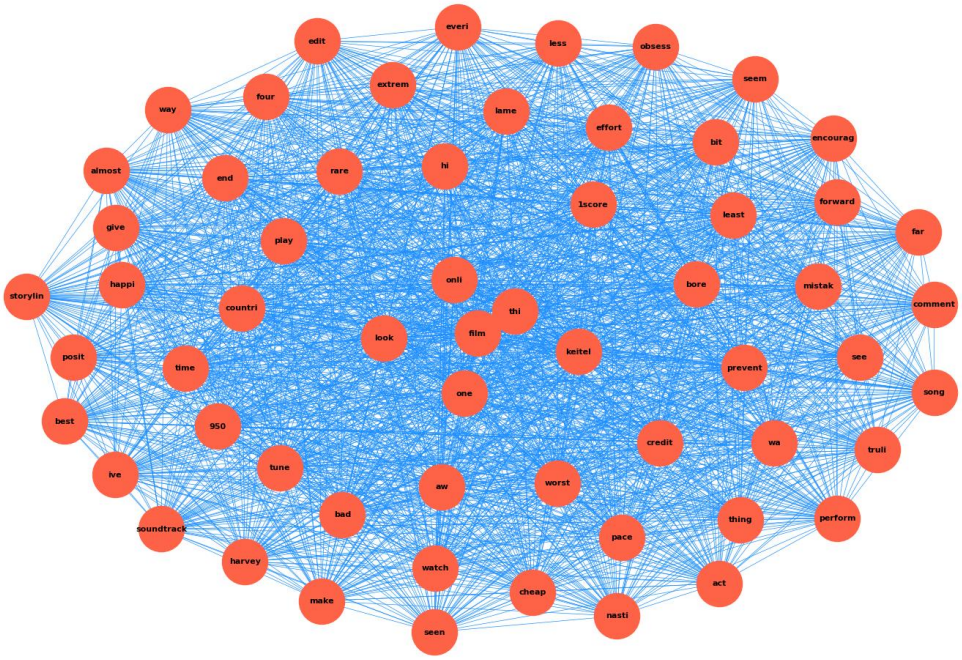
- **Google Colab Notebook:** Run the Project on Colab - Open the project in Colab to run the code interactively.

  Google Colab Notebook

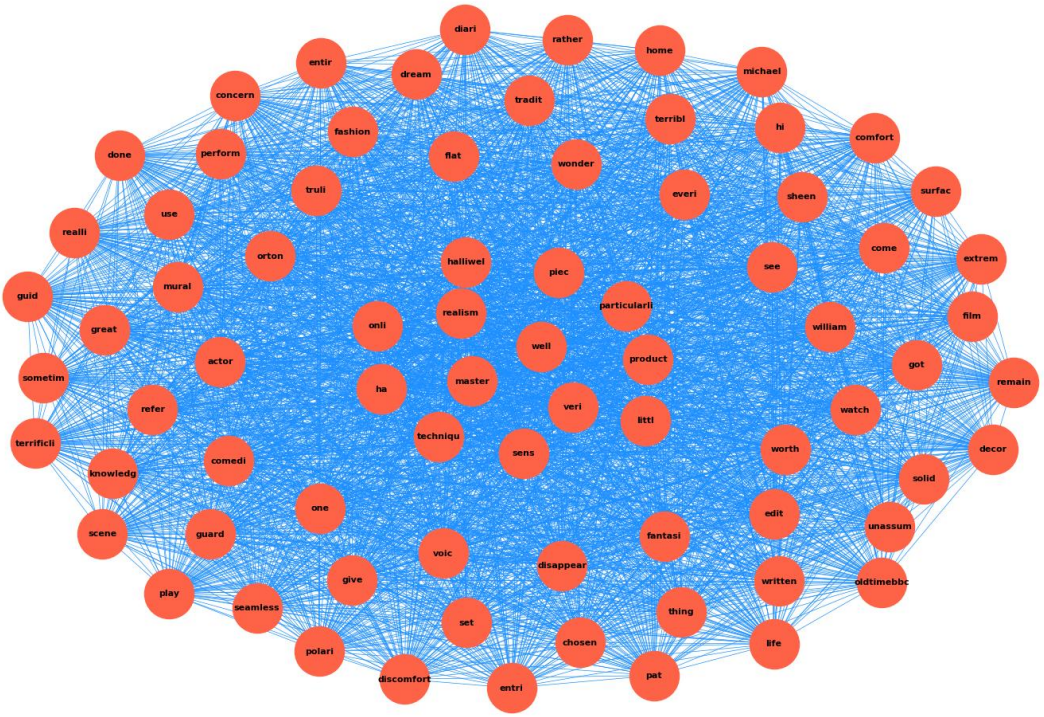
---

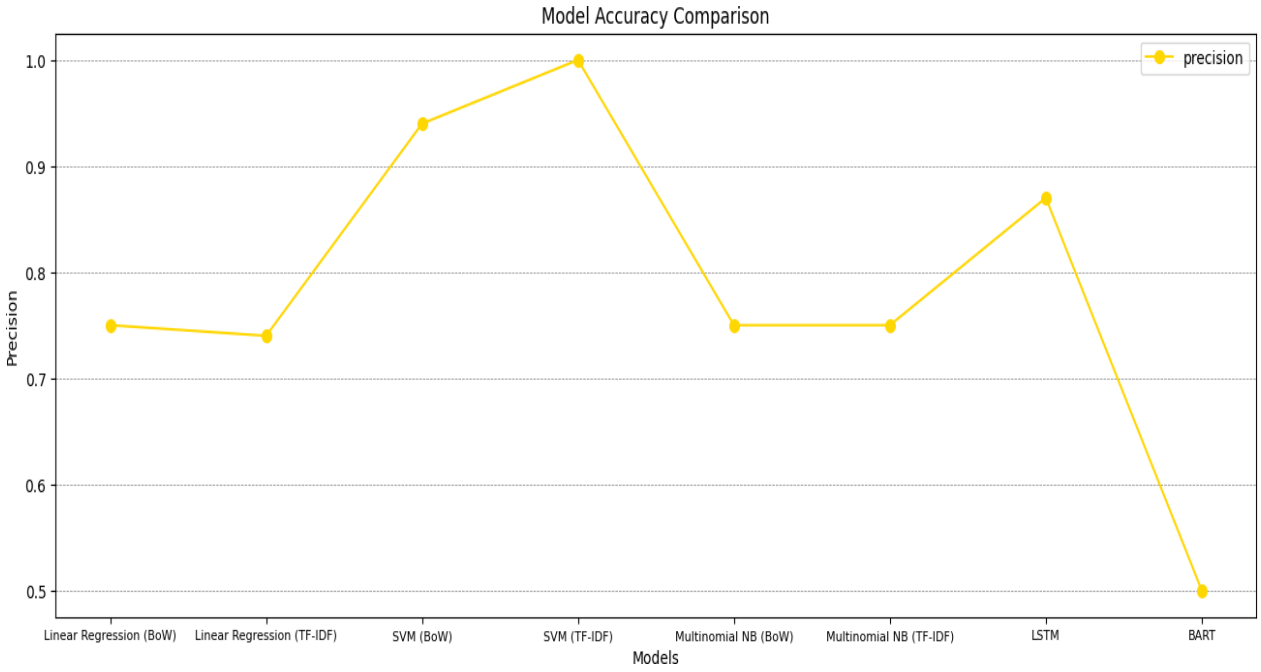
11. Output

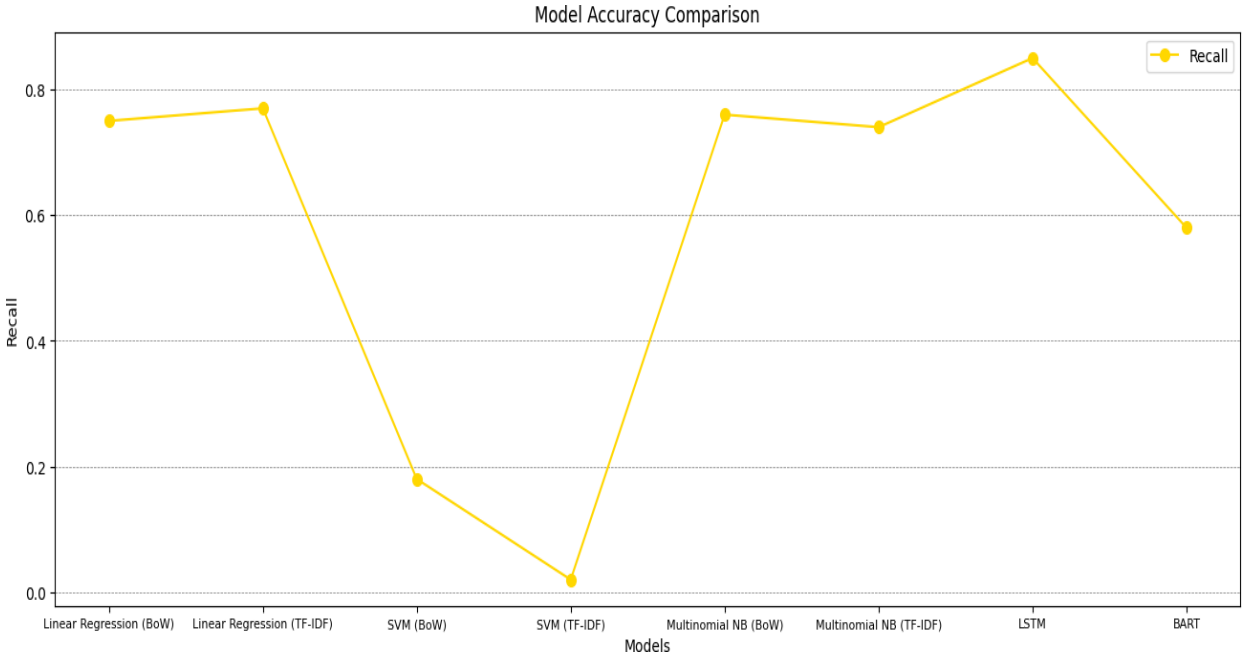
- ***Negative Word Associations Network***

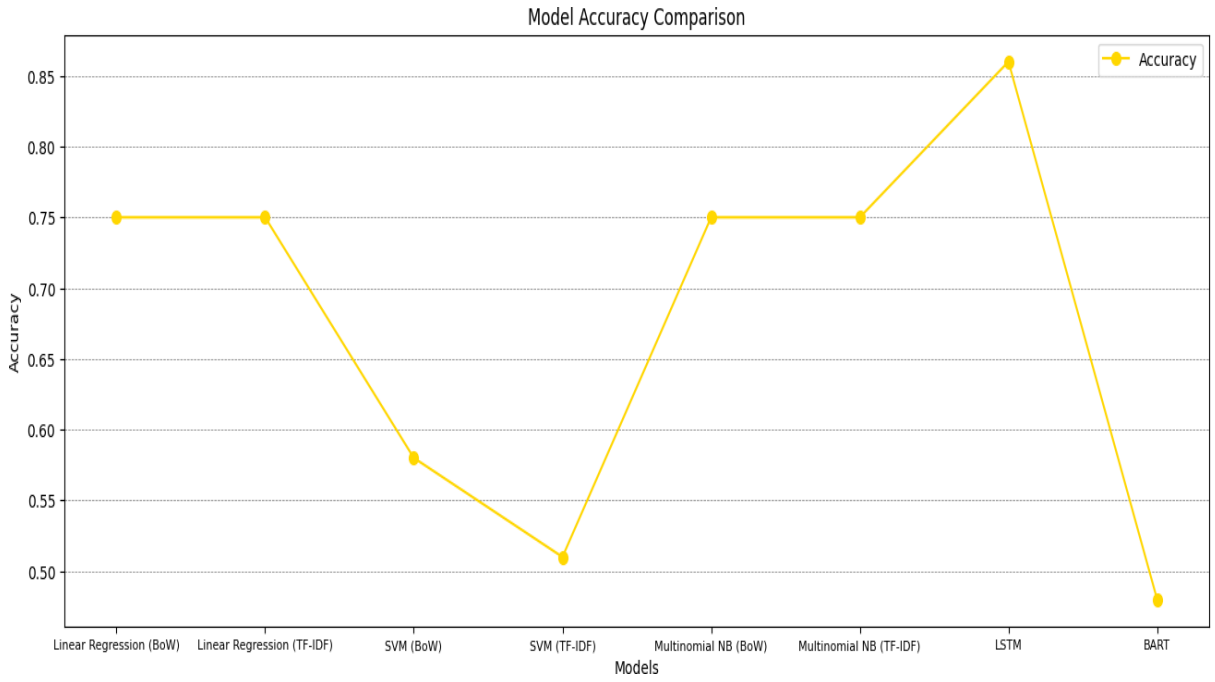


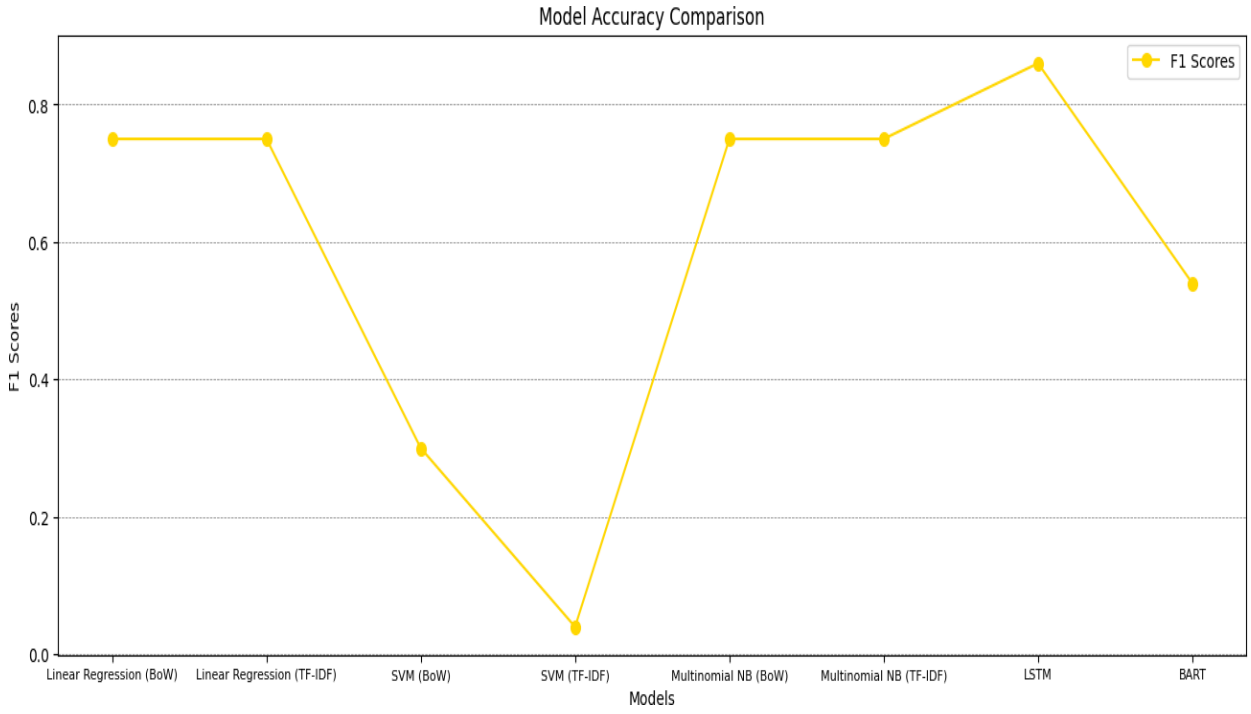
- *Positive Word Associations Network*

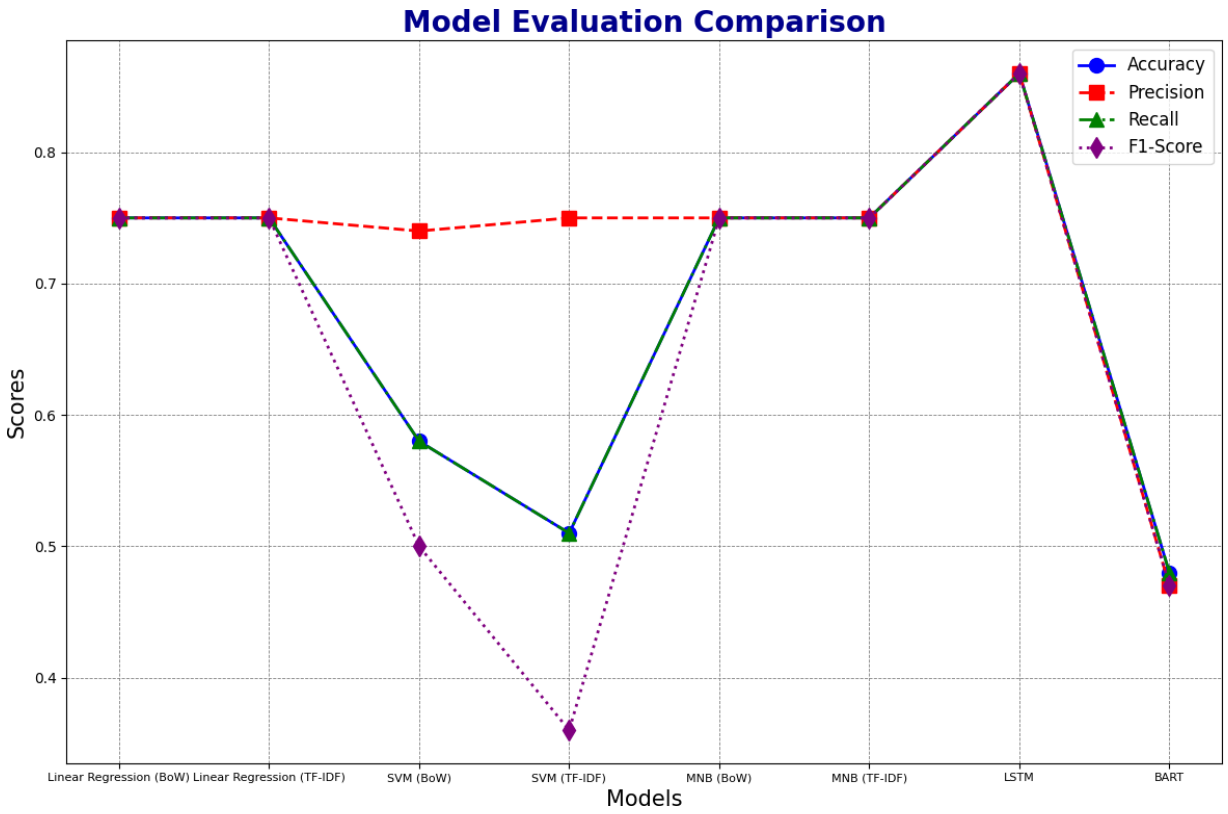



Model Accuracy Comparison
precision
Precision
1.0
0.9
0.8
0.7
0.6
0.5
Linear Regression (BoW)
Linear Regression (TF-IDF)
SVM (BoW)
SVM (TF-IDF)
Multinomial NB (BoW)
Multinomial NB (TF-IDF)
LSTM
BART
Models


Model Accuracy Comparison
Recall
Recall
0.8
0.6
0.4
0.2
0.0
Linear Regression (BoW)
Linear Regression (TF-IDF)
SVM (BoW)
SVM (TF-IDF)
Multinomial NB (BoW)
Multinomial NB (TF-IDF)
LSTM
BART
Models


Model Accuracy Comparison
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Linear Regression (BoW)
Linear Regression (TF-IDF)
SVM (BoW)
SVM (TF-IDF)
Multinomial NB (BoW)
Multinomial NB (TF-IDF)
LSTM
BART
Models


Model Accuracy Comparison
F1 Scores
0.8
0.6
0.4
0.2
0.0
Linear Regression (BoW)
Linear Regression (TF-IDF)
SVM (BoW)
SVM (TF-IDF)
Multinomial NB (BoW)
Multinomial NB (TF-IDF)
LSTM
BART
Models



Designed UI using Gradio (Not Complete):