Name: Parsa Mazaheri

Data Science and Machine Learning Fundamentals

1 Amazon Book Reviews

For starters, we download the data from Github as mentioned in the problem. If we look into the data, we see that the dataset comprises reviews of books from Amazon, where each entry has a review content (review/text), a score (review/score ranging from 1 to 5 stars), a book title (Title), and a review summary (review/summary).

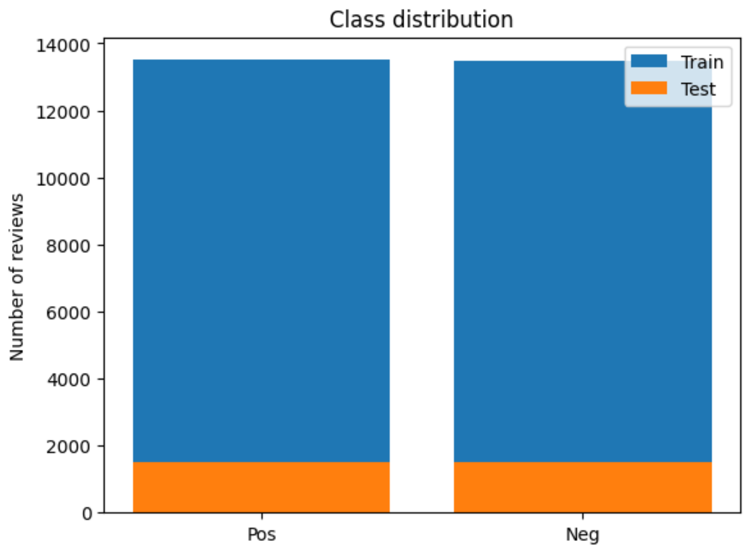
1.1 Data Preprocessing

For Binary classification, we do the following for splitting the positive and negative labels.
- The `review/score` was mapped to a binary scale:
- 4 or higher: Positive (1)
- 2 or lower: Negative (0)
- Reviews with a score of 3 were excluded from the analysis.

After this, we split the data from train and test. For this we use 10% of the data for the test and 90% for the training using a fixed random seed.

Table 1: Data distribution

Data Set	**Positive (Pos)**	**Negative (Neg)**
Train Data	13506	13494
Test Data	1506	1494



1.2 Feature Engineering Techniques

Three feature engineering techniques were applied:

1. Bag of Words (BoW): Represents text by the frequency of words, disregarding order and context.

2. Term Frequency-Inverse Document Frequency (TF-IDF): Weights words based on their importance in a document relative to the entire corpus, providing a balance between term frequency and its rarity across documents.

3. Word2Vec: Word2Vec creates embeddings of words in a high-dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space.

Discussion:

- **Bag of Words**: Converts text into a matrix of token occurrences.
- **TF-IDF**: Weighs terms by their importance relative to the entire corpus.
- **Word2Vec**: Embeds words in a vector space based on their contextual meaning.

For each engineered feature, two classification models were trained:

- **Naive Bayes**: A probabilistic model based on Bayes' theorem which is Ideal for text data due to its ability to handle large feature spaces.
- **SVM**: Finds the hyperplane that best separates the classes in the feature space.

By running the models on the our data for the six possible way that we have. we get these results shown in the table below.

Table 2: Model Evaluation Metrics

Model	Accuracy	F1 Score
Naive Bayes with BoW	0.8620	0.8542
SVM with BoW	0.8540	0.8556
Naive Bayes with TF-IDF	0.8766	0.8732
SVM with TF-IDF	0.9003	0.9003
Naive Bayes with Word2Vec	0.6573	0.6415
SVM with Word2Vec	0.7733	0.7725

TF-IDF Outperforms BoW:

- TF-IDF gives weightage to important words. It reduces the impact of frequently occurring words that are less informative than rarer words.
- It captures the importance of different terms relative to all documents, not just their presence or absence.

SVM vs Naive Bayes:

- SVM with TF-IDF gives the best results among all models. This indicates that the decision boundary might be complex and SVM does a good job capturing this.
- Naive Bayes assumes feature independence, which might not always be the case with text data.

Word2Vec's Lower Performance:

- Word2Vec embeds words based on their contextual semantics. This can sometimes introduce noise if the dataset doesn't provide a rich semantic context or if the embeddings aren't trained well.
- Given its lower score, it seems our Word2Vec model might not have been well-optimized or we can say the dataset wasn't vast enough to capture nuanced embeddings.

General:

- Even if one technique generally works well, dataset-specific characteristics can cause performance variations. - Fine-tuning and further optimization can potentially improve results, especially for Word2Vec.