Text Analysis of Amazon Fine Food Reviews Dataset

1 Introduction

This report presents a sentiment analysis on the Amazon Fine Food Reviews dataset, where reviews are classified into positive or negative sentiments. The dataset includes over 500,000 reviews with scores ranging from 1 to 5. The aim is to preprocess the reviews, apply various machine learning models, and evaluate their performance in classifying the sentiment of the reviews.

2 Text Preprocessing

The text preprocessing steps involved:

- **Missing Values**: The `Summary` and `Text` columns were concatenated, and missing values were filled with empty strings.
- **Neutral Score Removal**: Reviews with a score of 3 (neutral) were removed.
- **Sentiment Mapping**: Scores 1 and 2 were mapped to `0` (negative) and scores 4 and 5 to `1` (positive).
- **Column Dropping**: Irrelevant columns were removed.

Code for Preprocessing:

```
df = df[df['Score'] != 3]
df['sentiment'] = df['Score'].apply(lambda x: 1 if x > 3 else 0)
df = df.drop(columns=['Id', 'ProductId', 'UserId', 'ProfileName',
                      'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time'])
df['text'] = df['Summary'] + ' ' + df['Text']
```

3 Exploratory Data Analysis (EDA)

3.1 Distribution of Text Lengths by Sentiment

We examined the distribution of review lengths by sentiment using histograms for positive and negative reviews. This provides insights into the average length of reviews based on sentiment.

```
fig,(ax1,ax2) = plt.subplots(1,2,figsize=(10,8))
ax1.hist(df[df['sentiment']==1]['text'].str.len(),color='green')
ax1.set_title('Positive Reviews')
ax2.hist(df[df['sentiment']==0]['text'].str.len(),color='orange')
ax2.set_title('Negative Reviews')
plt.show()
```

3.2 Word Cloud Visualization

Word clouds were generated to visually represent the most frequent words in positive and negative reviews.

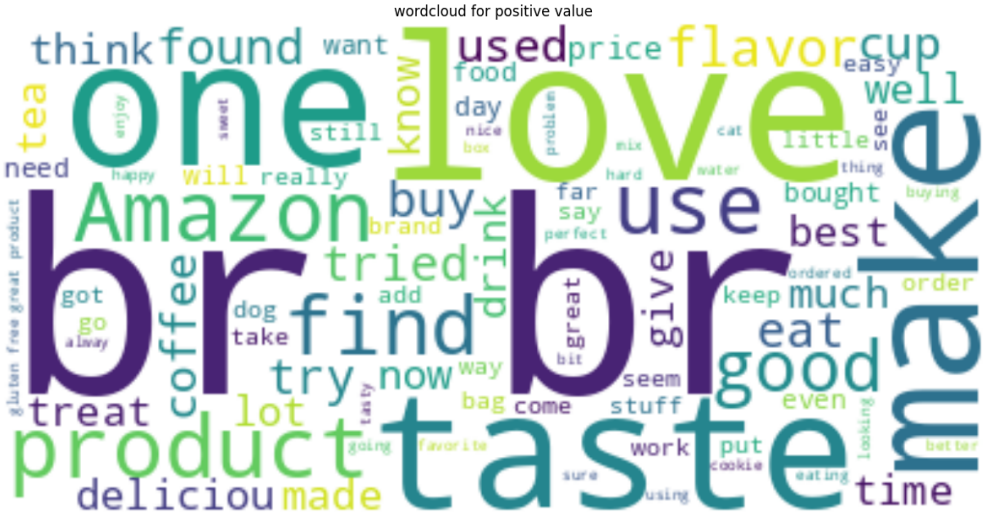

Figure 1: Word Cloud for Positive Reviews

Figure 2: Word Cloud for Negative Reviews

4 Common Words Analysis

We identified the most frequent words in the reviews using the `Counter` function. These words give insights into the commonly used terms across reviews.

Code to Identify Common Words:

```
cnt = Counter()
for text in df["text"].values:
    for word in text.split():
        cnt[word] += 1
```

```
temp = pd.DataFrame(cnt.most_common(10))
temp.columns = ['word', 'count']
px.bar(temp, x="count", y="word", title='Common Words in Reviews', orientation='h')
```

5 Model Training and Evaluation

We trained and evaluated three different models: Logistic Regression, Complement Naive Bayes, and Multinomial Naive Bayes. The text data was vectorized using TF-IDF, and the data was split into training and test sets.

5.1 TF-IDF Vectorization

The text was transformed into numerical data using TF-IDF (Term Frequency-Inverse Document Frequency).

```
word_vector = TfidfVectorizer()
X = word_vector.fit_transform(df['text'].values)
y = df['sentiment'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5.2 Logistic Regression

Logistic Regression is a linear model for binary classification. It was trained on the TF-IDF-transformed data.

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print(classification_report(y_test, y_pred))
```

Logistic Regression Performance:

- **Accuracy:** 95.14%
- **Precision, Recall, F1-Score (weighted average):**
  - Precision: 0.95
  - Recall: 0.95
  - F1-Score: 0.95

5.3 Complement Naive Bayes

Complement Naive Bayes is particularly effective for imbalanced datasets.

```
CNB = ComplementNB()
CNB.fit(X_train, y_train)
predicted = CNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, y_test)
print(f'ComplementNB Accuracy: {accuracy_score * 100:.2f}%')
print(classification_report(y_test, predicted))
```

Complement Naive Bayes Performance:

- **Accuracy:** 90.24%
- **Precision, Recall, F1-Score (weighted average):**
  - Precision: 0.90
  - Recall: 0.90
  - F1-Score: 0.88

5.4 Multinomial Naive Bayes

Multinomial Naive Bayes was also trained and evaluated.

```
MNB = MultinomialNB()
MNB.fit(X_train, y_train)
predicted = MNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, y_test)
print(f'MultinomialNB Accuracy: {accuracy_score * 100:.2f}%')
print(classification_report(y_test, predicted))
```

Multinomial Naive Bayes Performance:

- **Accuracy:** 86.90%
- **Precision, Recall, F1-Score (weighted average):**
  - Precision: 0.88
  - Recall: 0.87
  - F1-Score: 0.83

6 Model Comparison

Model	Accuracy	Precision (Weighted Avg)	Recall (Weighted Avg)	F1-Score (Weighted Avg)
Logistic Regression	95.14%	0.95	0.95	0.95
Complement Naive Bayes	90.24%	0.90	0.90	0.88
Multinomial Naive Bayes	86.90%	0.88	0.87	0.83

Table 1: Model Performance Summary

7 Conclusion

- Logistic Regression had the best performance with an accuracy of 95.14% and well-balanced precision and recall.
- Complement Naive Bayes also performed well, especially for imbalanced data.
- Multinomial Naive Bayes performed slightly lower, especially in detecting negative reviews.
- Future work could involve experimenting with deep learning models like LSTM or BERT to improve accuracy further.