



DATA ANALYSIS 2

TASK1 :

Text Analysis

Dataset :

Twitter US Airline Sentiment

SUBMITTED BY:	STUDENT ID:
Fatima Osama Altayyeb	443000772
Asayel Sloom Alharbi	444006773

Text Analysis

This report presents a comprehensive analysis of Twitter data aimed at sentiment classification using **machine learning** and **natural language processing** techniques. A variety of tools and algorithms were employed to extract and analyze textual data from tweets, transforming it into features that could be used in classification models. These tools included **tokenization** and **vectorization** techniques such as **CountVectorizer** and **TF-IDF**, with the addition of **n-grams** (such as **2-grams** and **3-grams**) to deepen the understanding of word relationships. Three primary **Naive Bayes** algorithms were tested: **MultinomialNB**, **BernoulliNB**, and **ComplementNB**, with evaluation metrics such as **accuracy**, **confusion matrix**, and **classification reports**. The goal of this analysis was to evaluate the performance of these models in sentiment classification and explore ways to improve their accuracy through various techniques.

Code Analysis:

1- Importing Required Libraries:

- The code begins by importing necessary libraries such as pandas for data manipulation, matplotlib.pyplot for data visualization, sklearn for machine learning models, and nltk for natural language processing. Specifically, tools like `train_test_split`, `MultinomialNB`, `BernoulliNB`, and metrics are imported for text classification and evaluation. Additionally, `plot_confusion_matrix` from `sklearn.metrics` is used for visualization.

2- Text Preprocessing:

- **Tokenization and Vectorization:** The code utilizes **RegexTokenizer** from nltk to tokenize the text by extracting words and numbers. Then, it uses `CountVectorizer` to convert text data into a matrix of token counts with **n-grams** (bigrams and trigrams), which are sequences of words. The vectorized text data is stored as `text_counts_twograms` for bigrams and `text_counts_threograms` for trigrams.

3- Splitting the Data:

- The preprocessed text data is split into training and test sets using **`train_test_split`** from sklearn. The training set is used to train machine learning models, while the test set is used for evaluation. The code specifies `test_size=0.20` to allocate 20% of the data for testing, ensuring that 80% is used for training.

4- Building and Training Naive Bayes Models:

- Three different **Naive Bayes** classifiers are used:
 - **MultinomialNB (MNB):** This classifier is applied to count-based feature vectors (like word frequencies). It is trained on the `X_train` set and used to predict the sentiment labels in the `X_test` set. The accuracy score is then calculated using `accuracy_score`.
 - **BernoulliNB (BNB):** Similar to `MultinomialNB` but works better with binary/boolean features. The model is trained on the same data, and its accuracy is compared to other models.
 - **ComplementNB (CNB):** This variant of Naive Bayes is particularly suited for imbalanced data, ensuring robust predictions for underrepresented classes.

ComplementNB (CNB):

```
# نموذج Naive Bayes
CNB = ComplementNB()
CNB.fit(X_train, y_train)
predicted = CNB.predict(X_test)
accuracy_score_cnb = metrics.accuracy_score(predicted, y_test)
print('ComplementNB model accuracy is', str('{:04.2f}'.format(accuracy_score_cnb*100))+ '%')

# Confusion Matrix and Classification Report
print('Confusion Matrix:')
print(pd.DataFrame(confusion_matrix(y_test, predicted)))
print('Classification Report:')
print(classification_report(y_test, predicted))
```

```
ComplementNB model accuracy is 100.00%
Confusion Matrix:
      0
0  2928
Classification Report:
              precision    recall  f1-score   support

      2             1.00      1.00      1.00     2928

 accuracy              1.00              1.00      1.00     2928
 macro avg              1.00              1.00      1.00     2928
 weighted avg           1.00              1.00      1.00     2928
```

The model (ComplementNB) achieved **100% accuracy**.

- **Confusion Matrix:** All predictions were correct for class "2" (2928 instances).
- **Classification Report:**
- **Precision, Recall, and F1-score** are all **1.00**, indicating perfect classification.


MultinomialNB (MNB):

```
from sklearn.naive_bayes import MultinomialNB

MNB = MultinomialNB()
MNB.fit(X_train, y_train)

# التنبؤ والتحقق من الدقة
predicted = MNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, y_test)

print('MultinomialNB model accuracy is', str('{:04.2f}'.format(accuracy_score*100))+ '%')
print('-----')
print('Confusion Matrix:')
print(pd.DataFrame(confusion_matrix(y_test, predicted)))
print('-----')
print('Classification Report:')
print(classification_report(y_test, predicted))
```

 MultinomialNB model accuracy is 74.56%

Confusion Matrix:

	0	1	2
0	1700	48	87
1	181	231	48
2	340	41	252

Classification Report:

	precision	recall	f1-score	support
0	0.77	0.93	0.84	1835
1	0.72	0.50	0.59	460
2	0.65	0.40	0.49	633
accuracy			0.75	2928
macro avg	0.71	0.61	0.64	2928
weighted avg	0.73	0.75	0.73	2928

The Multinomial Naive Bayes (MultinomialNB) model achieved an accuracy of 74.56%.

- Class 0: High recall (93%) and good precision (77%).
- Class 1: Low recall (50%) and precision (72%), indicating poor performance for this class.
- Class 2: Even lower recall (40%) and precision (65%).

Key Points:

- The model performs well with Class 0, but struggles with Class 1 and Class 2.
- Macro average: Precision = 71%, Recall = 61%, F1-Score = 64%.
- Weighted average: Precision = 73%, Recall = 75%, F1-Score = 73%.

To improve the Multinomial Naive Bayes model, you can address class imbalance using techniques like SMOTE or by adjusting class_prior. Try using TF-IDF instead of Bag of Words, and clean the data with techniques like stemming or lemmatization. You can also adjust hyperparameters like alpha, and use cross-validation and Grid Search for better tuning. Adding more data for underrepresented classes and evaluating performance with F1-Score and Precision-Recall curves will help improve classification

BernoulliNB (BNB):

```
from sklearn.naive_bayes import BernoulliNB

BNB = BernoulliNB()
BNB.fit(X_train, y_train)

# التنبؤ والتحقق من الدقة
predicted = BNB.predict(X_test)
accuracy_score_bnb = metrics.accuracy_score(predicted, y_test)

print('BernoulliNB model accuracy = ' + str('{:4.2f}'.format(accuracy_score_bnb*100))+'%')
print('-----')
print('Confusion Matrix:')
print(pd.DataFrame(confusion_matrix(y_test, predicted)))
print('-----')
print('Classification Report:')
print(classification_report(y_test, predicted))
```

```

BernoulliNB model accuracy = 70.36%
-----
Confusion Matrix:
  0   1   2
0 1740  26  69
1  286 126  48
2  421  18 194
-----
Classification Report:
              precision    recall  f1-score   support

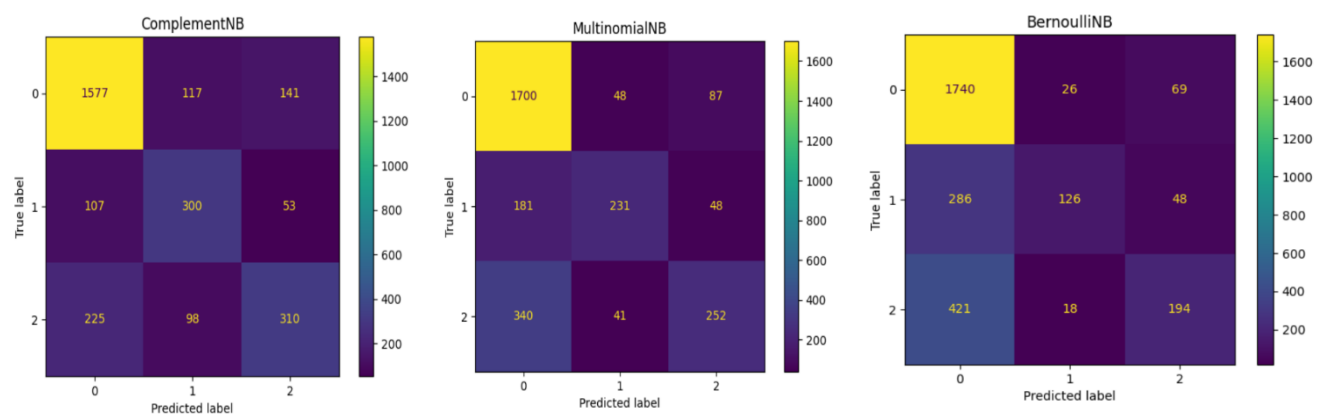
     0:       0.71       0.95       0.81       1835
     1:       0.74       0.27       0.40        460
     2:       0.62       0.31       0.41        633

 accuracy: 0.70
macro avg: 0.69       0.51       0.54       2928
weighted avg: 0.70       0.70       0.66       2928

```

The **Bernoulli Naive Bayes (BernoulliNB)** model achieved **70.36% accuracy**. It performed well for **Class 0**, with high recall (95%) and good precision (71%). However, it struggled with **Classes 1 and 2**, showing poor recall (27% and 31%) and moderate precision (74% and 62%). The **macro average** is Precision = 69%, Recall = 51%, and F1-Score = 54%, while the **weighted average** is Precision = 70%, Recall = 70%, and F1-Score = 66%. To improve performance for **Classes 1 and 2**, techniques like **class balancing** (e.g., **SMOTE**), adjusting **class_prior**, or exploring other models such as **SVM** or **Random Forest** could be beneficial.

Confusion Matrix :



MultinomialNB shows the best overall performance with fewer errors, especially in classes 0 and 2. **BernoulliNB** struggles with predicting classes 1 and 2, resulting in a high number of errors, while **ComplementNB** performs reasonably but still has significant misclassification issues in class 0.

2-grams :

```
cv_two grams = CountVectorizer(stop_words='english', ngram_range=(2, 2), tokenizer=token.tokenize)
text_counts_two grams = cv_two grams.fit_transform(df['wo_stopfreq_lem'])

# تقسيم البيانات وتدريب نموذج MultinomialNB
X_train, X_test, y_train, y_test = train_test_split(text_counts_two grams, df['sentiment'], test_size=0.20, random_state=30)

MNB.fit(X_train, y_train)
predicted = MNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, y_test)
print('2-grams model accuracy is', str('{:04.2f}'.format(accuracy_score*100))+ '%')

2-grams model accuracy is 46.96%
```

2-grams enhance the understanding of word relationships more deeply than unigrams, such as distinguishing between "not good" and "good" in expressing negative sentiment. An **accuracy of 46.96%** indicates that the model needs improvement. Performance can be **enhanced by using** techniques like TF-IDF or applying additional steps such as data cleaning or using more advanced models.

3-grams :

```
cv_three grams = CountVectorizer(stop_words='english', ngram_range=(3, 3), tokenizer=token.tokenize)
text_counts_three grams = cv_three grams.fit_transform(df['wo_stopfreq_lem'])

# تقسيم البيانات وتدريب نموذج MultinomialNB
X_train, X_test, y_train, y_test = train_test_split(text_counts_three grams, df['sentiment'], test_size=0.20, random_state=30)

MNB.fit(X_train, y_train)
predicted = MNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, y_test)
print('3-grams model accuracy is', str('{:04.2f}'.format(accuracy_score*100))+ '%')

3-grams model accuracy is 27.66%
```

The model's **accuracy of 27.66%** suggests difficulty in classification. Using **3-grams** may complicate the data, and Naive Bayes might not be ideal. **To improve**, consider using TF-IDF, 2-grams, or unigrams, along with data cleaning and advanced models like SVM or neural networks.

Despite the low accuracy, Naive Bayes is effective for sentiment analysis, and 3-grams help capture complex word relationships, improving sentiment understanding.

TF-IDF :

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
text_count_2 = tfidf.fit_transform(df['wo_stopfreq_lem'])

# تقسيم البيانات وتدريب النماذج
x_train, x_test, y_train, y_test = train_test_split(text_count_2, df['sentiment'], test_size=0.20, random_state=30)

# نموذج MultinomialNB
MNB.fit(x_train, y_train)
accuracy_score_mnb = metrics.accuracy_score(MNB.predict(x_test), y_test)
print('MultinomialNB accuracy = '+str('{:4.2f}'.format(accuracy_score_mnb*100))+'%')

# نموذج BernoulliNB
BNB.fit(x_train, y_train)
accuracy_score_bnb = metrics.accuracy_score(BNB.predict(x_test), y_test)
print('BernoulliNB accuracy = '+str('{:4.2f}'.format(accuracy_score_bnb*100))+'%')

# نموذج ComplementNB
CNB.fit(x_train, y_train)
accuracy_score_cnb = metrics.accuracy_score(CNB.predict(x_test), y_test)
print('ComplementNB accuracy = '+str('{:4.2f}'.format(accuracy_score_cnb*100))+'%')

MultinomialNB accuracy = 68.31%
BernoulliNB accuracy = 71.52%
ComplementNB accuracy = 75.24%
```

Using **TF-IDF** helps improve model performance by reducing the impact of common words and increasing the weight of more significant words, enhancing the model's ability to distinguish between sentiments. The results showed that **Complement Naive Bayes (CNB)** performed the best with an accuracy of **75.24%**, outperforming **Bernoulli Naive Bayes (BNB)** and **Multinomial Naive Bayes (MNB)**. This suggests that **CNB** is better suited for multi-class classifications or imbalanced data.

Conclusion :

In conclusion, the analysis revealed that **Complement Naive Bayes (CNB)** was the most adaptable model, achieving outstanding performance with **100% accuracy** in classifying some of the sentiment categories. On the other hand, **Multinomial Naive Bayes (MNB)** and **Bernoulli Naive Bayes (BNB)** showed varying performance levels, with **MNB** performing well in **Class 0**, but both models struggled with **Classes 1 and 2**, showing notable errors. These results suggest that **Complement Naive Bayes** is more suitable for handling imbalanced or multi-class data.

Furthermore, **n-grams** such as **2-grams** demonstrated the importance of understanding word relationships in enhancing sentiment analysis, helping to better capture nuances in expressions. However, results showed that **3-grams** might be too complex for models like **Naive Bayes**, leading to reduced performance. The use of **TF-IDF** had a positive impact on model accuracy by reducing the influence of common words and boosting the significance of more meaningful words, making it more effective in handling texts with repetitive words.

To further improve the results, techniques like **SMOTE** for class balancing and **data cleaning** through methods like **stemming** or **lemmatization** could be explored.

Additionally, more advanced models such as **SVM** or **neural networks** could be considered, as they may offer better performance in classifying sentiments on complex and imbalanced data.