Naive Bayes algorithm to classify text data

Problem Statement:

Based on the dataset provided, it appears to contain three columns: "id," "text," and "label." The "text" column seems to contain textual data, possibly reviews or descriptions, and the "label" column contains categories, in this case, 'dogs.'

You are provided with a dataset containing various textual entries categorized under different labels. Your task is to develop a model using the Naive Bayes algorithm to classify the text into its appropriate category based on the content. Specifically, focus on preprocessing the text data, converting it into a suitable format for the Naive Bayes model, training the model, and then evaluating its performance on unseen data. Aim to achieve the highest possible accuracy in correctly categorizing the text entries.

Additionally, explore different feature extraction techniques and Naive Bayes variants to understand their impact on the model's performance.

Analysis

The dataset consists of ~ 15,500 rows and has three columns - "id," "text," and "label." After checking for missing values, there are none.

```
Missing Values:
 id       0
text      0
label     0
dtype: int64
```

The label column is encoded into numerical values. After splitting the dataset into training and test datasets using a 70/30% combination, I use 3 different types of feature extraction techniques:

a) **CountVectorizer:** Represents text as a bag-of-words model where each word in the vocabulary is a feature, and its value corresponds to the count of that word in the text.
b) **TF-IDF (Term Frequency-Inverse Document Frequency)**: Scales word frequencies by their importance across all documents in a collection. Rare words get higher weights, while common words are penalized.
c) **CountVectorizer with Bigrams:** Extends the bag-of-words model to include both single words (unigrams) and two-word combinations (bigrams) as features.

Each of these feature extraction techniques are provided max_features as equal to 5,000. In addition to the different feature extraction techniques, I also use both the Multinomial and Bernoulli NaiveBayes algorithm and use a function to iterate through the various combinations

of feature extractions and type of Naive Bayes models to see which gives us the best results. After fitting the training dataset to each combination, I look at the accuracy scores, F1 scores and results from the confusion matrix.

The model performance summary with accuracy and F1 scores are shown in the below table:

```
Model Performance Summary:
                     Vectorizer          Model  Accuracy  F1 Score
0                CountVectorizer  MultinomialNB  0.839785  0.839138
1                CountVectorizer    BernoulliNB  0.814839  0.811601
2                       TF-IDF  MultinomialNB  0.823441  0.818831
3                       TF-IDF    BernoulliNB  0.814839  0.811601
4  CountVectorizer with Bigrams  MultinomialNB  0.841935  0.841126
5  CountVectorizer with Bigrams    BernoulliNB  0.823226  0.819957
```

From the above, we can see the CountVectorizer with Bigrams feature extraction technique in the Multinomial models produces the best results. And overall, Multinomial Naive Bayes consistently performs better than BernoulliNB, achieving higher precision, recall, and F1 scores for both classes

The Confusion matrix for each combination is also shown below:

```
=== CountVectorizer with MultinomialNB ===
Confusion Matrix:
 [[1451  414]
 [ 331 2454]]
```

```
=== CountVectorizer with BernoulliNB ===
Confusion Matrix:
 [[1289  576]
 [ 285 2500]]
```

```
=== TF-IDF with MultinomialNB ===
Confusion Matrix:
 [[1257  608]
 [ 213 2572]]
```

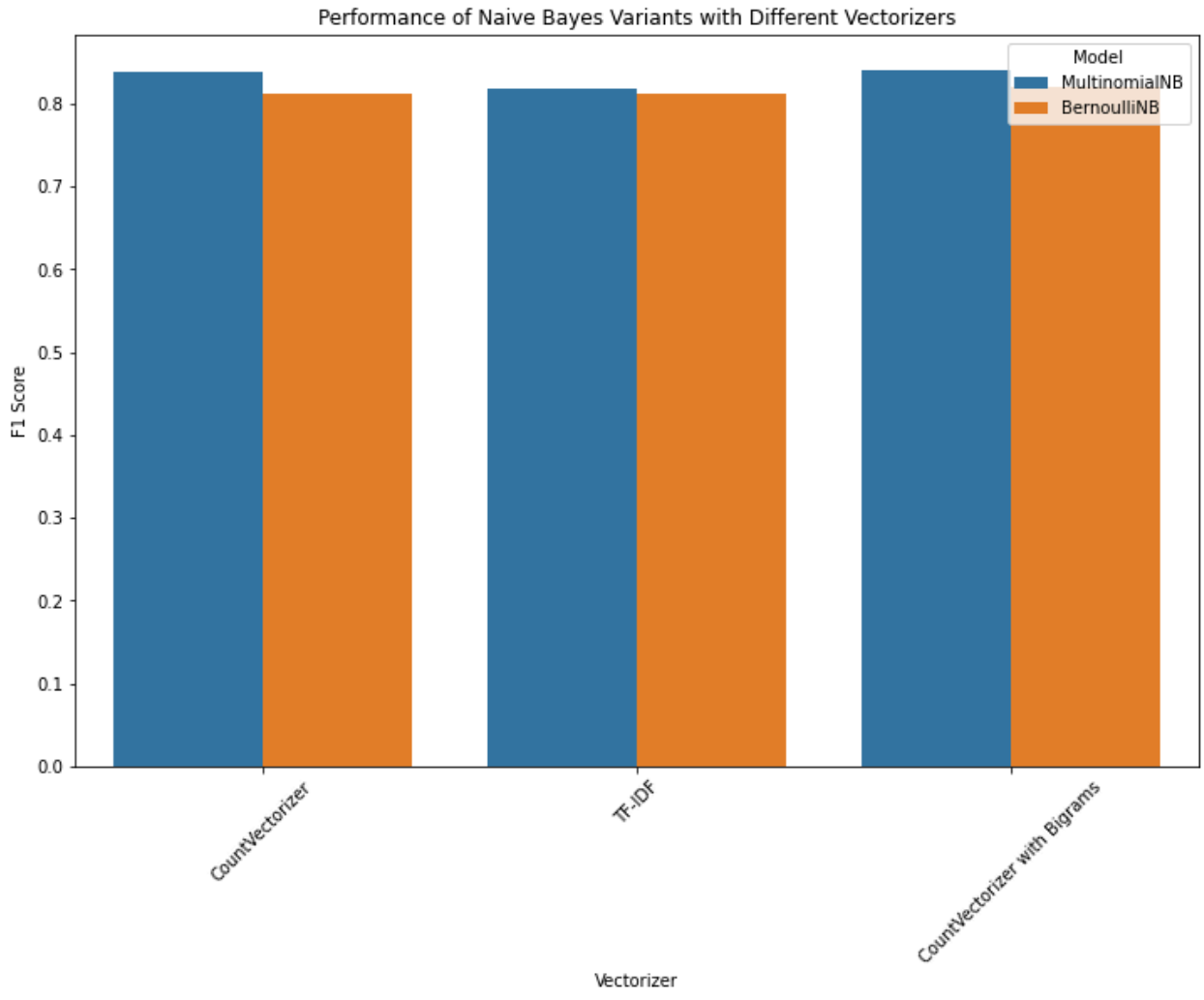
```
=== TF-IDF with BernoulliNB ===
Confusion Matrix:
 [[1289  576]
 [ 285 2500]]
```

```
=== CountVectorizer with Bigrams with MultinomialNB ===
Confusion Matrix:
 [[1446  419]
 [ 316 2469]]
```

```
=== CountVectorizer with Bigrams with BernoulliNB ===
Confusion Matrix:
 [[1302  563]
 [ 259 2526]]
```

As can be seen from the above results, MultinomialNB consistently maintains better balance between true positives and true negatives, with fewer false negatives, especially when paired with bigrams.

For better visualization, I use a barplot comparing F1 scores across vectorizers and models as shown below:



Hyperparameter tuning:

Tuned the alpha parameter (Laplace smoothing) of MultinomialNB and both alpha and threshold for binarization for BernoulliNB to find the best-performing value using GridSearchCV.

I get the best parameters for Bernoulli as:

```
Best Parameters for BernoulliNB: {'alpha': 0.1, 'binarize': 0.0}
```

With the hyperparameter tuning, we get a slightly improved Confusion matrix for both true positives and true negatives:

```
Confusion Matrix (Tuned BernoulliNB):
 [[1328  537]
 [ 274 2511]]
```

Similarly, the F1 score also improves to 82% for BernoulliNB -

```
Classification Report (Tuned BernoulliNB):
              precision    recall  f1-score   support

           0       0.83      0.71      0.77      1865
           1       0.82      0.90      0.86      2785

    accuracy                           0.83      4650
   macro avg       0.83      0.81      0.81      4650
weighted avg       0.83      0.83      0.82      4650
```

The best parameter for Multinomial Naive Bayes gives an alpha of 10 -

```
Best Parameters for MultinomialNB: {'alpha': 10.0}
```

Using the tuned Multinomial NB model, we don't see any particular improvement while considering F1 scores or confusion matrix for true positives and true negatives as compared to what we had earlier -

```
Confusion Matrix (Tuned MultinomialNB):
 [[1418  447]
 [ 281 2504]]

Classification Report (Tuned MultinomialNB):
              precision    recall  f1-score   support

           0       0.83      0.76      0.80      1865
           1       0.85      0.90      0.87      2785

    accuracy                           0.84      4650
   macro avg       0.84      0.83      0.83      4650
weighted avg       0.84      0.84      0.84      4650

Tuned MultinomialNB F1 Score: 0.84
Tuned MultinomialNB Accuracy Score: 0.84
```