A2.71 Naive Bayes Without Weight Balancing

Naive Bayes WithOut Class Weight Balancing

```
[ ] from sklearn.naive_bayes import GaussianNB
    gnb = GaussianNB()
    nb = gnb.fit(X_train, y_train)

    from sklearn.metrics import classification_report
    print("Naive Bayes withOut Class Weight Balancing - Classification Report (Test Set Results):")
    print(classification_report(y_test,nb.predict(X_test)))


    from sklearn.metrics import classification_report
    print("Naive Bayes withOut Class Weight Balancing - Classification Report (Train Set Results):")
    print(classification_report(y_train,nb.predict(X_train)))


    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    confmat = confusion_matrix(y_test, nb.predict(X_test))

    fig, ax = plt.subplots(figsize=(8,8))
    g = sns.heatmap(confmat,annot=True,ax=ax, fmt='0.1f',cmap='Accent_r')
    g.set_yticklabels(g.get_yticklabels(), rotation = 0, fontsize = 12)
    g.set_xticklabels(g.get_xticklabels(), rotation = 90, fontsize = 12)
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
```

1. Naive Bayes Model Training:

```python
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
nb = gnb.fit(X_train, y_train)
```

- Import the `GaussianNB` class from `sklearn.naive_bayes`. This is a Naive Bayes classifier for Gaussian-distributed data.
- Create an instance of the GaussianNB classifier, which is assigned to the variable `gnb`.
- Fit (train) this model on the training data (`X_train` and `y_train`). The trained model is then assigned to the variable `nb`.

2. Evaluation on Test Set:

```python
from sklearn.metrics import classification_report
print(classification_report(y_test,nb.predict(X_test)))
```

- Import the `classification_report` function from `sklearn.metrics`.
- Evaluate the trained Naive Bayes model (`nb`) on the test data (`X_test`).
- Print out a detailed classification report which provides precision, recall, f1-score, and support for each class.

3. Evaluation on Training Set:

```python
from sklearn.metrics import classification_report
print(classification_report(y_train,nb.predict(X_train)))
```

This section is similar to the previous one but evaluates the model on the training data instead of the test data. This helps in understanding how well the model has learned from the training data and can be a way to check for overfitting.

4. Plotting the Confusion Matrix:

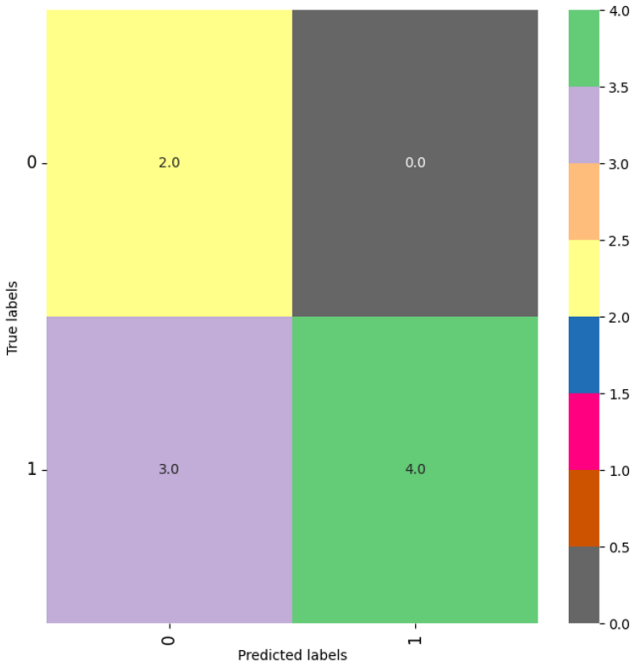
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

confmat = confusion_matrix(y_test, nb.predict(X_test))

fig, ax = plt.subplots(figsize=(8,8))
g = sns.heatmap(confmat,annot=True,ax=ax, fmt='0.1f',cmap='Accent_r')
```

	precision	recall	f1-score	support
0	0.40	1.00	0.57	2
1	1.00	0.57	0.73	7
accuracy			0.67	9
macro avg	0.70	0.79	0.65	9
weighted avg	0.87	0.67	0.69	9

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4
1	1.00	1.00	1.00	16
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20



The provided results detail the performance of the Gaussian Naive Bayes classifier on both the test and training sets. I'll break down the results for better clarity:

Test Set Results:

Class 0:
- **Precision**: Out of all instances predicted as class 0, 40% were actually class 0.
- **Recall**: Out of all actual instances of class 0, the model identified (or recalled) 100% of them correctly.
- **F1-Score**: The harmonic mean of precision and recall, which provides a balance between the two, is 57%.
- **Support**: There were 2 instances of class 0 in the test set.

Class 1:
- **Precision**: Out of all instances predicted as class 1, 100% were actually class 1.
- **Recall**: Out of all actual instances of class 1, the model identified 57% of them correctly.
- **F1-Score**: The harmonic mean of precision and recall is 73%.
- **Support**: There were 7 instances of class 1 in the test set.

Overall:
- **Accuracy**: Overall, 67% of all predictions on the test set were correct.
- **Macro avg**: This averages the unweighted mean per label. It's essentially taking the average of the metric for class 0 and the metric for class 1. For this data, it is 0.70 for precision, 0.79 for recall, and 0.65 for F1-score.
- **Weighted avg**: This provides an average metric but gives a weight to each metric based on the number of true instances for each class (support). In this data, it's 0.87 for precision, 0.67 for recall, and 0.69 for F1-score.

Training Set Results:

For the training set, the model seems to have achieved perfect performance.

Class 0 & Class 1:
- **Precision**: 100% for both classes, which means every prediction the model made for each class was correct.
- **Recall**: 100% for both classes, indicating that the model identified all instances of each class correctly.
- **F1-Score**: 100% for both classes, as both precision and recall are perfect.
- **Support**: There were 4 instances of class 0 and 16 instances of class 1 in the training set.

Output Interpretation:
1. Accuracy: 100%, meaning every single prediction made on the training set was correct.
2. Macro avg & Weighted avg**: Both averages show a perfect score of 1.00 across precision, recall, and F1-score because the model perfectly classified all instances.

Analysis Summary for both Training and Test data:
The Naive Bayes model has achieved perfect accuracy on the training set, but its performance on the test set, especially for class 1, was not so good. While it perfectly predicts class 0 in the test set, it only correctly identifies 57% of class 1 instances. This suggests that the model might be overfitting to the training data, as evidenced by the perfect scores on the training set and the reduced performance on the test set.