A2.72 Naive Bayes With Weight Balancing

```
# Applyig Weight Balancing to address the class imbalance in the dataset

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Assuming you have already preprocessed the data and split it into X_train, X_test, y_train, and y_test.

# 1. Class Weight Balancing:
# Calculate the class weights to balance the classes
class_weight = {0: len(y_train) / (2 * sum(y_train == 0)), 1: len(y_train) / (2 * sum(y_train == 1))}

# 2. Create the Gaussian Naive Bayes model without setting class_prior
gnb = GaussianNB()

# 3. Train the Gaussian Naive Bayes model on the training set
nb = gnb.fit(X_train, y_train, sample_weight=[class_weight[y] for y in y_train])

# 4. Evaluate the Gaussian Naive Bayes model on the test set
print("Naive Bayes with Class Weight Balancing - Classification Report (Test Set):")
print(classification_report(y_test, nb.predict(X_test)))

# 5. Evaluate the Gaussian Naive Bayes model on the training set
print("Naive Bayes with Class Weight Balancing - Classification Report (Training Set):")
print(classification_report(y_train, nb.predict(X_train)))

# 6. Plot the confusion matrix for the Gaussian Naive Bayes model with class weight balancing
confmat_weighted = confusion_matrix(y_test, nb.predict(X_test))
fig_weighted, ax_weighted = plt.subplots(figsize=(8, 8))
g_weighted = sns.heatmap(confmat_weighted, annot=True, ax=ax_weighted, fmt='0.1f', cmap='Accent_r')
g_weighted.set_yticklabels(g_weighted.get_yticklabels(), rotation=0, fontsize=12)
g_weighted.set_xticklabels(g_weighted.get_xticklabels(), rotation=90, fontsize=12)
ax_weighted.set_xlabel('Predicted labels')
ax_weighted.set_ylabel('True labels')
plt.title("Confusion Matrix for Naive Bayes with Class Weight Balancing")
plt.show()
```

The provided code demonstrates how to train and evaluate a Gaussian Naive Bayes classifier while addressing class imbalances using class weight balancing:

1. **Class Weight Balancing**:
   - It first calculates weights for each class, with the aim of giving more weight to the class that has fewer samples.
   - The formula for calculating the weights is: total number of samples divided by (2 times the number of samples of that class). The multiplication by 2 ensures that the sum of the two class weights remains equal to the total number of samples.

2. **Create the Gaussian Naive Bayes model**:
   - An instance of the `GaussianNB` classifier is created. Note that unlike some other classifiers, `GaussianNB` doesn't have a `class_weight` parameter. Instead, sample weights will be used during model training.

3. **Train the Gaussian Naive Bayes model**:
   - The classifier is trained using the `fit` method on the training data `X_train` and `y_train`.
   - The `sample_weight` parameter is provided during training. This parameter adjusts the weight of each training sample based on the class it belongs to, according to the previously calculated `class_weight`.

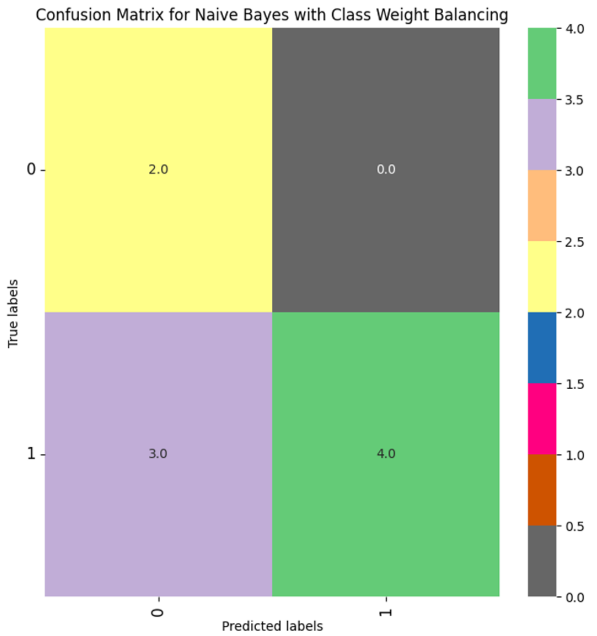
4. **Evaluate on the Test Set**:
   - The trained model is evaluated on the test set.
   - A classification report for the test set is printed, which includes metrics such as precision, recall, and F1-score.

5. **Evaluate on the Training Set**:
   - Similarly, a classification report for the training set is printed to observe the model's performance on the data it was trained on.

6. **Plotting the Confusion Matrix**:
   - A confusion matrix for the test set predictions is generated.
   - This matrix is visualised using the `seaborn` library to show how many samples of each class were correctly or incorrectly classified.
   - The heatmap provides a visual representation of the confusion matrix, with actual classes on the y-axis and predicted classes on the x-axis. The numbers inside the heatmap represent the count of samples for each combination of actual and predicted classes.

This code showcases how to apply class weight balancing to a Gaussian Naive Bayes classifier to handle class imbalance, followed by evaluating the model's performance on both the training and test sets and visualising the results using a confusion matrix.

```
Naive Bayes with Class Weight Balancing - Classification Report (Test Set):
              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

Naive Bayes with Class Weight Balancing - Classification Report (Training Set):
              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
```



Results for the Naive Bayes model with class weight balancing:

Test Set:

1. **For class 0**:
   - **Precision:** Out of all the instances predicted as class 0, 40% of them are actually of class 0. This means that there might be some instances predicted as class 0 but are actually of another class.
   - **Recall:** Out of all the actual instances of class 0, the model correctly identified 100% of them. So, the two instances of class 0 in the test set were both correctly identified.
   - **F1-Score:** This is the harmonic mean of precision and recall and is 0.57 for class 0. The F1-score gives a single metric that balances both precision and recall. Considering that the recall is high and precision is comparatively low, the F1-score is somewhat in between.

2. **For class 1**:
   - **Precision:** Out of all the instances predicted as class 1, 100% of them are actually of class 1. This means the model's predictions for class 1 are entirely accurate.
   - **Recall:** Out of the seven actual instances of class 1, the model correctly identified only 57% of them. So, the remaining 43% of instances of class 1 were incorrectly predicted as another class.
   - **F1-Score:** With a precision of 1.00 and recall of 0.57, the F1-score is 0.73 for class 1.

3. **Accuracy:** Overall, 67% of all the predictions on the test set are correct.

4. **Macro Avg:** This is the average of the unweighted mean per label. For precision, recall, and F1-score, the macro average is 0.70, 0.79, and 0.65 respectively.

5. **Weighted Avg:** This is the average of the support-weighted mean per label. For precision, recall, and F1-score, the weighted average is 0.87, 0.67, and 0.69 respectively.

Training Set:

For the training set, the results indicate perfect performance:

1. For both classes (0 and 1), the precision, recall, and F1-score are all 1.00. This means that every single instance was correctly classified, leading to an overall accuracy of 100%.

2. The macro average and weighted average metrics for precision, recall, and F1-score are all 1.00, which further confirms the model's perfect performance on the training data.