A2.8 KNN Without Weight Balancing

KNeighborsClassifier

The KNeighborsClassifier, is a classification algorithm nested within the scikit-learn library in Python. This classifier operates on the underlying principles of the K-Nearest Neighbors (KNN) algorithm. At its core, KNN is a non-parametric and instance-based supervised learning algorithm. Rather than relying on an explicit training phase, it makes predictions based on the similarity of new data points to those already present in the training dataset. When tasked with predicting the class of a new data point, KNN considers the 'K' nearest data points from the training set. For instance, in our code, we've initialised the KNeighborsClassifier to consider three nearest neighbours, denoted by 'n_neighbors=3'. From these selected neighbours, the algorithm identifies the majority class and assigns this class to the new data point. The effectiveness of this process heavily hinges on the choice of the hyperparameter 'K' and the distance metric used, often the Euclidean distance, to gauge the similarity between data points. Notably, an inappropriate choice of 'K' could make the model sensitive to noise or overly biased. Additionally, while KNN can be potent in various scenarios, its performance may falter on high-dimensional or imbalanced datasets if one doesn't undertake suitable preprocessing or hyperparameter tuning.

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(X_train, y_train)

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


    from sklearn.metrics import classification_report
    print("KNN withOut Class Weight Balancing - Classification Report (Test Set Results):")
    print(classification_report(y_train,knn.predict(X_train)))


    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    confmat = confusion_matrix(y_test, knn.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');
```

This code implements the K-Nearest Neighbors (KNN) algorithm and evaluates its performance on a dataset:

1. KNN Model Creation:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```
- The code begins by importing the `KNeighborsClassifier` from `sklearn.neighbors`.
- An instance of the KNN classifier is then created with the parameter `n_neighbors=3`, meaning it will consider the 3 nearest data points (or neighbors) when making predictions.
- The classifier is trained using the `fit` method on the training data (`X_train` and `y_train`).

2. Evaluating the Model on Test Data:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, knn.predict(X_test)))
```
- The `classification_report` is imported from `sklearn.metrics`. It provides a detailed breakdown of the model's performance in terms of precision, recall, F1-score, and support for each class.
- The model's predictions on the test data are compared with the true labels (`y_test`) to generate the report, which is then printed.

3. Evaluating the Model on Training Data:
```python
from sklearn.metrics import classification_report
print(classification_report(y_train, knn.predict(X_train)))
```
- Using the same `classification_report`, the model's performance is evaluated on the training data. This is useful to see how well the model fits the data it was trained on and can be an indicator of potential overfitting if the performance is too good.

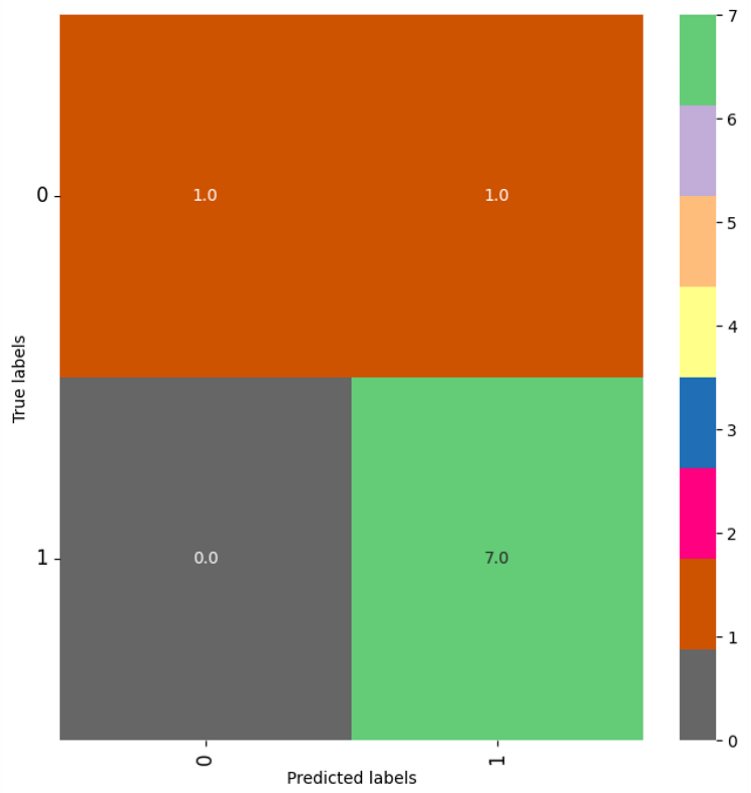
4. Plotting the Confusion Matrix for Test Data:
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

confmat = confusion_matrix(y_test, knn.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)
```

	precision	recall	f1-score	support
0	1.00	0.50	0.67	2
1	0.88	1.00	0.93	7
accuracy			0.89	9
macro avg	0.94	0.75	0.80	9
weighted avg	0.90	0.89	0.87	9

	precision	recall	f1-score	support
0	1.00	0.50	0.67	4
1	0.89	1.00	0.94	16
accuracy			0.90	20
macro avg	0.94	0.75	0.80	20
weighted avg	0.91	0.90	0.89	20



Below is the analysis of the performance of the KNN model:

Test Data Results:

1. **Class 0**:
   - **Precision**: 1.00 - Out of all the instances predicted as class 0, all were correct.
   - **Recall**: 0.50 - Out of all the actual class 0 instances, only half of them were correctly predicted.
   - **F1-score**: 0.67 - The harmonic mean of precision and recall. Given the precision of 1.00 and recall of 0.50, the F1-score is 0.67.

2. **Class 1**:
   - **Precision**: 0.88 - 88% of instances predicted as class 1 were correct.
   - **Recall**: 1.00 - All instances of actual class 1 were correctly predicted.
   - **F1-score**: 0.93 - Given the high precision and recall, the F1-score is also high.

3. **Accuracy**: 0.89 - 89% of the total predictions on the test set were correct.

4. **Macro Avg**: This averages the unweighted mean per label.
   - **Precision**: 0.94
   - **Recall**: 0.75
   - **F1-score**: 0.80

5. **Weighted Avg**: This averages the support-weighted mean per label.
   - **Precision**: 0.90
   - **Recall**: 0.89
   - **F1-score**: 0.87

Training Data Results:

1. **Class 0**:
   - **Precision**: 1.00
   - **Recall**: 0.50
   - **F1-score**: 0.67

2. **Class 1**:
   - **Precision**: 0.89
   - **Recall**: 1.00
   - **F1-score**: 0.94

3. **Accuracy**: 0.90 - 90% of the total predictions on the training set were correct.

4. **Macro Avg**:
   - **Precision**: 0.94
   - **Recall**: 0.75
   - **F1-score**: 0.80

5. **Weighted Avg**:
   - **Precision**: 0.91
   - **Recall**: 0.90
   - **F1-score**: 0.89

Output Interpretation of both training and test data:
1. The KNN model performs relatively well on both the test and training datasets with a high overall accuracy.
2. While it perfectly identifies instances of class 1 (high recall for class 1), it seems to struggle more with class 0, particularly in terms of recall. This means some of the true class 0 instances are being misclassified as class 1.
3. The F1-score gives a balanced perspective on performance considering both precision and recall. An F1-score close to 1 indicates better performance, while a score close to 0 indicates poor performance. The F1-scores for class 0 are lower than those for class 1, reinforcing the point that the model struggles more with class 0.
4. It's also important to notice that the results on the training and test data are relatively close, indicating that the model is likely not overfitting.