A2.31 Support Vector Machine WithOut Class Weight Balancing

SVM WithOut Class Weight Balancing

```
[ ] # With out Applying weight balancing
    import warnings
    warnings.filterwarnings('ignore')
    from sklearn.svm import SVC

    SVm = SVC(gamma='auto')
    SVm.fit(X_train, y_train)


    from sklearn.metrics import classification_report
    # print(classification_report(y_test,SVm.predict(X_test)))

    from sklearn.metrics import classification_report

    from sklearn.metrics import confusion_matrix
    import matplotlib.pyplot as plt
    import seaborn as sns

    print("SVM WithOUT Weight Balancing - Classification Report (Test Set Results):")
    print(classification_report(y_test, SVm.predict(X_test)))

    # 5. Evaluate the Random Forest model on the training set
    print("SVM WithOUT Weight Balancing - Classification Report (Training Set Results):")
    print(classification_report(y_train, SVm.predict(X_train)))

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

The code above is used to train a Support Vector Machine (SVM) classifier on a dataset and assess its performance using various metrics. This code trains an SVM classifier without balancing the weights of the classes, evaluates its performance using a classification report, and visualises its performance using a heatmap of the confusion matrix.

1. `# Without Applying weight balancing`:
   - The model is trained without taking into account any class imbalance in the data. In many real-world datasets, certain classes can be underrepresented, and special techniques are applied to balance the weights of classes. Here, such techniques haven't been applied at this stage.

2. `from sklearn.svm import SVC`:
   - Import the Support Vector Classification model from `sklearn`'s SVM module.

3. `SVm = SVC(gamma='auto')`:
   - Initialize an SVM classifier. The parameter `gamma='auto'` sets the gamma parameter for the kernel.
When it's set to 'auto', it uses the formula `1 / n_features` for the value of gamma.

4. `SVm.fit(X_train, y_train)`:
   - Train the SVM classifier using the training data `X_train` and their corresponding labels `y_train`.

5. `from sklearn.metrics import classification_report`:
   - Import the `classification_report` function to assess the performance of the classifier.

6. `print(classification_report(y_test,SVm.predict(X_test)))`:
   - This line produces a detailed classification report, which includes metrics like precision, recall, f1-score for each class. It's useful to assess how well the classifier is doing for each class, especially when there's class imbalance. When X_test is replaced by X_train this does the same for the training data. These then generate the reports shown below in the matrix table.

7. `from sklearn.metrics import confusion_matrix`:
   - Import the `confusion_matrix` function, which computes the confusion matrix—a table used to evaluate the accuracy of a classification.

Visualisation of the confusion matrix:

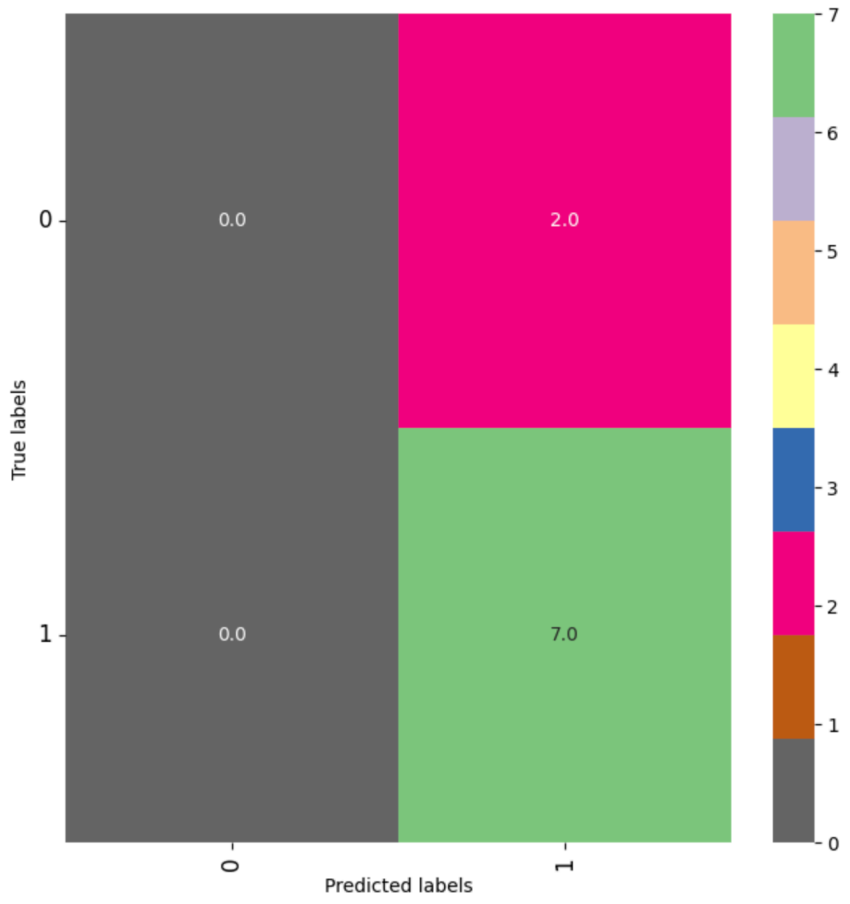
- `import matplotlib.pyplot as plt` and `import seaborn as sns`: Import the necessary libraries for visualization.
- `confmat = confusion_matrix(y_test, SVm.predict(X_test))`: Calculate the confusion matrix.
- `fig, ax = plt.subplots(figsize=(8,8))`: Create a new figure and axis for plotting.
- `g = sns.heatmap(...)`: Draw a heatmap of the confusion matrix. The heatmap will show the number of samples that belong to a given true class and were predicted as another class.
- `g.set_yticklabels(...)` and `g.set_xticklabels(...)`: Adjust the labels for the y and x axis.
- `ax.set_xlabel('Predicted labels')` and `ax.set_ylabel('True labels')`: Set the labels for the x and y axes.

Class 0 is for non-survivors and Class 1 is for Survivors

```
SVM WithOUT Weight Balancing - Classification Report (Test Set Results):
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.78      1.00      0.88         7

    accuracy                           0.78         9
   macro avg       0.39      0.50      0.44         9
weighted avg       0.60      0.78      0.68         9

SVM WithOUT Weight Balancing - Classification Report (Training Set Results):
              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 output result of a classification report for a binary classification problem using a Support Vector Machine (SVM) classifier:

Columns:
- **Precision**: This measures the proportion of positive identifications that were actually correct. It's calculated as \(\frac{True Positives}{True Positives + False Positives} \).
- **Recall**: This measures the proportion of actual positives that were correctly identified. It's calculated as \(\frac{True Positives}{True Positives + False Negatives} \).
- **f1-score**: The F1 score is the harmonic mean of precision and recall, and it ranges between 0 and 1. It provides a balance between the two. The formula is \(2 \times \frac{Precision \times Recall}{Precision + Recall} \).
- **Support**: The number of actual occurrences of the class in the specified dataset.



Rows:
1. **Class 0**:
   - **Precision**: 0.00 indicates that none of the samples predicted as class 0 were actually class 0.
   - **Recall**: 0.00 indicates that none of the actual class 0 samples were correctly predicted.
   - **f1-score**: 0.00 is expected, given the precision and recall are both zero.
   - **Support**: 2 means there are two samples of class 0 in the test dataset.
2. **Class 1**:
   - **Precision**: 0.78 suggests that 78% of samples predicted as class 1 were truly class 1.
   - **Recall**: 1.00 suggests that all actual class 1 samples were correctly predicted.
   - **f1-score**: 0.88 is the harmonic mean of the precision and recall for class 1.
   - **Support**: 7 means there are seven samples of class 1 in the test dataset.
3. **Accuracy**:
   - This is the ratio of correctly predicted samples to the total samples. An accuracy of 0.78 means that 78% of all predictions were correct.
4. **Macro avg**:
   - **Precision**: The average precision across all classes, treating each class equally, is 0.39.
   - **Recall**: The average recall across all classes is 0.50.
   - **f1-score**: The average F1 score across all classes is 0.44.
   - **Support**: 9 indicates the total number of samples.
5. **Weighted avg**:
   - **Precision**: The average precision, weighted by the number of true instances for each label, is 0.60.
   - **Recall**: The average recall, weighted by the number of true instances for each label, is 0.78.
   - **f1-score**: The average F1 score, weighted by the number of true instances for each label, is 0.68.
   - **Support**: Again, the total number of samples is 9.

Output Interpretation:
1. From this output, we can infer that the classifier has a harder time predicting Class 0 correctly since both its precision and recall are zero.
2. The classifier does relatively well with Class 1.
3. The macro average provides an unweighted metric, which might be more informative in situations with imbalanced classes. The weighted average gives us an idea of the overall performance when accounting for the class distribution in the dataset.