Data Analytics Assignment 4

- Team 31

How Class Imbalance Affects Multiclass Classification:

1. **Bias Toward Majority Classes**: Most classifiers optimise for overall accuracy. In imbalanced datasets, this leads to the classifier focusing on the majority class and ignoring minority classes. As a result, the model may have high accuracy but poor performance in predicting minority classes.
2. **Poor Generalisation for Minority Classes**: The classifier may not learn enough about the minority classes, resulting in higher misclassification rates for those classes. This can affect metrics like recall and F1-score, especially for the minority classes.
3. **Skewed Evaluation Metrics**: With imbalanced datasets, accuracy alone becomes a misleading metric. The model may show high accuracy just by correctly predicting the majority class, but it may fail to identify the minority classes effectively.
4. **Overfitting on Small Datasets**: In small datasets with class imbalance, the risk of overfitting increases. The classifier might memorise patterns from the majority class, further ignoring the minority class.

Strategies to Mitigate the Impact of Class Imbalance:

1. **Undersampling the Majority Classes** to balance the dataset. This is useful when the dataset is large enough that losing data from the majority class won't harm the overall performance. However, it can lead to loss of important information.
2. Many classifiers, such as Support Vector Machines (SVM) and Decision Trees, allow setting **class weights**. By assigning higher weights to the minority classes, the classifier can penalise misclassifications for those classes more heavily, forcing the model to pay more attention to them.
3. **In Cost-Sensitive Learning**, different misclassification costs are assigned to different classes. This forces the model to minimise errors on the minority classes by introducing higher penalties for misclassifying them.
4. **Balanced Random Forest** or **EasyEnsemble** can be used to build a series of classifiers with balanced datasets. Each model in the ensemble is trained on a balanced subset of data, leading to improved performance on minority classes.
5. **Boosting Algorithms** like **AdaBoost** or **XGBoost** can also focus on hard-to-classify instances, often minority class samples.
6. Use **macro-averaged precision, recall, and F1-score**, which give equal weight to each class, ensuring that minority classes are equally represented in the evaluation.
7. When using small datasets, applying **stratified cross-validation** ensures that each fold of the dataset has a balanced distribution of classes, improving the reliability of performance estimates across minority and majority classes.

<u>Effect of Choice of Hyperparameters :</u>

Random Forest Classifier:

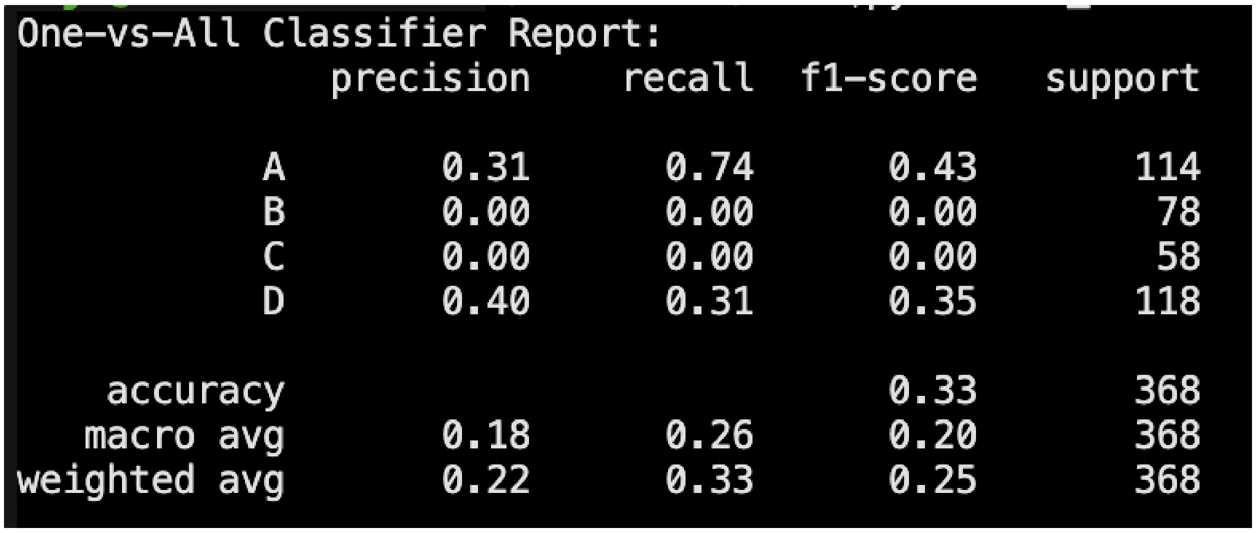
- **Number of Trees**:
  - **Too Few Trees**: If the number of trees is too small, the model will be underfit, as it won't have enough diverse learners to capture the complexity of the data.
  - **Too Many Trees**: With a large number of trees, the model is less likely to overfit due to the averaging effect, but computational complexity will increase significantly, leading to inefficiency.
- **Max Depth**:
  - **Too Shallow Depth**: A small maximum depth will restrict the depth of the individual decision trees, causing underfitting, as the model won't capture deeper patterns and interactions in the data.
  - **Too Deep**: Setting maximum depth too high will allow trees to grow too large, leading to overfitting. The trees will capture noise and irrelevant patterns, reducing generalisation ability.
- **Minimum Samples for Splitting and Leaf Nodes**:
  - **Too Large**: A large minimum number of samples for a split or leaf node can lead to underfitting, as it forces the model to create less specific, more generalised trees.
  - **Too Small**: Small values for these parameters allow the trees to grow more complex, making them more prone to overfitting, especially in noisy data.
- **Max Features**:
  - **Too Few Features**: Limiting the number of features considered at each split can result in underfitting because the model may overlook important relationships between variables.
  - **Too Many Features**: If too many features are used, especially in high-dimensional data, the model may overfit, as individual trees will resemble each other too closely, capturing noise in the data.

Support Vector Machine (SVM) Classifier:

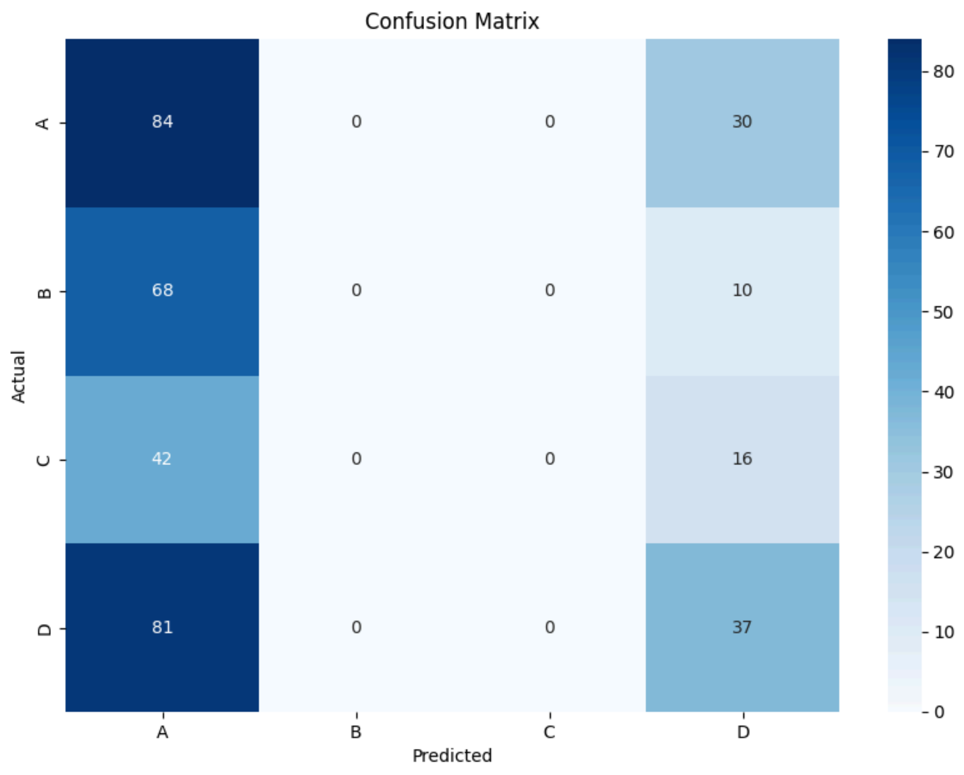
- **Regularization Parameter**:
  - **Too High**: A large value reduces the margin, allowing the SVM to prioritize correctly classifying all training points, which can lead to overfitting. The model becomes too sensitive to individual data points and noise.
  - **Too Low**: A small value encourages a larger margin and higher tolerance for misclassification, which can cause underfitting, as the model becomes too simplistic and doesn't capture the complexity of the data.

- **Kernel Choice**:
    - **Linear Kernel**: Works well when the data is linearly separable. However, using a linear kernel for non-linear data will lead to underfitting, as it cannot capture the non-linear relationships.
    - **Non-Linear Kernels (e.g., RBF, Polynomial)**: These can model complex data, but choosing overly complex kernels, like high-degree polynomials, can lead to overfitting by modelling noise as significant patterns.
- **Kernel Coefficient**:
    - **Too High**: A high gamma makes the model focus on individual points, creating highly complex decision boundaries, which leads to overfitting, especially in noisy data.
    - **Too Low**: A low gamma results in a smoother, less flexible decision boundary, which can cause underfitting, as the model fails to capture important variations in the data.
- **Overfitting**: Tends to occur with too deep trees (in Random Forest), high regularisation parameter (SVM), or high kernel coefficient, leading to models that are overly complex and sensitive to noise.
- **Underfitting**: Happens when the model is too simplistic, such as with shallow trees (Random Forest), low regularisation parameter, or overly smooth kernels with low kernel coefficient, failing to capture data complexity.

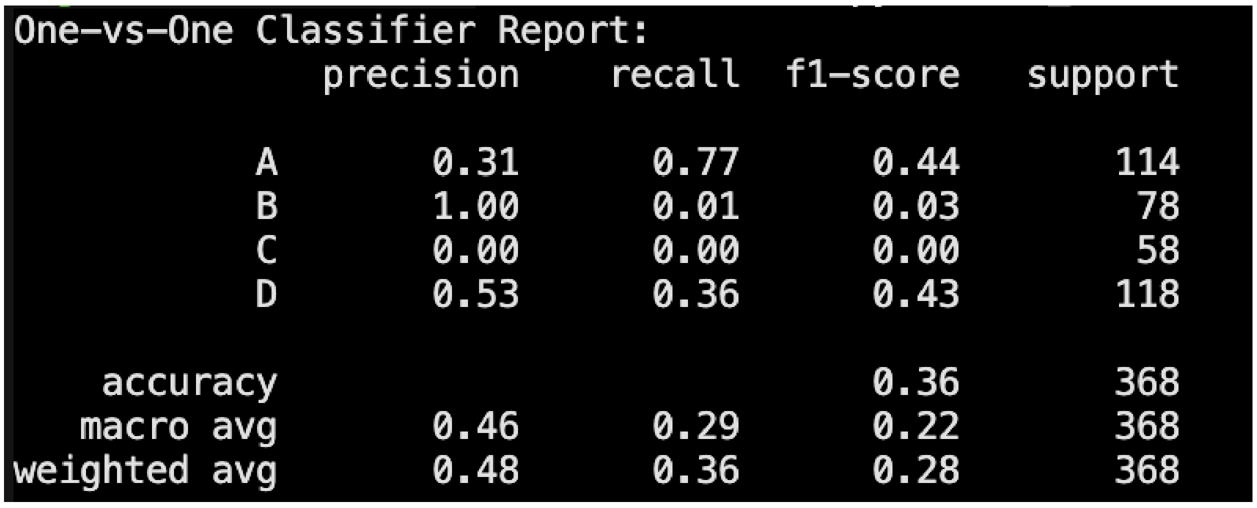
<u>One-vs-All Metrics and Confusion Matrix:</u>



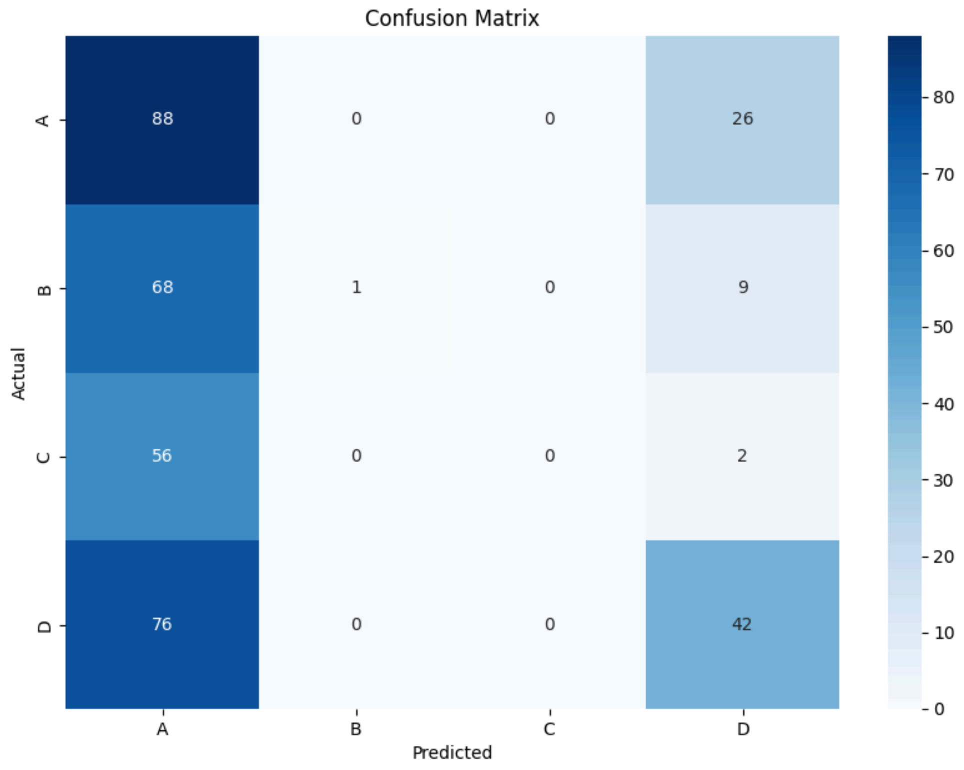
```
One-vs-All Classifier Report:
              precision    recall  f1-score   support

           A       0.31      0.74      0.43       114
           B       0.00      0.00      0.00        78
           C       0.00      0.00      0.00        58
           D       0.40      0.31      0.35       118

    accuracy                           0.33       368
   macro avg       0.18      0.26      0.20       368
weighted avg       0.22      0.33      0.25       368
```



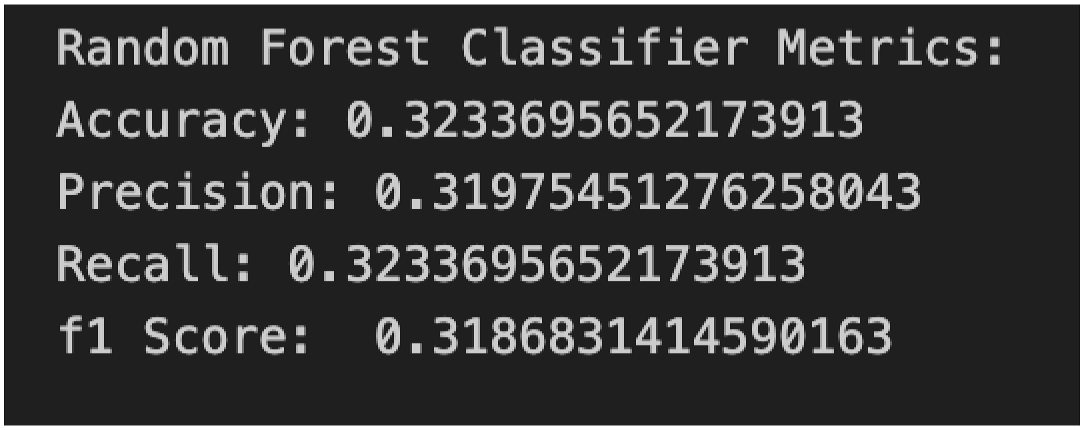
One-vs-One Metrics and Confusion Matrix:



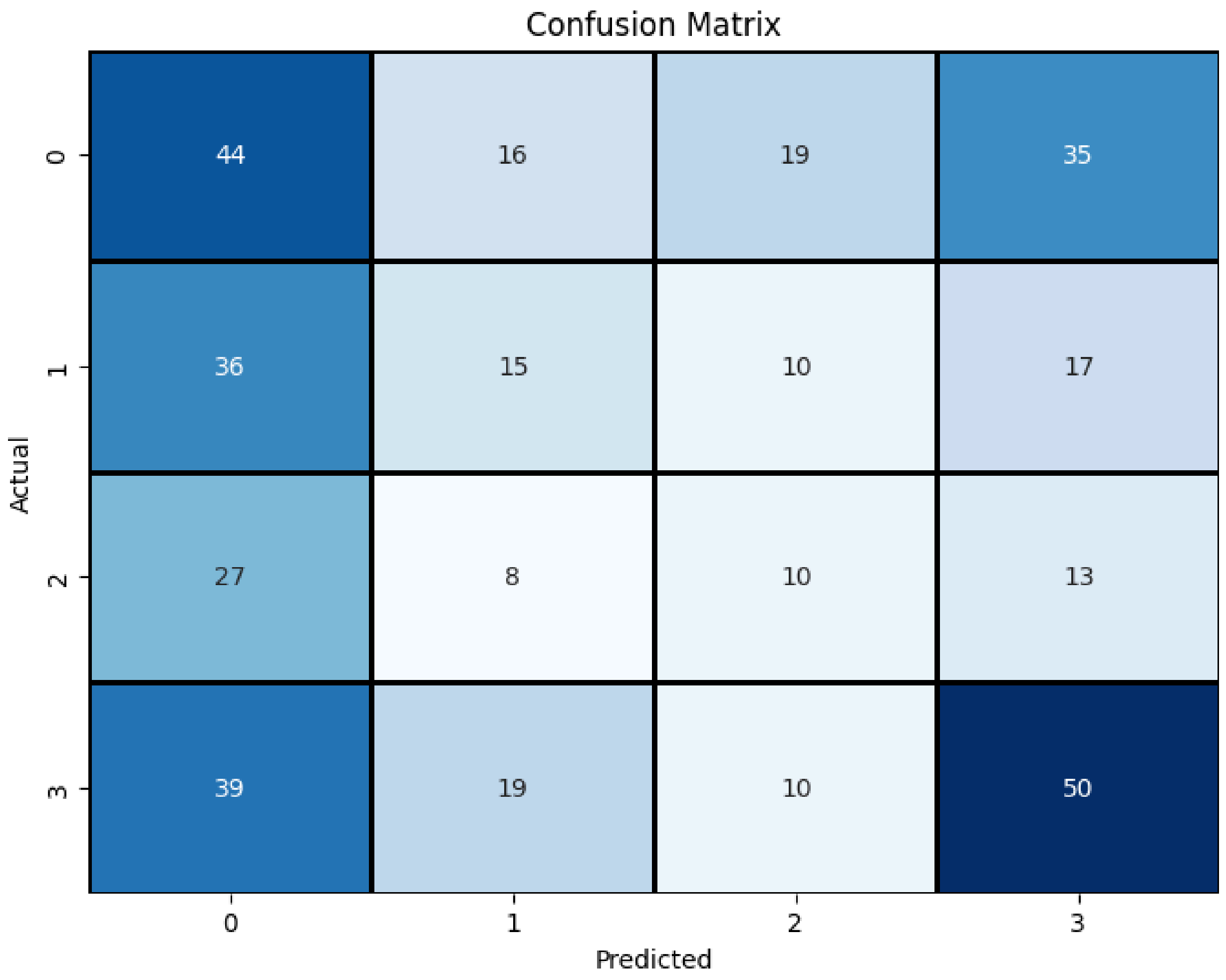
```
One-vs-One Classifier Report:
              precision    recall  f1-score   support

           A       0.31      0.77      0.44       114
           B       1.00      0.01      0.03        78
           C       0.00      0.00      0.00        58
           D       0.53      0.36      0.43       118

    accuracy                           0.36       368
   macro avg       0.46      0.29      0.22       368
weighted avg       0.48      0.36      0.28       368
```



Random Forest Classifier Metrics and Confusion Matrix:



```
Random Forest Classifier Metrics:
Accuracy: 0.3233695652173913
Precision: 0.3197545127625804З
Recall: 0.3233695652173913
f1 Score:  0.318683141459016З
```



Comparison of results of OvO and OvA Classifiers:

One-vs-One (OvO): The One-vs-One strategy constructs a separate binary classifier for every possible pair of classes. If you have n classes, this approach will generate $\frac{n(n+1)}{2}$ classifiers, making it computationally expensive as the number of classes increases. Each classifier is responsible for distinguishing between just two classes, meaning that a smaller amount of data is required per classifier. OvO tends to perform better when dealing with class imbalance between small class subsets, as each classifier only deals with two classes at a time, reducing the bias introduced by heavily imbalanced datasets. However, as the number of classifiers increases quadratically with the number of classes, the computational cost grows significantly, making this approach less practical for datasets with many classes. Additionally, once all classifiers are trained, a voting system is usually used to determine the final prediction, which may introduce some complexity in decision-making.

One-vs-All (OvA): In contrast, the One-vs-All approach constructs a single binary classifier for each class, treating that class as the positive class and all other classes combined as the negative class. This means that for n classes, n classifiers are generated. OvA is simpler and more computationally efficient than OvO, as it requires fewer classifiers. However, one drawback is that the model can struggle with imbalanced datasets where certain classes dominate others. Since each classifier treats multiple classes as negatives, the large number of negative samples might overpower the positive class, leading to suboptimal performance for underrepresented classes. This can be mitigated by using techniques like class weighting or balancing the data. Despite its limitations, OvA is often preferred when computational resources are limited or when working with a large number of classes, as it reduces the training time compared to OvO.

Recommendation: For the given dataset, which likely involves a smaller number of classes and some degree of class imbalance, the **One-vs-All (OvA)** strategy generally performs better. OvA is particularly effective when dealing with smaller datasets where class imbalance is more pronounced, as it requires fewer classifiers and is computationally efficient. However, careful attention should be given to class balancing techniques to avoid bias toward dominant classes. While One-vs-One might offer advantages in certain scenarios, its computational demands make it less suitable for this particular dataset. Thus, OvA would be the more practical and reliable choice in this case.