Assignment 4 Report

Team 51

Muskan Raina (2021101066)

Arghya Roy (2021115008)

Data Preprocessing

1. The `ID` column was dropped as it was not relevant for the model.
2. Handling Missing Values

- Training Data:
    - Rows with missing values in more than 2 columns were dropped.
    - Rows with missing values in the `Graduated` column were dropped.
    - Missing values in the `Work_Experience` column were imputed with 0 if the `Graduated` column was `No`.
    - Remaining rows with missing values were dropped.
- Test Data:
    - Missing values in `Work_Experience` were replaced with the median value.
    - Missing values in `Family_Size` were replaced with the median value.
    - Missing values in `Var_1` were replaced with the mode value.
    - Missing values in `Profession` were replaced with the mode value.
    - Missing values in `Ever_Married` were replaced with the mode value.
    - Missing values in `Graduated` were replaced with the mode value.

3. Encoding Categorical Features

- One-Hot Encoding:

  The features `Gender` and `Profession` were one-hot encoded, creating separate columns for each unique value in these columns.
- Label Encoding:

The features `Spending_Score`, `Graduated`, `Ever_Married`, and `Var_1` were label encoded. This maps each unique value in these columns to a numerical value.

  - `Spending_Score`: Low → 0, Average → 1, High → 2
  - `Graduated`: Yes → 1, No → 0
  - `Ever_Married`: Yes → 1, No → 0
  - `Var_1`: Cat_1 → 1, Cat_2 → 2, ..., Cat_7 → 7

- Target Variable:

  The target variable `Segmentation` was label encoded: A → 0, B → 1, C → 2, D → 3.

4. Scaling Numerical Features

   The features `Work_Experience` and `Family_Size` were scaled to a range between 0 and 1.

Task 3: Report and Analysis

a) Class Imbalance and Its Impact on Multiclass Classification

Class imbalance occurs when some classes are underrepresented compared to others in a dataset. In multiclass classification, this can lead to a bias where the classifier favors the majority classes, resulting in poor performance for minority classes.

The impact of class imbalance can lead to inaccurate predictions where the classifier may be biased towards the majority class, making it harder to correctly classify minority classes, or skewed metrics where accuracy may appear high simply because the classifier concentrates on one class the majority class rather than how well all other classes perform.

Strategies to Mitigate Class Imbalance:

1. Resampling the Dataset:

Oversampling the minority class or undersampling the majority class can help balance the dataset.

2. Class Weighting:

   In algorithms like SVM and Random Forest, you can use the `class_weight` parameter to assign higher penalties for misclassifying minority classes. This forces the model to pay more attention to the underrepresented classes.

3. Ensemble Methods:

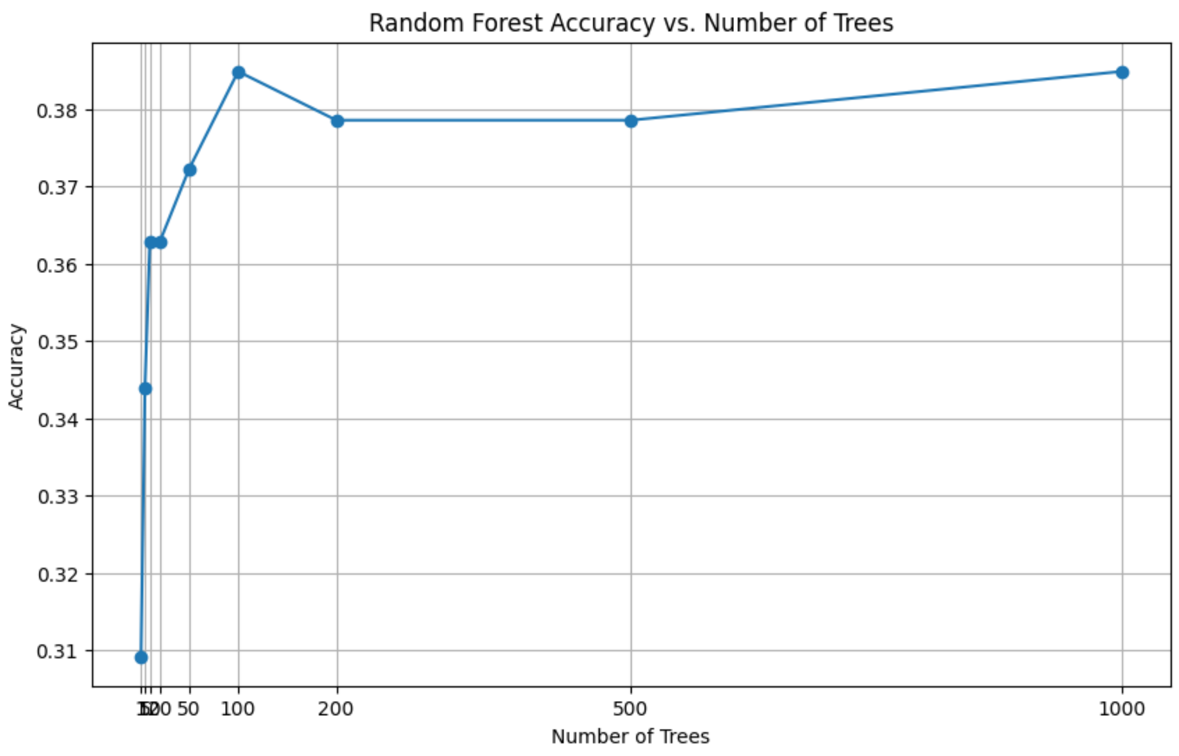
   Methods like Boosting (e.g., AdaBoost) can help by focusing more on misclassified examples, which often belong to minority classes.

b) Hyperparameters and Overfitting/Underfitting

The choice of hyperparameters is important in determining whether a model is more prone to **overfitting** (captures too much noise from the training data and performs poorly on unseen data) or **underfitting** (too simple and fails to capture the underlying patterns in the data).

Random Forest Hyperparameters:

1. Number of Trees:
   - Too low may lead to underfitting since the model won't have enough diversity to capture complex patterns.
   - Too high reduces the risk of overfitting but may increase computation time unnecessarily.
   - Our Random Forest model shows rapid improvement in accuracy up to 100 trees (reaching about 0.385), followed by a slight decrease and then a very gradual improvement up to 1000 trees. The minimal improvement in accuracy beyond 100 trees, despite a 10x increase in the number of trees, suggests that using around 100 trees would be the most computationally efficient choice while maintaining good performance.



2. Maximum Depth:
   - Low values can lead to underfitting as each tree in the forest will be shallow and not capture enough data complexity.
   - High values can cause overfitting since deeper trees may capture noise in the data, leading to poor generalization.
3. Max Features:
   - Low values help prevent overfitting by limiting how much of the data each tree sees.
   - High values may cause overfitting as trees become more correlated, reducing the randomization benefits.

SVM Hyperparameters:

1. Regularization Parameter **(** `C` **)**:
   - Low `C` promotes a smoother decision boundary, which may lead to underfitting (ignores some data points).
   - High `C` tries to classify all training points correctly, potentially leading to overfitting.

2. Kernel Choice:
    - Linear kernel: Works well if data is linearly separable; otherwise, underfitting may occur.
    - Non-linear kernels (e.g., RBF): Can capture more complex patterns but may lead to overfitting if not regularized properly.

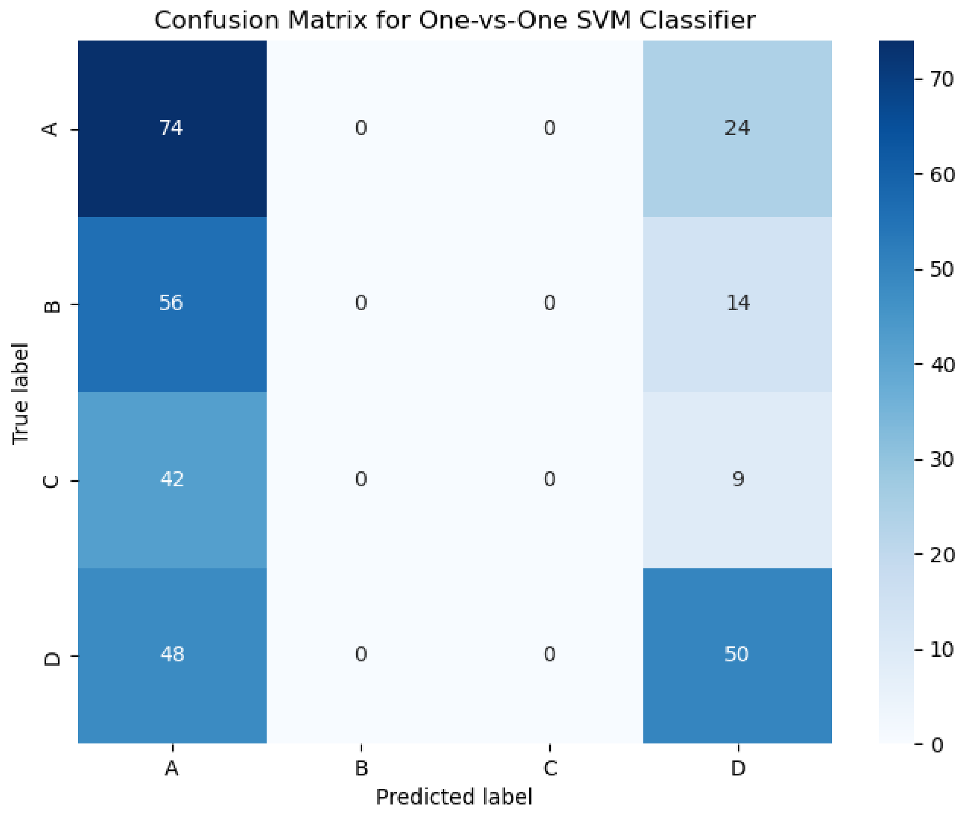
c) Confusion Matrix, Precision, Recall, and F1-Score

1. OVO Classifier

```
Final One-vs-One Classifier Accuracy: 0.39
Confusion Matrix
[[74  0  0 24]
 [56  0  0 14]
 [42  0  0  9]
 [48  0  0 50]]
Classification Report
              precision    recall  f1-score   support

           0       0.34      0.76      0.47        98
           1       0.00      0.00      0.00        70
           2       0.00      0.00      0.00        51
           3       0.52      0.51      0.51        98

    accuracy                           0.39       317
   macro avg       0.21      0.32      0.24       317
weighted avg       0.26      0.39      0.30       317
```



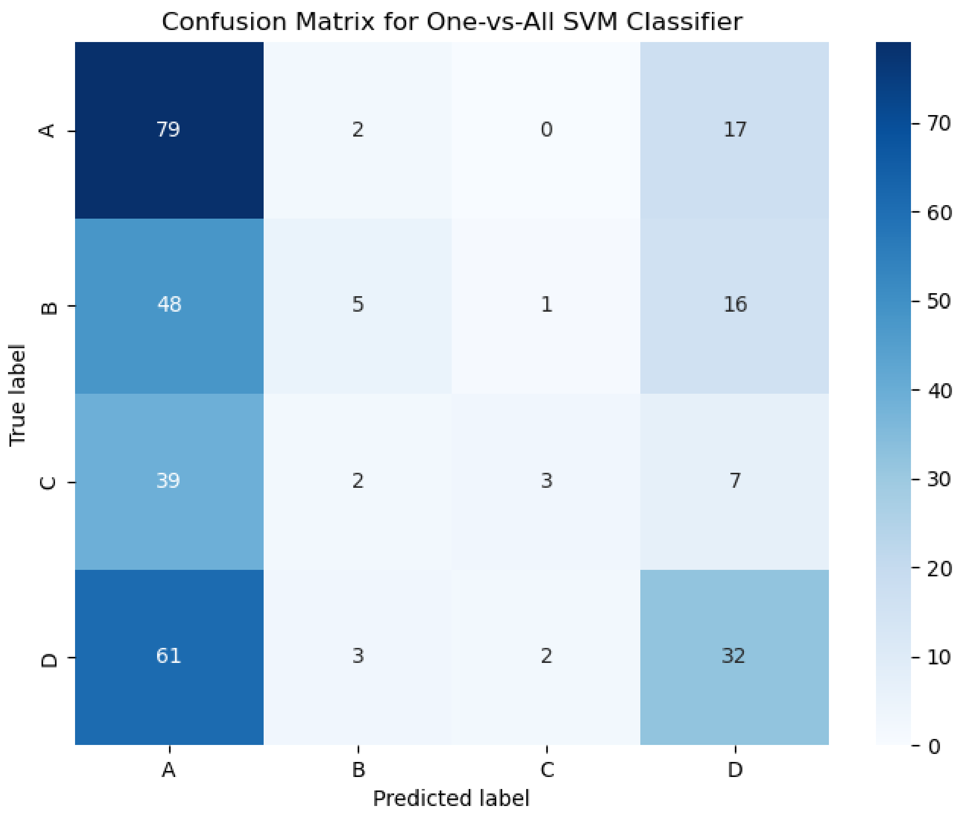
---

2. OVA Classifier

```
Final One-vs-All Classifier Accuracy: 0.38
Confusion Matrix
[[79  2  0 17]
 [48  5  1 16]
 [39  2  3  7]
 [61  3  2 32]]
Classification Report
              precision    recall  f1-score   support

           0       0.35      0.81      0.49        98
           1       0.42      0.07      0.12        70
           2       0.50      0.06      0.11        51
           3       0.44      0.33      0.38        98
```

```
    accuracy                           0.38       317
   macro avg       0.43      0.32      0.27       317
weighted avg       0.42      0.38      0.31       317
```



d) Comparing One-vs-One and One-vs-All

- **One-vs-One (OVO)** tends to perform better when there are many classes, as each classifier only needs to distinguish between two classes at a time. However, it can become computationally expensive for large datasets with many classes.
- **One-vs-All (OVA)** is computationally simpler, but each classifier must distinguish one class from all others, which can be more challenging in cases of overlapping classes.

Results based on Dataset:

- **Accuracy**: The One-vs-One classifier (0.39) slightly outperforms the One-vs-All classifier (0.38).
- **Class Performance**: One-vs-One achieves a much higher recall for Class 1 and 2, while One-vs-All distributes the recall more evenly across classes.
- **F1-Score**: The One-vs-All approach provides better balance in F1-scores, whereas One-vs-One is more skewed, performing well for some classes but failing for others.