**(Konstantin Orlovskiy, student ID 13157188)**

**Deliverables from previous stages of the project.**

Preprocessing stage delivered two output datasets:

* 1. Robust Scaler preprocessed
  2. Standard Scaler preprocessed

Algorithmic feature selection stage output:

* 1. Decision Tree (10 features)
  2. Principal Component Analysis (5 features)
  3. Recursive Feature Elimination Logistic Classifier (5 features)
  4. Recursive Feature Elimination Logistic Classifier (10 features)

The algorithms selected for refining are:

* 1. Logistic Regression Classifier (LR)
  2. Decision Tree Classifier (DT)
  3. Random Forest Classifier (RF)
  4. Support Vector Classifier (SVC)
  5. K-Nearest Neighbour (KNN)
  6. Naive Bayes Classifier (NBC)

**Refining algorithms**

Considering the number of possible combinations of outputs from preprocessing, feature and algorithm selection stages the following approach was chosen for refining the algorithms:

* Build a skeleton for tuning all algorithms on one combination of preprocessed data and selected features.
* Use the skeleton to refine algorithms for all other combinations.

The intrusion detection is the classification problem therefore the classificatory models are used for predictions. For each algorithm the approach was the following:

* Default model: test model with default hyperparameters. The result accuracy and confusion matrix is used as a benchmark to evaluate the improvement after refining hyperparameters.
* Hyperparameter selection depending on the type of the algorythm based on the documentation [1], [2], [3], [4], [5], [6].
* Search for best estimator (a combination of hyperparameters) using randomized search cross-validation (RandimizedSearchCV) from sklearn library on the train set. The cross-validation approach leads to detecting the generalized model hyperparameters. RandimizedSearchCV was chosen instead of GridSearchCV because it is less computationally extensive and proved to provide relatively the same result [7]. Bayesian optimisation was not used due to the lack of time and resource considering the range of models to be tuned. For reproducibility of results set random\_state = 2019.
* Test refined model on the test set.
* Iteratively analyze the selected set of the hyperparameters and refine the arrays with the nearest values for each selected hyperparameter to achieve better accuracy with the next iteration. For models that are not improving run more extensive search of hyperparameters to achieve even slight improvement in accuracy.

Since different algorithms have different computational cost, initially LR/DT/RF have been run with default n\_iter=10 which means the RandimizedSearchCV is trying 10 combinations of hyperparameters randomly, while for KNN/NBC/SVC are computationally expensive and n\_iter=3 was chosen but when there was significant improvement from default model, the value of n\_iter was increased. In some cases n\_iter value was chosen to try all possible combinations which is technically grid search. This approach was used for only those models where there was a potential for improvement.

Results of refining each model were added to summary table which contains model accuracy, confusion matrix as well as calculated based on the confusion matrix attributes like specificity, sensitivity, precision, etc. which are required to evaluate the model performance and compare to others and choose the best model at next stage.

The best performing algorithm appeared to be Random Forest with RFE log 10 feature selection from Robust Scaler preprocessed dataset. Hyperparameters tuned are below. The tradeoff of improvement of accuracy is the computational cost/speed of the algorithm.

* ’n\_estimators’ determines the number of trees in the forest, the higher it is the better the data is learnt;
* 'max\_depth': shows the depth of each tree, the higher it is the more information is considered;
* ‘min\_samples\_leaf’: limits the minimum number of samples needed to be in the leaf node, the higher the parameter is, the more under fitted is the model;
* 'max\_features': determines the number of features to consider when splitting the node which improves the model accuracy.

The refined best model specifications and performance:

* accuracy: 0.9761442302903531
* parameters: {'bootstrap': True, 'class\_weight': None, 'criterion': 'gini', 'max\_depth': 50, 'max\_features': 'log2', 'max\_leaf\_nodes': None, 'min\_impurity\_decrease': 0.0, 'min\_impurity\_split': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'min\_weight\_fraction\_leaf': 0.0, 'n\_estimators': 10, 'n\_jobs': None, 'oob\_score': False, 'random\_state': None, 'verbose': 0, 'warm\_start': False}
* Confusion matrix: [ [19124 955]

[ 3 20076] ]

Model tuning report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **precision** | **recall** | **f1-score** | **support** |
| **0.0** | 1.00 | 0.95 | 0.98 | 20079 |
| **1.0** | 0.95 | 1.00 | 0.98 | 20079 |
|  |  |  |  |  |
| **accuracy** |  |  | 0.98 | 40158 |
| **macro avg** | 0.98 | 0.98 | 0.98 | 40158 |
| **weighted avg** | 0.98 | 0.98 | 0.98 | 40158 |

**Literature**:

[1] Python Scikit Learn documentation. “LogisticRegression”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

[2] Python Scikit Learn documentation. “DecisionTreeClassifier”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

[3] Python Scikit Learn documentation. “RandomForestClassifier”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

[4] Python Scikit Learn documentation. “SVC”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

[5] Python Scikit Learn documentation. “KNeighborsClassifier”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

[6] Python Scikit Learn documentation. “GaussianNB”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

[7] Python Scikit Learn documentation. “Comparing randomized search and grid search for hyperparameter estimation”. Available: <https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html>