Phase 1 Machine learning Report

Team no. : 1

Section : 1

Team Members: Ahmed Mohsen 202301138 , Jana Ahmad 202301800 , Ali Elrouby 202301680 , Mazen Ahmed 202300900

Problem Statement:

Network breach Detection System

Cyber threats are becoming more sophisticated, particularly in military networks, necessitating the development of adaptive machine learning-based solutions to identify new attack patterns.

We built a system that can detect if a network connection is safe (normal) or dangerous (anomaly).We used data from a military network, which includes examples of suspicious (anomaly) network activity. The goal is to train the system to correctly classify between normal and malicious (anomaly) behavior.

Challenges:

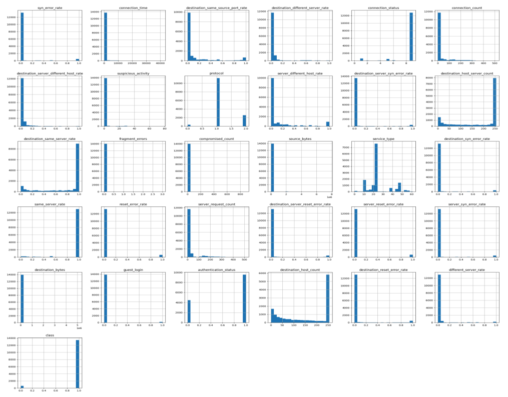
❖ The current dataset shows a severe class imbalance with all samples labeled as anomalies. In a real-world scenario, we would expect a mix of normal and anomalous connections.

❖ Feature Diversity: The dataset contains 31 quantitative and qualitative features describing network connection attributes requiring careful feature engineering and selection.

❖ Detection Accuracy: The model must achieve high precision in identifying true anomalies while minimizing false positives to avoid overwhelming security personnel with unnecessary alerts.

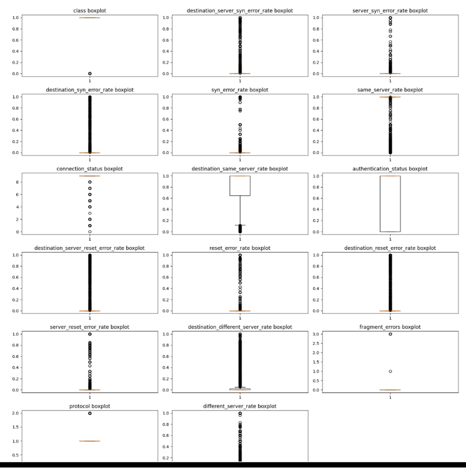
Data Exploration:

At first, we examined the shape of data and it was (14036,31) with 15 float columns, 12 integer columns and 4 string columns. The data was clean and there were no duplicates or nulls. We showed the correlation matrix of all features to see the dependencies and effect of features on target. We began to visualize the data to see the relations between features, and we visualized it using histograms, box plots, and scatter plots.



- Histograms:

We made a histogram for the whole dataframe with the function df.hist() to see the frequency and distribution of each feature. From observation, there were some columns that had some outliers which are (destination_host_count, destination_host_server_count, connection_count, service_type). Their range was too big compared with the other features.

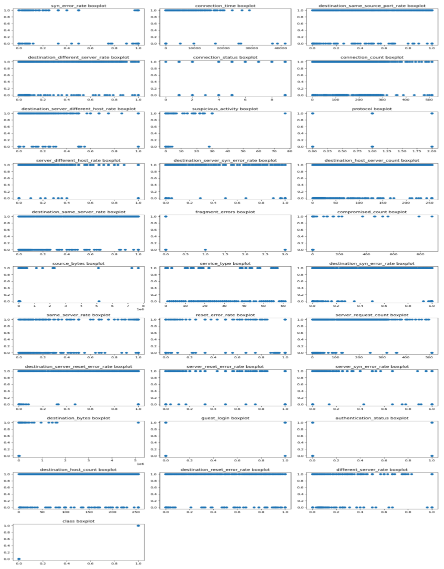


- Boxplot:

We made a boxplot for the whole dataframe using a for loop on all features to tell how the data is distributed relative to the IQR and min and max values. It cannot always be true as it can express the real data that has low frequencies.

- Scatter plot:

We made a scatter plot for the whole dataframe with a for loop to tell how the data is categorized to its categories. It is splitted up and down because it is a classification problem that has two classes (normal and anomaly) that can be 0 or 1.



Data Preprocessing:

Following the EDA, we made data processing techniques before implementing any model to ensure the quality of features.

We began by converting the categorical features to numerical features by using LabelEncoder. We used LabelEncoder instead of using OneHotEncoding to maintain the same dimensions and not increase the number of features too much.

We began by computing the correlation matrix of the dataset. We then sorted the absolute correlation values of each feature with respect to the target variable class. The goal was to identify the top 20 features most strongly correlated with the class, which could be potential candidates for model input.
We plotted histograms for all features in the dataset. This helped us understand the distribution of values across each feature, identify any skewness, and detect potential outliers or anomalies.

Boxplots were generated for each feature to further investigate outliers. This visualization helped us detect features with extreme values that could affect model training or scaling performance. To assess how features relate to the target variable class, scatter plots were created. These plots provided visual insights into potential separability between classes based on specific features.

From the correlation analysis, we selected the top 20 most relevant features (including the class column) and created a new DataFrame df2. This helped reduce noise and focus the model on the most informative features. Then, we splitted down the data into x and y. x is the features, so we dropped the column of class from it, while y is the target so it is the class column.

We used RobustScaler to scale selected features that can be not affected by outliers. Unlike standard scaling, RobustScaler uses the median and interquartile range, making it more resilient to extreme values. We made RobustScaler after splitting to prevent data leakage and help in increasing the accuracy of the training data. Also, we applied the RobustScaler by fitting it on only the training data while transforming it to the whole data. This is because the validation and testing data are unseen data that cannot be used in fitting calculations.

The original versions of the scaled features were dropped from the DataFrame, and their scaled versions were reinserted into the training and testing datasets. This finalized the cleaned and scaled features set for model training.

Model Selection and Implementation:

1) KNN:

K-Nearest Neighbors (K-NN) is a simple, powerful supervised machine learning algorithm used for both classification and regression tasks. It works well with both numerical and categorical data, and its transparency allows checking similar connections for anomalies. We chose to use (K-NN) to detect anomalies easily with no training, and work with complex patterns and mixed data.

Hyperparameters:

In the k-Nearest Neighbours (k-NN) algorithm (k) is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

Implementation:

After preprocessing the data then splitting the data into (x_train,y_train,x_test,y_test) we specify the n_neighbours = 3, `knn = KNeighborsClassifier(n_neighbors=3)` then fitting the model by (x_train,y_train)`knn.fit(x_train, y_train)` and predict using (x_test) `knn_pred = knn.predict(x_test)`

2) RidgeRegression:

Ridge regression is a regularization technique used in linear regression.We use RidgeRegression to prevent overfitting by adding a penalty term to the loss function that is proportional to the square of the model's coefficients.

Hyperparameters:

α (alpha): Regularization strength (L2 penalty).
max_iter (t): Max iterations for convergence.

Implementation:

After preprocessing the data then splitting the data into (x_train,y_train,x_test,y_test) we specify the alpha = 5 after many trials to find the best_alpha `ridge = RidgeClassifier(alpha=5)` then
Fitting the model by (x_train,y_train)`ridge.fit(x_train, y_train)` and predict using (x_test) `ridge_pred_scores = ridge.predict(x_test)`

3) Logistic regression:

It is a binary classifier that uses the sigmoid function to simulate being classified to a specific class, producing values between 0 and 1. We selected Logistic Regression due to its simplicity, interpretability, and efficiency for binary classification tasks. It performs well when the data is linearly separable and is especially useful for baseline model performance comparison.

Hyperparameters:

C: Inverse of regularization strength. Smaller values specify stronger regularization.

Implementation:

We initialize the C then implement the classifier log_reg = LogisticRegression(C=1.0) and then predict and test.

4) <u>SVM:</u>

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. It aims to find the optimal hyperplane that best separates data into distinct classes. We used SVM for its robustness in handling high-dimensional feature spaces and its effectiveness in non-linear classification problems using kernel tricks.

Hyperparameters:

- C: Regularization parameter. A smaller value allows for a softer margin, while a larger value aims for stricter separation.

- kernel: Specifies the kernel type (linear, poly, rbf).

- gamma: Defines how far the influence of a single training example reaches (used in rbf, poly kernels).

Implementation:

After preprocessing the data then splitting the data into (x_train,y_train,x_test,y_test) we specify C, kernel, and gamma then implement svm_model = SVC(C=1.0, kernel='rbf', gamma='scale') and finally fit and predict.

5) <u>Decision tree:</u>

A decision tree is a supervised learning algorithm used for both classification and regression tasks. It models decisions as a tree-like structure where internal nodes represent attribute tests, branches represent attribute values and leaf nodes represent final decisions or predictions.
We use Decision Trees for their simplicity, ease of interpretation, and ability to handle both numerical and categorical data.

Hyperparameters:

max_depth: The maximum depth of the tree. Limiting depth can prevent overfitting

Implementation:

```
model = DecisionTreeClassifier(max_depth=6)
model.fit(x_train, y_train)
text_representation = tree.export_text(model)
tree_pred = model.predict(x_test)
```

6) Random forest:

Random Forest algorithm is a powerful tree learning technique in Machine Learning to make predictions and then we do voting of all the trees to make predictions.
It takes different random parts of the dataset to train each tree, and then it combines the results by averaging them. This approach helps improve the accuracy of predictions. Random Forest is based on ensemble learning. We chose to use Random Forests to improve accuracy and reduce overfitting by combining multiple decision trees for more robust predictions.

Hyperparameters:

n_estimators: Number of trees in the forest (default is 100).

Implementation:

After preprocessing the data, then splitting the data is split into (x_train,y_train,x_test,y_test)

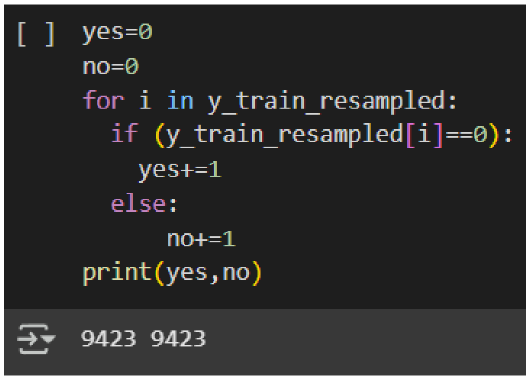
```
rf_classifier = RandomForestClassifier(n_estimators=100)
rf_classifier.fit(x_train, y_train)
y_pred_Forest= rf_classifier.predict(x_test)
```

Model Evaluation:

Using Models before balancing:
After splitting the data and applying RobustScaler to remove outliers, we started training the models and testing them and calculating their accuracy:
Accuracy of KNN: 0.9964387464387464
Accuracy of Linear Classifier: 0.9886039886039886
Accuracy of Ridge regression: 0.9882478632478633
Accuracy of Logistic regressor: 0.9615384615384616
Accuracy of SVM: 0.9882478632478633
Accuracy of Tree: 0.9921652421652422
Accuracy of Random Forest: 0.9967948717948718
The highest accuracy model was Random Forest, and the lowest accuracy was Logistic Regressor. We also applied the ensembling techniques Voting, Bagging, and Stacking. Their accuracy was:
Accuracy of Soft Voting: 0.9957264957264957
Accuracy of Hard Voting: 0.9960826210826211
Accuracy of Stacking: 0.9971509971509972
Accuracy of Bagging: 0.9971509971509972
The highest accuracy was Stacking and Bagging, and the lowest accuracy was Soft Voting. We chose these techniques because they deal with data classification.
We observed that the accuracy of the models was too high, so it might be overfitting. We checked the data and observed that the data was not balanced (divided into 13,449 normal and 587 anomalies).

So, we searched about balancing techniques and found two main techniques; sampling up (SMOT) and sampling down (Near miss):

Using Models after Sampling up (SMOT):
To check for overfitting, we splitted the data into training, validation and testing sets to visualize the three sets and see how the model learns.
And to balance the data, Synthetic Minority Oversampling Technique (SMOTE) technique increase the minor class to equal the major class. We applied the SMOTE on the training data only as validation and testing data are unseen. After applying SMOTE, we checked for the balance of the classes, and it applied correctly.



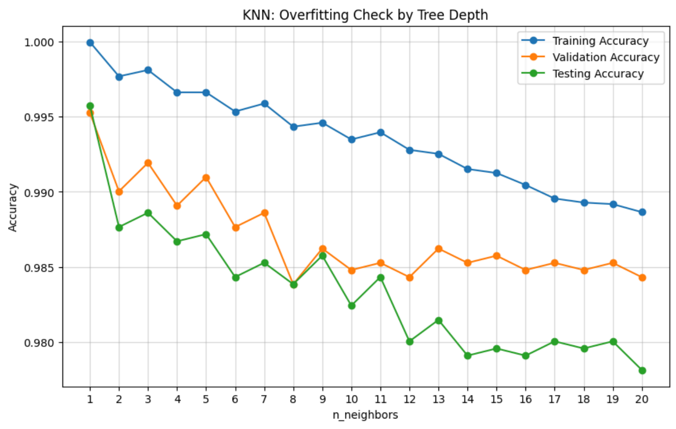
```
[ ]  yes=0
     no=0
     for i in y_train_resampled:
       if (y_train_resampled[i]==0):
         yes+=1
       else:
           no+=1
     print(yes,no)

     9423 9423
```

We tested our models after SMOTE, tuned our hyperparameters and visualized it to see the difference. We observed that the accuracy improved a lot by seeing that the validation (train-validation-test) graph doesn't show any overfitting as the curves are close to each other.
Accuracies of models were:
Accuracy of KNN after SMOTE: 0.9886039886039886
Accuracy of Ridge regression after SMOTE: 0.9586894586894587
Accuracy of Logistic regression after SMOTE: 0.9601139601139601
Accuracy of SVM after SMOTE: 0.9890788224121557
Accuracy of Tree after SMOTE: 0.9843304843304843
Accuracy of randomforest after SMOTE: 0.99667616334283

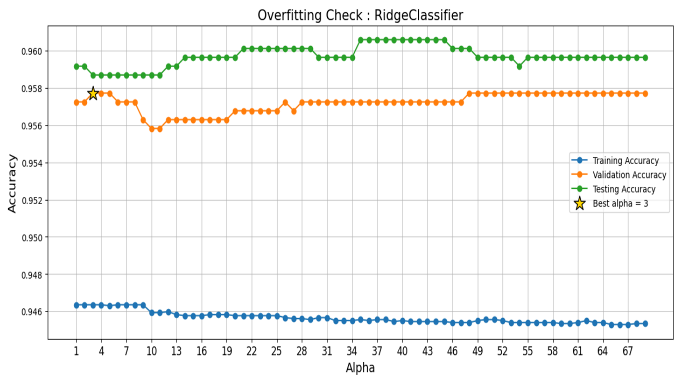
For the validation graphs:

KNN:
We tried values of K from 1 to 21 and saw the difference between them, the best k was nearly 3.
The graph in general doesn't show overfitting as the training accuracy curve is the highest one, while the validation and testing curves are lowest but still close to each other which show a very good testing accuracy of average 0.98



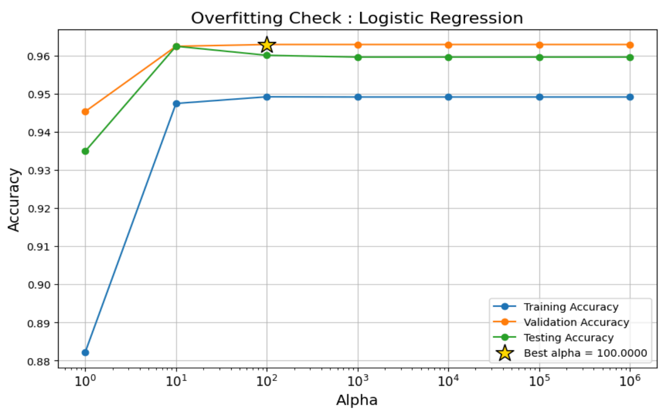
Ridge classifier:

We tried alpha values from 1 to 70 and saw the difference between them, the best alpha was nearly 3.
The graph shows a bit of overfitting as the training testing curve is the highest one, while the validation and training curves are lowest, which can indicate that the ridge classifier is not the best option for us for classifying this data. It shows a medium good testing accuracy of average 0.958



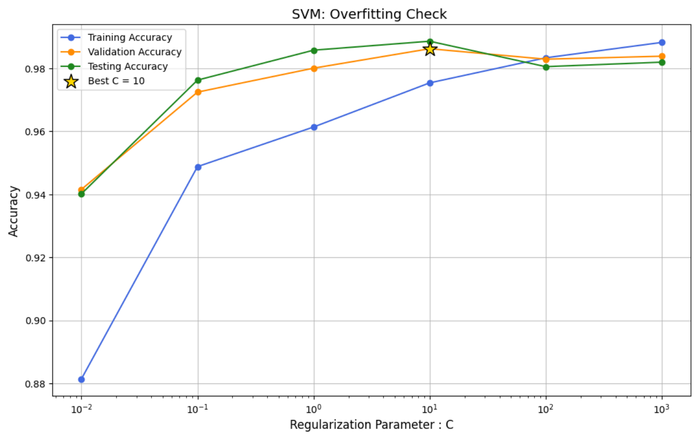
Logistic regression:

We tried max iterations from 1 to 10000000 and saw the difference between them, the best max iteration was 100. The graph shows less overfitting than the ridge regression but still had a little overfitting percentage as the training testing curve is the highest one, while the validation and training curves are lowest. This can indicates the the logistic regression is a medium classifier for our data of an average accuracy of 0.96



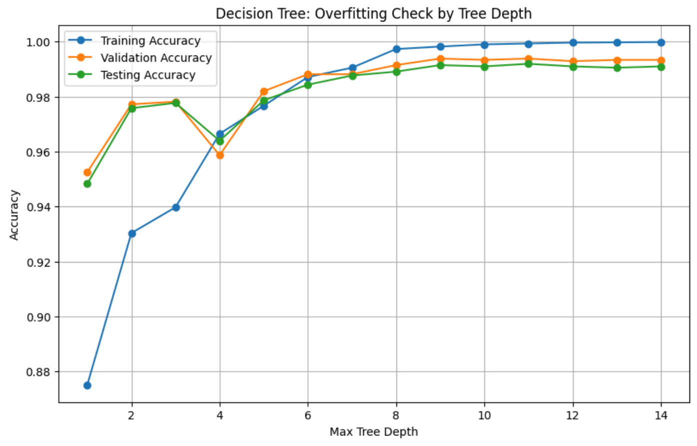
SVM:

We tried C values from 0.01 to 1000 which is the best range to examine and saw the difference between them, the best C was 10.
The graph shows a good relation and distance between the training, validation and testing lines more than logistic regression but not the best. This can indicates that the SVM is a slightly good classifier for our data with an average accuracy of 0.989



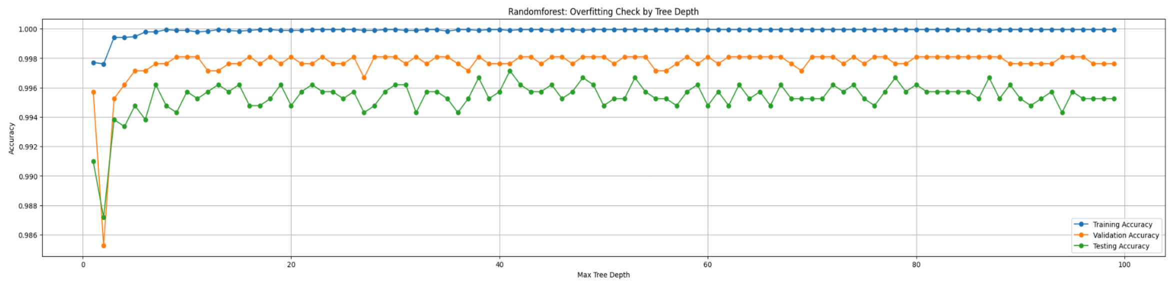
Decision tree:

We tried max depth from 1 to 15 and saw the difference between them, the best max depth was 11. The graph shows a very good relation between the training, validation and testing lines more than both logistic regression and decision tree but not still the best as all the lines nearly are so close. This can indicates that the decision tree is a very good classifier for our data with an average accuracy of 0.984



Random forest:

We tried n_estimator from 0 to 100 and saw the difference between them, the best max n_estimator was nearly 74. The graph shows the best relation between the training, validation, and testing lines more than all of them. This can indicates that the random forest is the best and the most strongest classifier for our data with an average accuracy of 0.996



Using Models after Sampling down (Near Miss):

We tried the NearMiss technique, as a possible way for sampling down .To check for overfitting, we split the data into training, validation, and testing sets to visualize the three sets and observe how the model learns. To balance the data, we applied the NearMiss technique, which reduces the majority class to match the size of the minority class. We applied NearMiss only on the training data, as the validation and testing data remain unseen. After applying NearMiss, we checked the balance of the classes, and it was correctly applied.

We tested our models after applying NearMiss, tuned our hyperparameters, and visualized the results to see the difference. We observed that the accuracy decreased significantly compared to before, as the validation (train-validation-test) graph showed a large gap between the curves, indicating potential issues with the feature set or data size.

Before Undersampling, counts of label 'Normal': 9423

Before Undersampling, counts of label 'Anomaly': 402

After Undersampling, the shape of train_X: (804, 20)

After Undersampling, the shape of train_y: (804,)

After Undersampling, counts of label 'Normal': 402

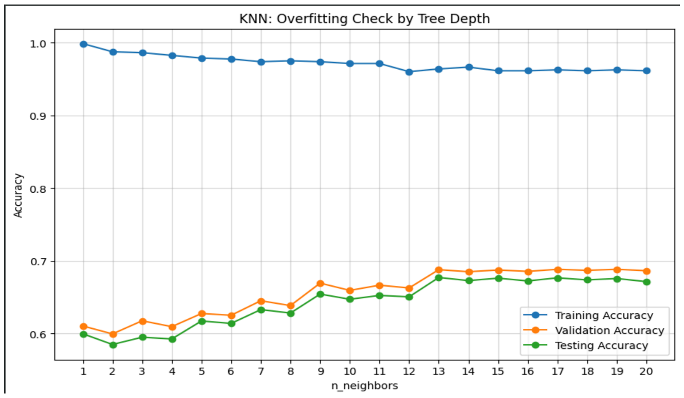
After Undersampling, counts of label 'Anomaly': 402

Accuracies of models were:
Accuracy of KNN after SMOT: 0.5949667616334283
Accuracy of Ridge regression: 0.7207977207977208
Accuracy of Logistic regressor: 0.6172839506172839
Accuracy of SVM: 0.5807217473884141
Accuracy of Tree: 0.6016144349477682
Accuracy of randomforest: 0.5826210826210826

For the validation graphs:

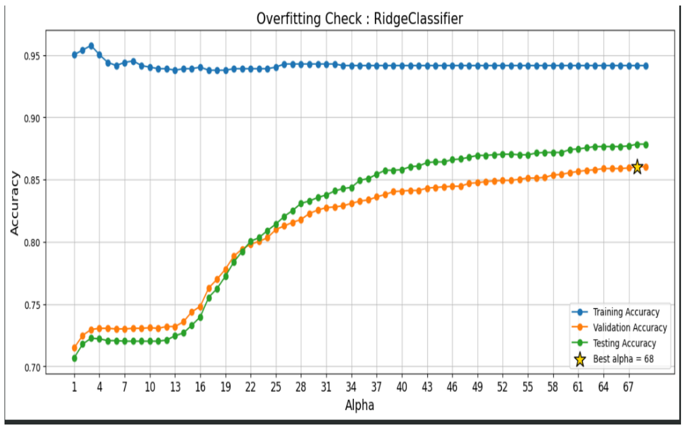
KNN:

We tried values of K from 1 to 21 and saw the difference between them, the best k was nearly 3.
The graph in general doesn't show overfitting as the training accuracy curve is the highest one, while the validation and testing curves are lowest but still close to each other which show a very bad testing accuracy of average 0.59



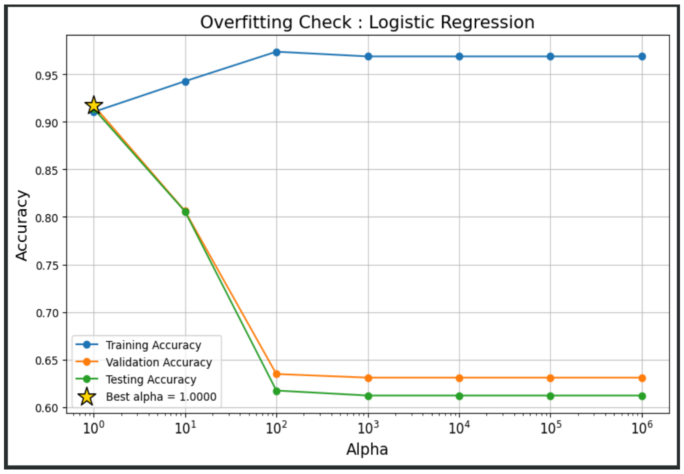
Ridge classifier:

We tried alpha values from 1 to 70 and saw the difference between them, the best alpha was nearly 3. The graph shows an overfitting as the training curve is the highest one, while the validation and testing curves are lowest, which can indicate that the ridge classifier is not the best option for us for classifying this data. It shows a bad testing accuracy of average 0.72



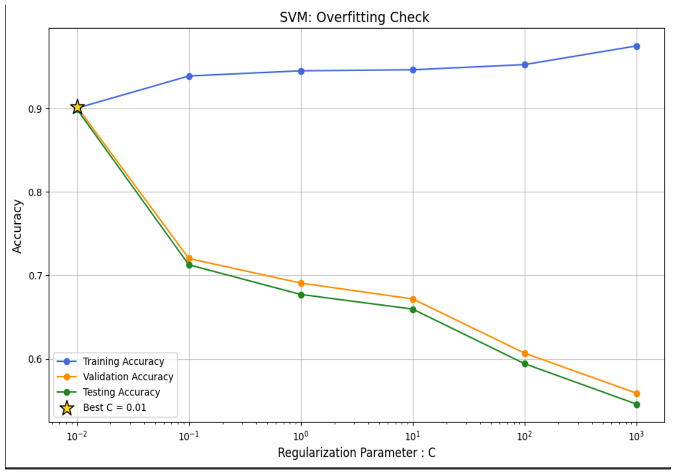
Logistic regression:

We tried max iterations from 1 to 10000000 and saw the difference between them, the best max iteration was 100. The graph shows less overfitting than the ridge regression but still had a little overfitting percentage as the training testing curve is the highest one, while the validation and training curves are lowest. This can indicates the the logistic regression is a bad classifier for our data of an average accuracy of 0.61



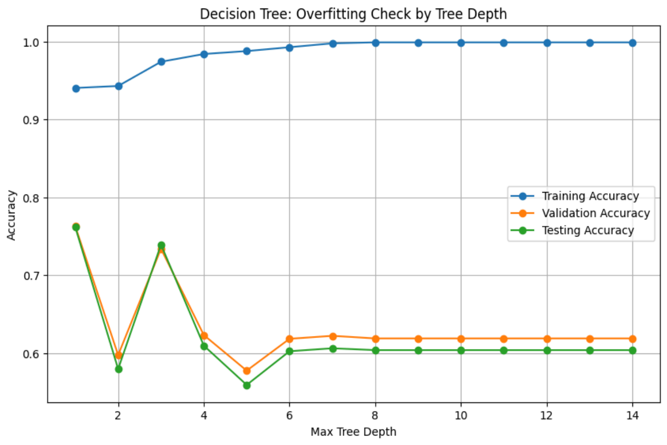
SVM:

We tried C values from 0.01 to 1000 which is the best range to examine and saw the difference between them, the best C was 10.
The graph shows a bad relation and distance between the training, validation and testing lines more than logistic regression which is not the best. This can indicates that the SVM is a very bad classifier for our data with an average accuracy of 0.58



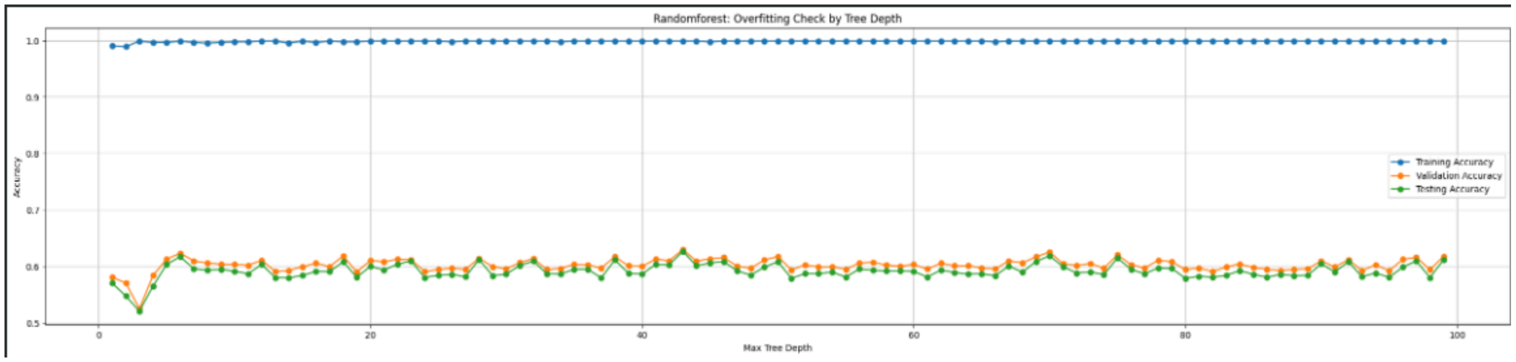
Decision tree:

We tried max depth from 1 to 15 and saw the difference between them, the best max depth was 11.
The graph shows a very bad relation between the training, validation and testing lines more than both logistic regression and decision tree but not still the best as all the lines nearly are so close. This can indicates that the decision tree is a very bad classifier for our data with an average accuracy of 0.601



Random forest:

 We tried n_estimator from 0 to 100 and saw the difference between them, the best max n_estimator was nearly 74. The graph shows the worst relation between the training, validation, and testing lines more than all of them. This can indicates that the random forest is very bad and the weakest classifier for our data with an average accuracy of 0.58



Conclusion:

First we made preprocessing techniques LabelEncoder and RobustScaler .Then we tested six models: KNN, Linear Classifier, Ridge regression, Logistic regression, SVM, Decision Tree and Randomforest. Then we tried ensemble techniques: Voting, Bagging, Stackingز After that , we tried to resample the data set in two ways, sampling up (SMOTE) and sampling down (Near miss).Finally, we found the best model was Random forest after SMOT of accuracy 0.996.

Task division:

- Jana Ahmed: Data cleaning, EDA, SMOT visualization
- Ahmed Mohsen: Model implementation, Near miss visualization
- Ali Elrouby: Data preprocessing, SMOT implementation
- Mazen Ahmed: Near miss implementation, Feature engineering
- All of us: report writing and investigation.