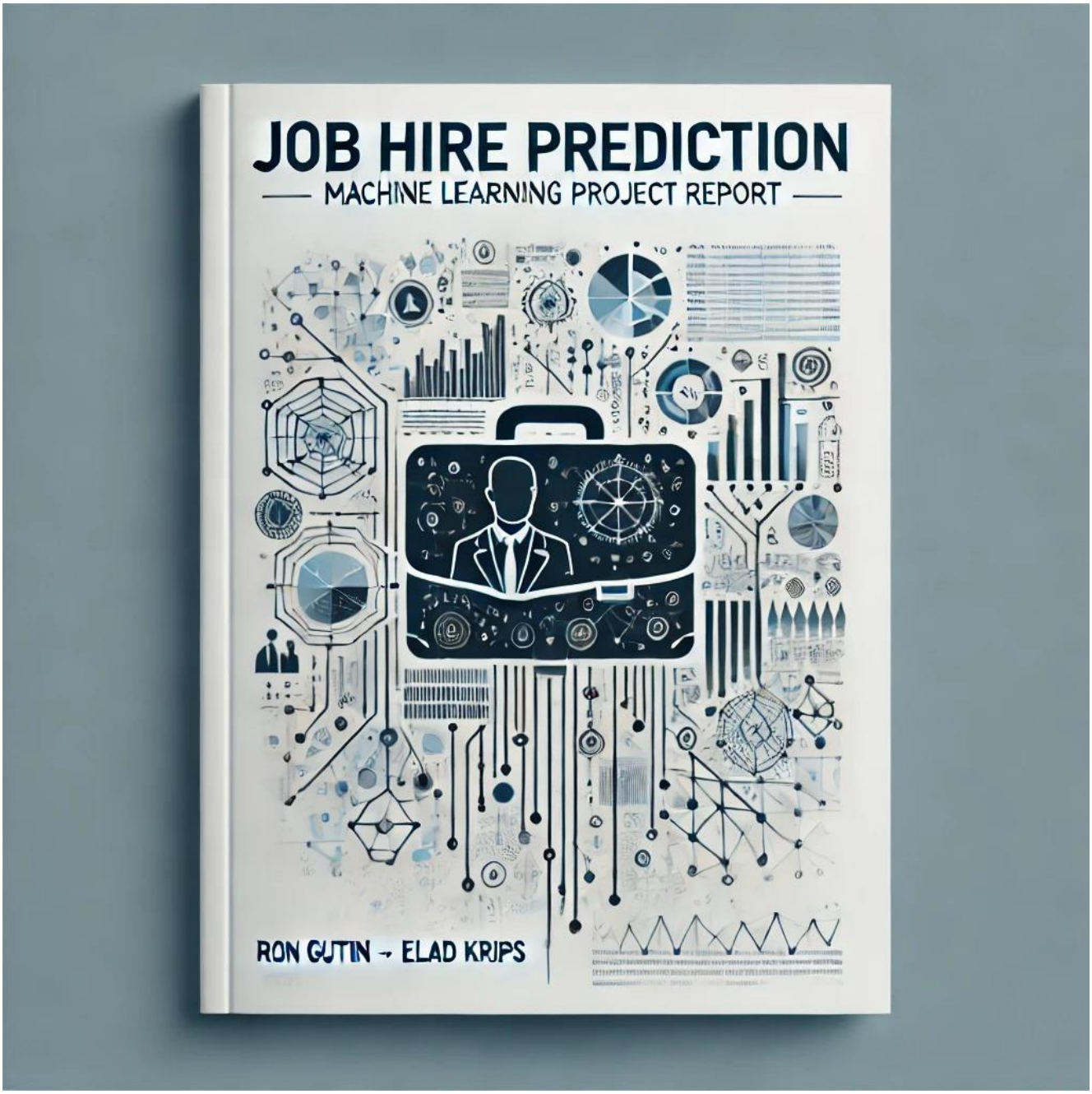
Machine Learning Project Report: Job Acceptance Prediction

Ron Gutin – 209305341 | Elad Krips - 211895958



Introduction

In this project, we were tasked with developing a machine learning model to predict the probability of a candidate being accepted for an unspecified job. This involved experimenting with a real-world dataset.

Part 1 - Data Exploration

The project starts by importing libraries and loading the dataset from 'train.csv'. Initial exploration includes `df.head()` for first rows, `df.info()` for column details, and `df.describe(include='all')` for a statistical summary. The ID column is excluded due to unique values per sample.

Numerical Features Analysis

Histograms and box plots were utilized to visualize the distribution of numerical features.(**Appendix 1**)

This analysis helped us understand the spread, skewness, and presence of outliers. From the histograms, we observed that features 'A' and 'D' follow a normal distribution, while 'B', 'years_of_experience', and 'prev_salary' do not. These features will be referred to accordingly throughout the report.

Categorical Features Analysis

We examined value counts for each categorical feature to understand the distribution of categories.

Correlation Analysis

A correlation matrix (heatmap) was plotted to identify relationships between numerical features. Notable correlation was observed between 'B' and 'years_of_experience', which is 90% (very high). Another important correlation we noticed is between ‘D’ and ‘label’ – 41% that indicates ‘D’ might be important for our analysis. (**Appendix 2**)

Part 2 - Data Preprocessing

Outlier Detection and Removal

First, we thought what are the right ways to detect outliers according to each distribution. According to this **article**, the most common way to detect outliers from normally distributed data is using the Z-score method which applies to features 'A' and 'D'. As we noticed in the data exploration part, the features 'B', 'years_of_experience', 'prev_salary' aren’t normally distributed, hence we conclude that we should use other outlier detection method. We chose to use IQR method since its robust for non-normally distributed data.

- Z-score method for features assumed to follow a normal distribution ('A' and 'D').
- IQR method for features assumed to doesn't follow a normal distribution ('B', 'years_of_experience', 'prev_salary').

As we can see the output of our program, while using Z-score (Z = 3)* we didn't detect any outliers, while using IQR method we were able to detect 3625 outliers. (**Appendix 3.2**)

*We decided to use the common value Z=3 to detect outliers according to this **article**. (**Appendix 3.1**)

Data Normalization

It seems that some features might not be normalized like: 'A', 'B', 'D', 'years_of_experience' and 'prev_salary' ('ID' and 'label' are numeric but we aren't normalizing them because they aren't relevant for model training). We chose to normalize these features for several key reasons:

-Improves Model Performance: Ensures all features contribute equally by scaling them to a similar range, preventing dominance by larger numerical ranges.

-Facilitates Efficient Computation: Reduces computational complexity and resource consumption, enabling faster execution and reduced memory usage.

-Enhances Model Interpretability: Standardizes data for easier comparison and interpretation of model coefficients or feature importance.

Min-Max scaling preserves the relationships among the original data values and sets a [0,1] boundaries for all numerical features which can be important for our analyses.

Null Handling

We checked if there are null values in our data using df.isnull().sum() command.
As we can see in the output, every column except 'ID' and 'label' has null values.
(**Appendix 4**)

We chose to handle different features using different methods.
We wrote a function called fix_null with the next logic:

1.**'worked_in_the_past'** - if the person has work experience we set it to 'T' else set to 'F'.

2.**'years_of_experience'** – if the person worked in the past the median value of this feature will be set, else we set it to 0.

3.**'sex'** – if the value is null we are not assuming the gender and setting the value to 'other'.

4.**'education'** - if the value is null we are not assuming the type of education and setting the value to 'other'. We assume that 'other' is the lowest type education and we don't want to give a person extra credit because we think it might affect his acceptance rate.

5.**'is_dev', 'disability', 'mental_issues', 'C', 'country', 'age_group'**-
Fill missing values with the mode (most common value) for each column to impute a likely value, maintaining data consistency without introducing bias.

6.**'prev_salary' , 'A' , 'B', 'D'-** Fill missing values with the median. We decided to handle them like that because 'A','B','D' are anonymous and it is reasonable to do so.
For 'prev_salary' we asked 3 different language models about their opinion for this null handling and they all suggested to set the median.

7.**'stack_experience'** – we decided to consider null cell as an unskilled candidate. Which will be handled in the next part.

Categorial Features

'Education' is a feature that represents the highest level of education achieved by an individual. Conversion: We map various education levels to numerical values using the following mapping: 'High school' to 1, 'BA/BSc' to 2, 'MA/MSc' to 3, 'PhD' to 4, and 'other' to 0. This ordinal transformation captures the hierarchical nature of educational attainment.

'Sex' is a feature that represents the gender of an individual. We used one-hot encoding

to create separate binary columns for each gender category. This conversion would handle the categorical nature of gender appropriately for the model.
'C' is an anonymous feature and we don't know what it represents. Since we don't know 'C's meaning we chose to use one-hot encoding to not over/under value specific category. Using this method we divided 'C' to 7 different features.
'Country' – We observed 170 unique categories in this feature. Grouping by continents could introduce biases due to over/under-representation. One-hot encoding was another option but it would create 170 sparse features, leading to inefficiencies. Therefore, we decided to ignore this feature in our analysis.
'Stack_Experience' – A feature that represents the technological skills of a candidate. We decided to transform the list of skills to the amount of them because it seemed simple and logical. Later we discovered due to a feature importance test we conducted on the data that this feature is the most crucial one for our prediction. (**Appendix 5**)
As a result of the feature importance test we decided to handle this feature with the next technique: first counting the occurrences of each skill across all entries, then filtering out skills that appeared less than 500 times, we have tested few different thresholds e.g. 100,250,1000,2000 and 500 resulted the highest AUC score. For the remaining frequently mentioned skills, binary columns were created to indicate their presence or absence in each row. Notice that null values of this feature means that the candidate have no skills.
'is_dev', 'disability', 'mental_issues', 'age_group', **'worked_in_the_past'**- We transformed all of those features to binary format (basic logic), with specific values mapped to 1 and all other values mapped to 0.

Dimensionality and PCA

The dimensionality of the problem is not too big, because there are more than 50K samples in our data and no more than 150 features leading to a minimal ratio of 1:300 feature to sample.
High dimensionality can cause several issues in machine learning, such as the curse of dimensionality, where data points become sparse and patterns are difficult to detect. It can lead to overfitting, where models fit noise rather than true patterns, resulting in poor generalization. Additionally, it increases computational costs, adds data redundancy with irrelevant features, and reduces model interpretability. To evaluate if the dimensionality is too high, one can look at model performance discrepancies between training and validation sets, analyze feature importance to identify redundancy, and use PCA to see if a few components explain most of the variance. For instance, if PCA shows that a small number of components capture the majority of the variance, it suggests that the feature space may be unnecessarily large. Another rule of thumb for too high dimensionality is a 1:10 minimal ratio between features and samples.

In our project we decided to use feature selection for dimensionality reduction.
We decided to exclude the following features: 'country','ID','C','B','sex', 'mental_issues' ,'disability','age_group','worked_in_the_past'.
'B' and 'years_of_experience' are highly correlated(0.9), hence we decided to exclude 'B' because it is anonymous and in the feature importance analysis it comes after 'years_of_experience'. (**Appendix 5**)

'ID' – unique feature for every sample, won't contribute at all.
'country' – explained in categorial features part.
'mental_issues','disability','age_group','worked_in_the_past','C' and 'sex' – As we saw in the histogram of the importance, those features are not contributing to the success of our models. (**Appendix 5**)
The reduction of features improved our model's speed by utilizing only the important features and increased our AUC score compared to the AUC score after PCA reduction. Therefore, we decided not to continue with PCA and to perform manual feature selection only. After applying PCA in the Random Forest model, the train AUC increased to 0.99 while the test AUC dropped to 0.92. Despite the increased run time, we suspected overfitting and decided to avoid using PCA. (**Appendix 11**)

New features – As we already mentioned, we divided the 'stack_experience' feature into a standalone skills using the method we mentioned in the Categorial Features handling. As a mathematical manipulation, we used Min Max Normalization (as explained earlier).

Part 3 – Model Test Run

The data was split into training, validation, and test sets using train_test_split, allocating 25% to the test set and maintaining class distribution with stratify. Additionally, the training data was further split into training and validation sets with a 20% split, and it was verified that the classes are balanced.
Logistic Regression – The hyperparameters we chose to change are 'c' the regularization strength in order to optimize the bias-variance trade-off, default 'c' is 1 we discovered that 'c'=0.1 is the optimal 'c'. We changed the regularization type from 'L2'-ridge to 'L1'-lasso because we suspected that part of the features aren't useful and as we learned in class L1 regularization might reduce the number of features being considered. We changed the 'solver' from the default one to 'liblinear' because it supports 'L1' regularization.**(Appendix 7)**
KNN – We selected the number of neighbors K, as the hyperparameter to tune in our model. The number of neighbors affects the bias-variance trade-off, as the higher the K, the bias rises and the variance drops. While the default value is K=5, we adjusted it to K=17 based on the results of a 3-fold cross-validation we conducted. This adjustment was guided by optimizing the AUC-ROC score, ensuring the best performance for our model. **(Appendix 8)**
Random Forest – We chose to use GridSearch-cross validation algorithm to calculate the optimal hyperparameters based on the AUC-ROC score. Full list of parameters at (**Appendix 13**).
'criterion': Changed from 'gini' to 'entropy' because it provided better results for our dataset by handling complex relationships more effectively.
'max_depth': Set to 15 instead of 'None' to balance the model's ability to generalize while avoiding overfitting.
'min_samples_split': Increased from 2 to 5 to reduce the risk of overfitting by requiring more samples to split a node.
'n_estimators': Increased from 100 to 150 to enhance model robustness and reduce

variance by averaging over more trees.

Adaboost – As in Random Forest, we used GridSearch-cross validation algorithm to calculate optimal hyperparameter values. Full list of parameters at (**Appendix 12**).
'max_depth': Changed from 'None' to 2 to create shallow trees that reduce overfitting, which is suitable given the boosting nature of AdaBoost.
'min_samples_leaf': : Changed from 1 to 3 to prevent trees from becoming too specific by ensuring each leaf has at least three samples.
'n_estimators': Increased from 50 to 200 to boost performance by using more weak learners, which enhances the ensemble's overall predictive accuracy.

Part 4 – Model Evaluation

Confusion Matrix – A table used to evaluate the performance of a classification model by comparing the predicted labels with the actual labels. It consists of four cells, each representing a different outcome of the classification. We are going to analyse the confusion matrix based on the Adaboost model, on the test data that we extracted from the 'train.csv'.
TP(True Positive) - Correctly identified as 'Hired' , in our case its **6507**.
FP(False Positive) - Incorrectly identified as 'Hired' (false alarms), in our case its **591**.
TN(True Negative) - Correctly identified as 'Not Hired', in our case its **5376**.
FN(False Negative) - Incorrectly identified as 'Not Hired', in our case its **486**.
According to our business analysis a company would like to avoid hiring a candidate that shouldn't be hired (time, resource and money waste). Therefore the most important metric for us is FPR, which in our test is 0.099. This metric indicates a well preforming model. Additional metrics that we think are relevant: Accuracy ≈ 0.917, Recall ≈ 0.931 ,Precision ≈0.916, Specificity ≈ 0.901.

K-Fold Cross Validation – This ROC curve shows the performance of the AdaBoost model across five different folds in a cross-validation process. Each fold's ROC curve has an AUC of 0.98, indicating consistent and high model performance, with excellent discrimination between the positive and negative classes across all folds. **(Appendix 9)**

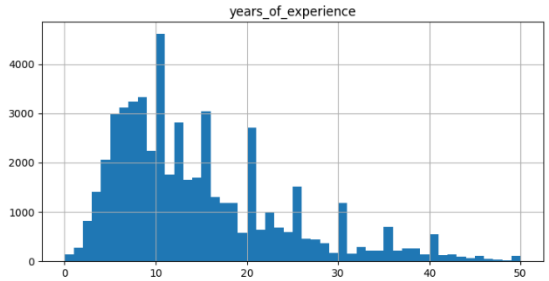
Model Performance – Our model isn't overfitting, as shown by the small AUC score difference (less than 0.1) between the train and test sets. This indicates that the model generalizes well and has learned patterns that apply to both the training and test data. To reduce overfitting, we removed a condition for excluding outliers that applied to only a single sample. By eliminating this overly specific condition, we improved the model's ability to generalize to new data. **(Appendix 10)**

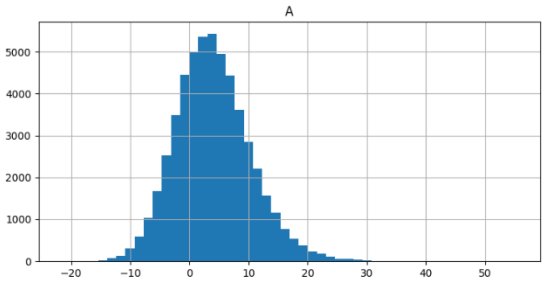
Part 6 – Out Of Scope Usage

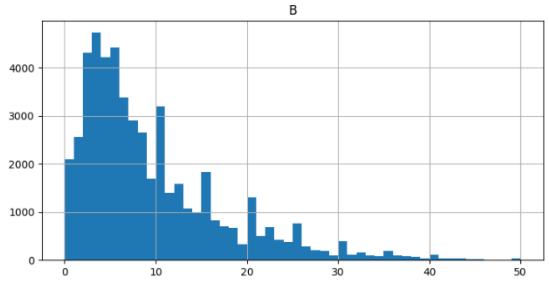
1.**IQR** - The Interquartile Range (IQR) method identifies outliers by measuring the spread between the first and third quartiles and finding values that fall significantly outside this range. It is effective for non-normally distributed data because it focuses on the middle 50% of the data, making it less sensitive to skewed distributions and extreme values. We used this method to reduce noise in the dataset.
2. **Threading -** Threading speeds up the process of finding the optimal K for KNN by allowing multiple computations to run concurrently. This parallelization reduces the overall time required for evaluating different K values, leading to faster optimization.

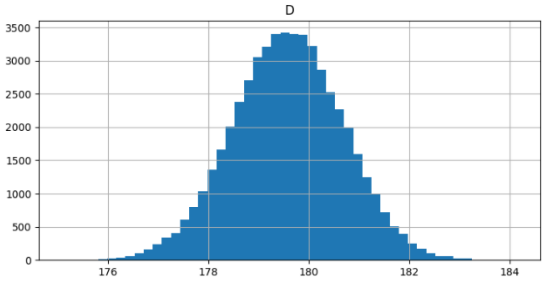
Appendix

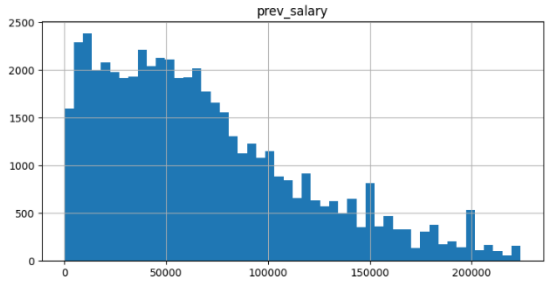
1. Numerical Features Analysis –see ‘A’ and ‘D' act normally. ‘B’, years_of_experience’ and ‘prev_salary’ aren’t acting normally.

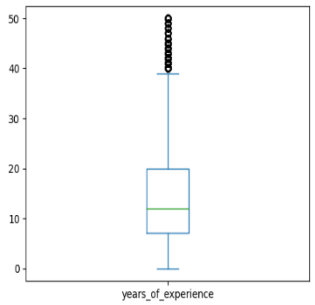


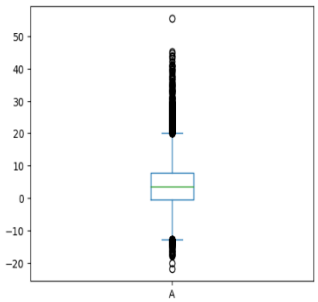


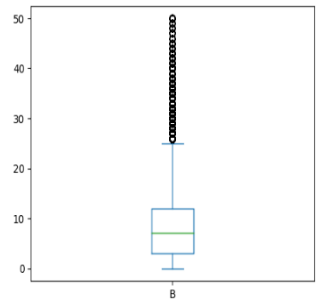


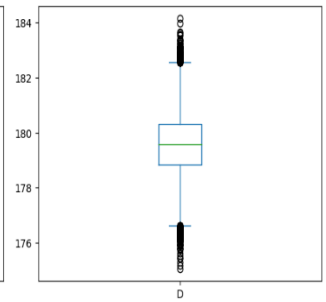


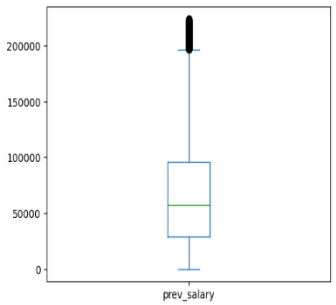




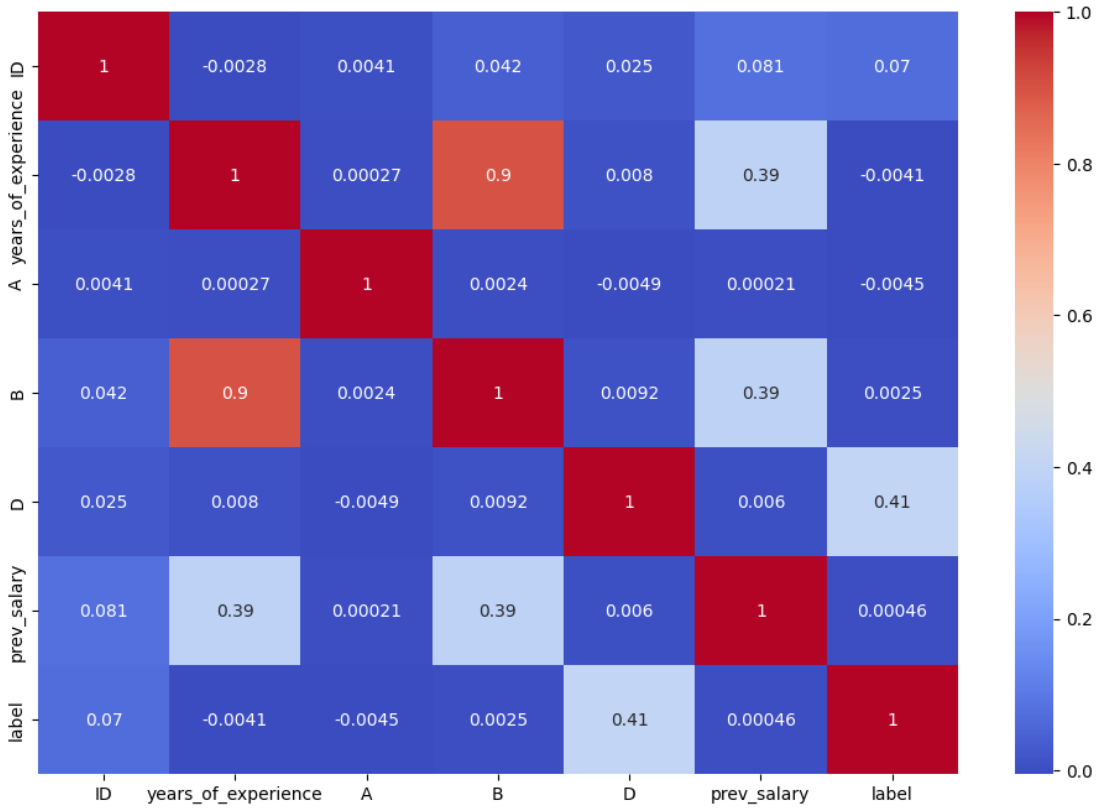








2. Correlation Analysis – see high correlation between 'B' and 'years_of_experience' and the highest correlation with 'label' is with 'D'.



3. Outliers Detection and Removal –

3.1 -

https://www.researchgate.net/publication/364095357_Detecting_Outliers_in_High_Dimensional_Data_Sets_Using_Z-Score_Methodology

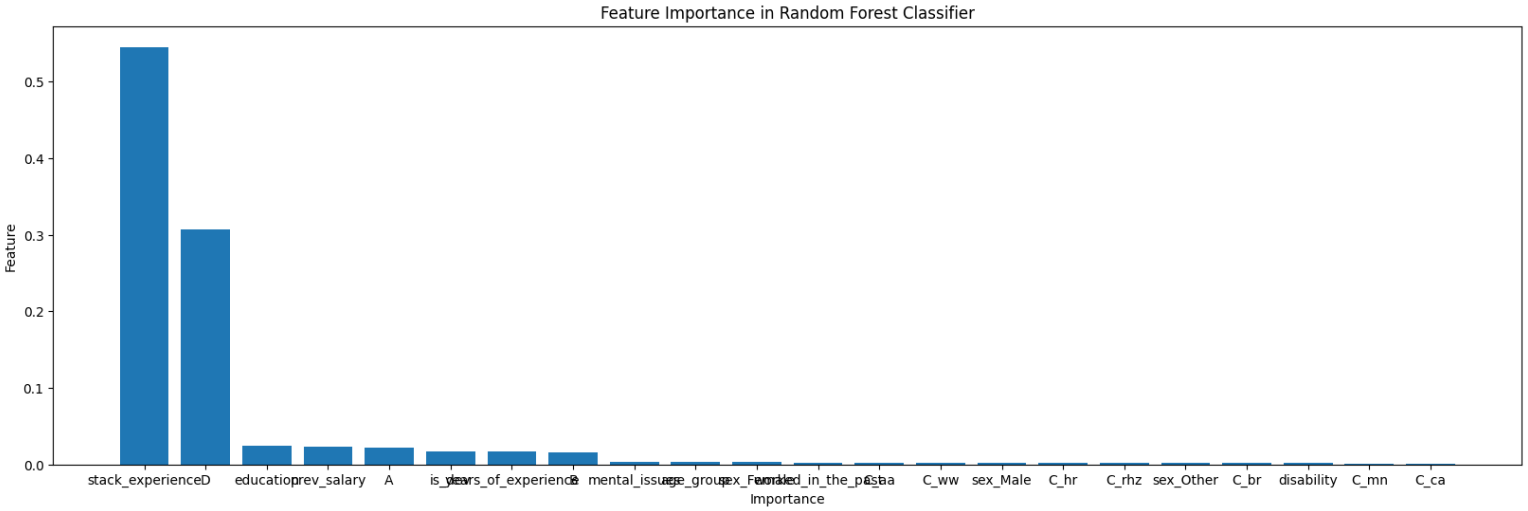
```
Total amount of outliers in features : ['A', 'D'] using Z-score method is: 0
Total amount of outliers in features : ['B', 'years_of_experience', 'prev_salary'] using IQR method is: 3625
```

3.2 -

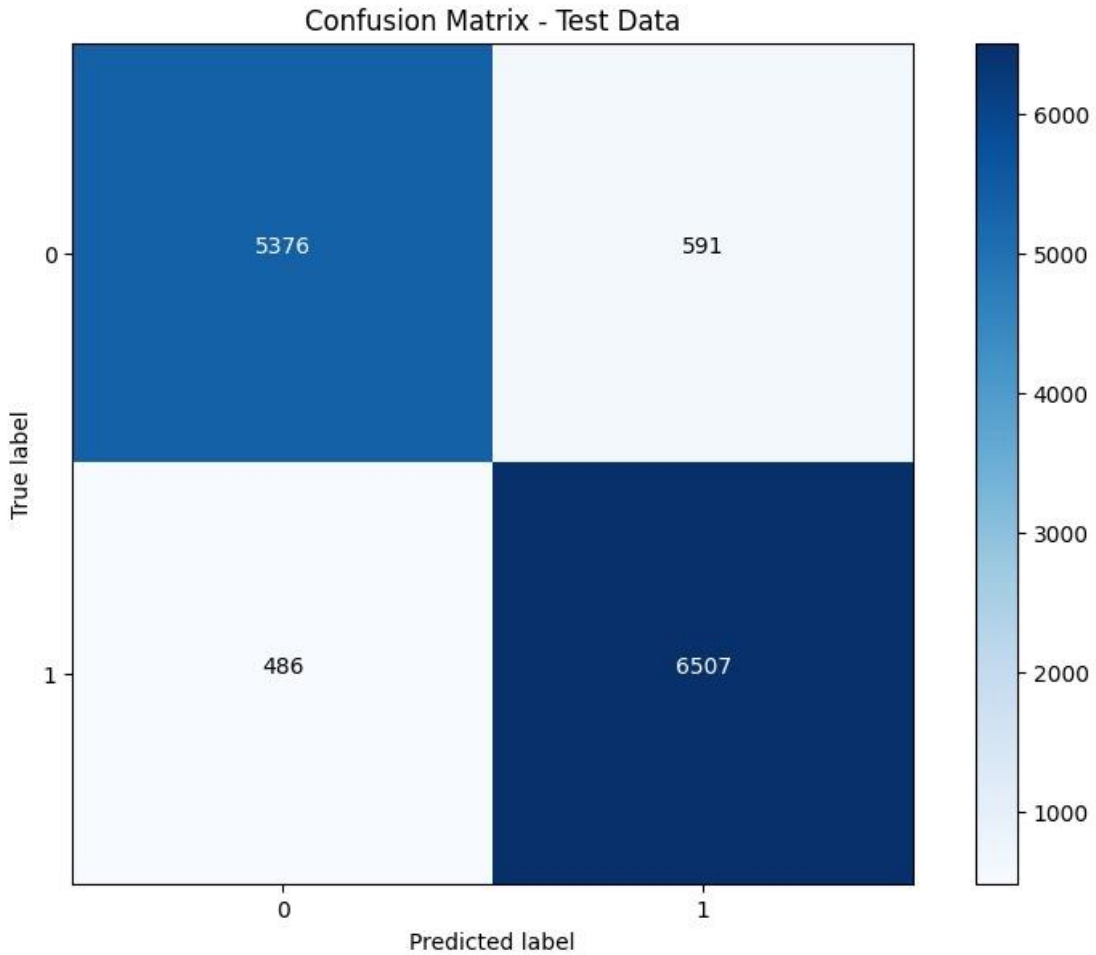
4. Null Handling – Nulls exists in every column except 'ID' and 'label'.

```
ID                      0
worked_in_the_past   1644
age group            2172
disability           4442
is dev               1928
education            2822
sex                  2995
mental_issues        2243
years_of_experience  2400
A                    1083
B                    3364
C                    2829
D                    2411
country              1980
prev_salary          2531
stack_experience    13090
label                   0
```

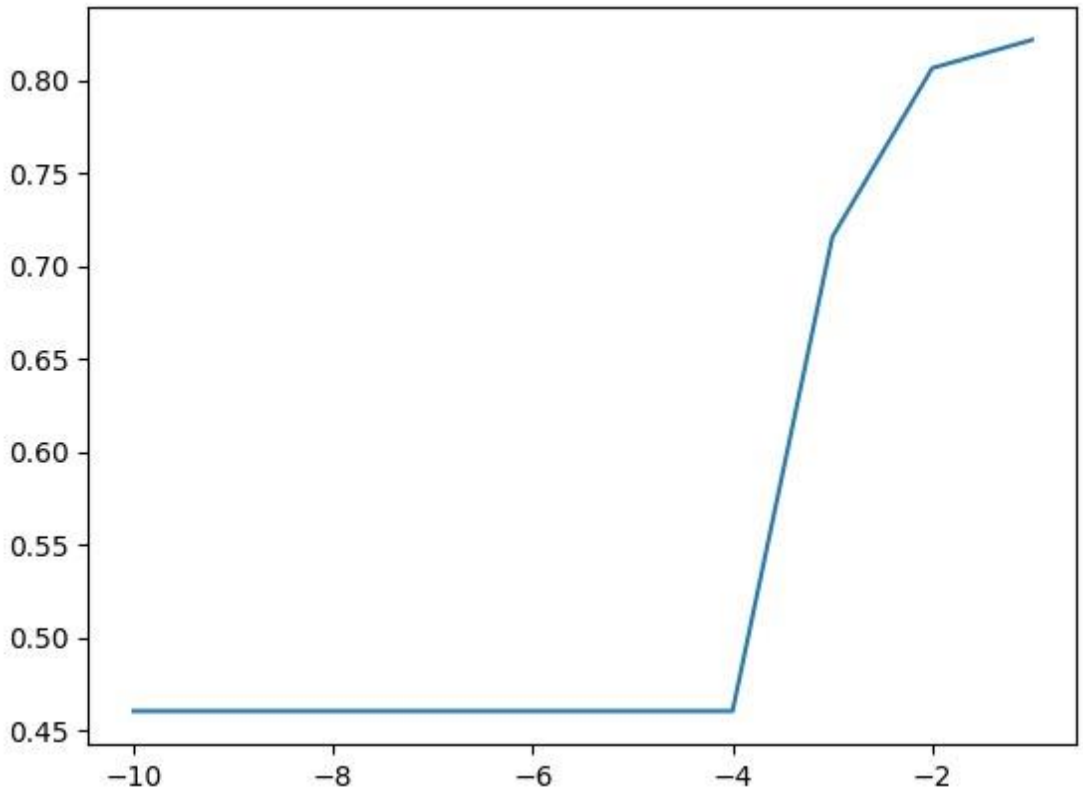
5. Feature Importance -



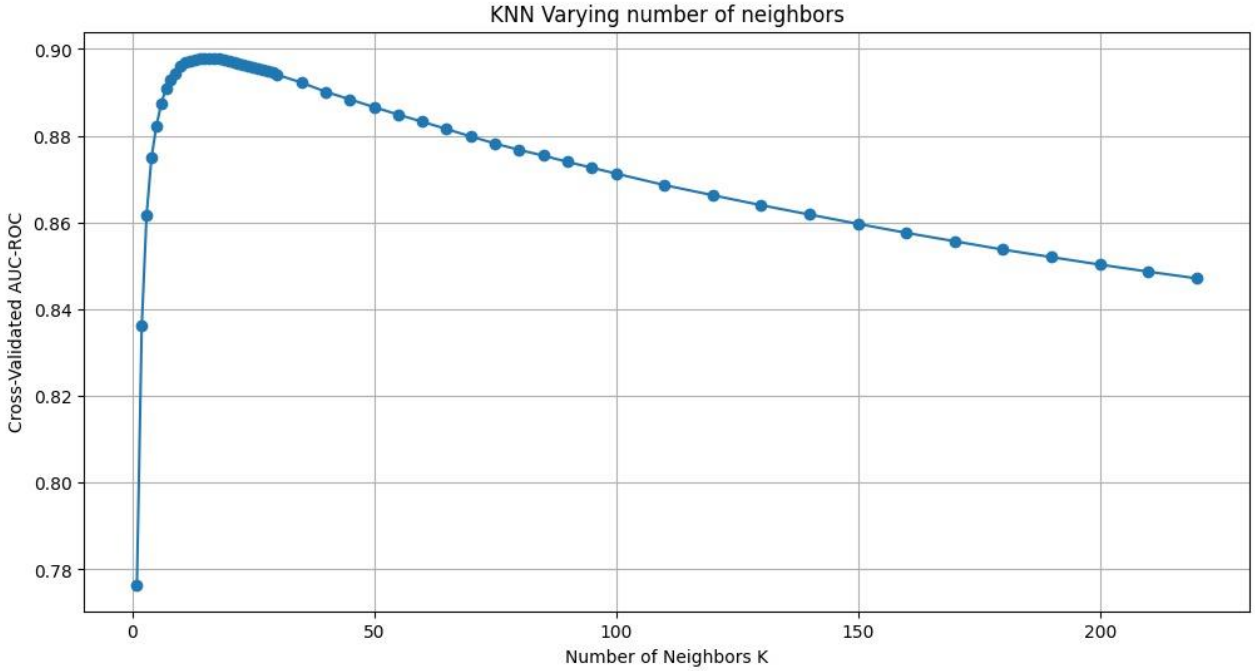
6. Confusion Matrix –



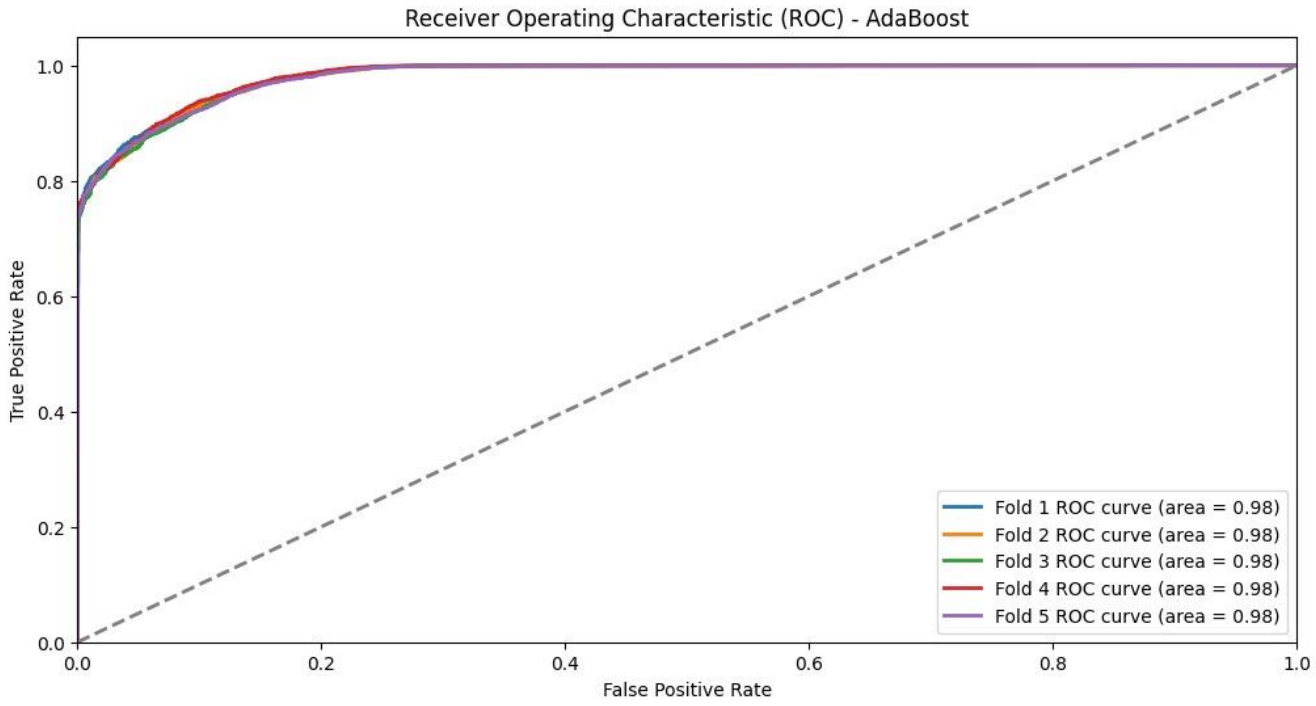
7. Calculation of optimal C – Logistic Regression



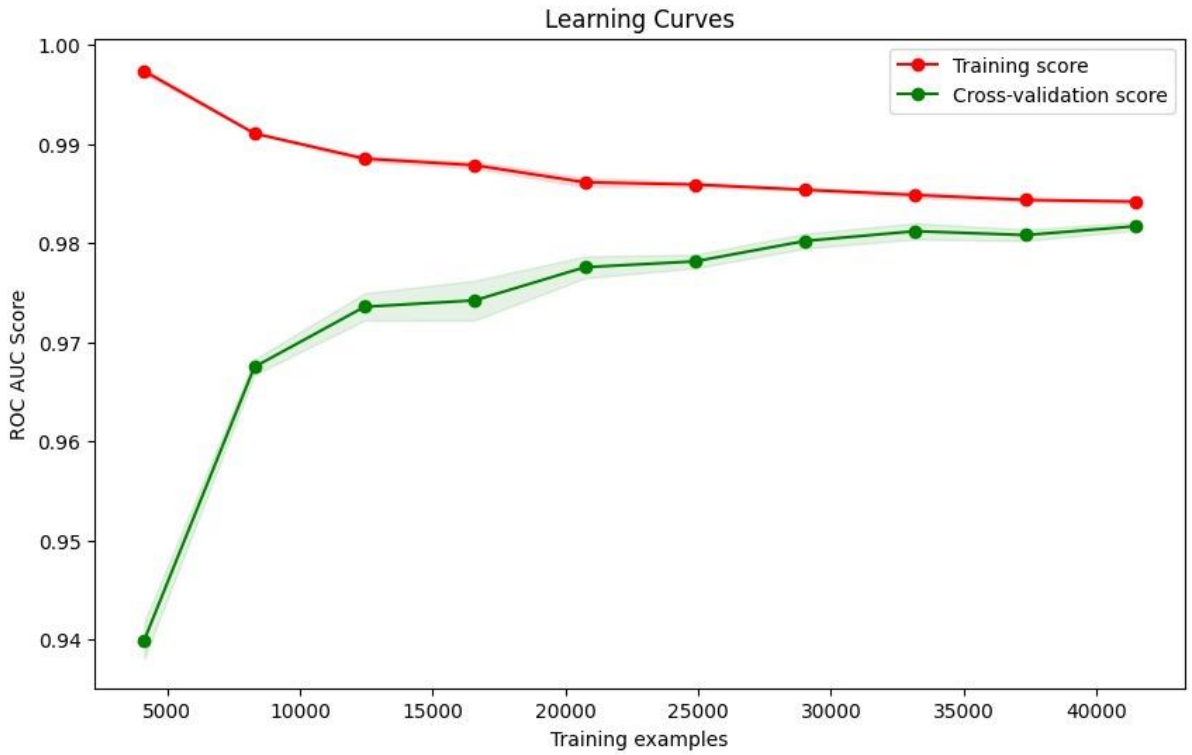
8. Calculation of optimal K - KNN



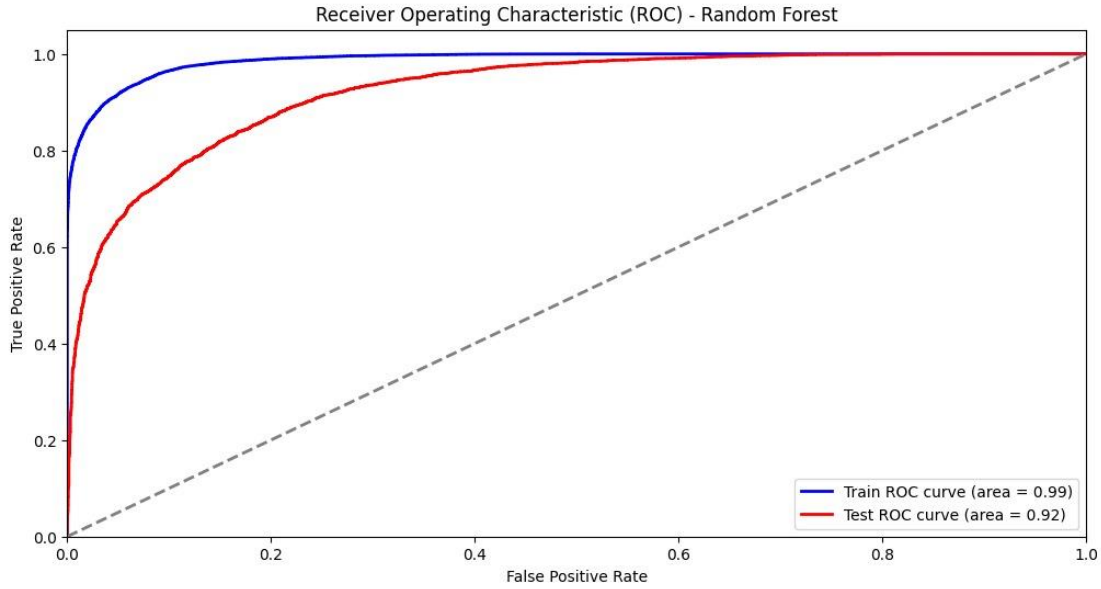
9. K-Fold Cross Validation – AdaBoost



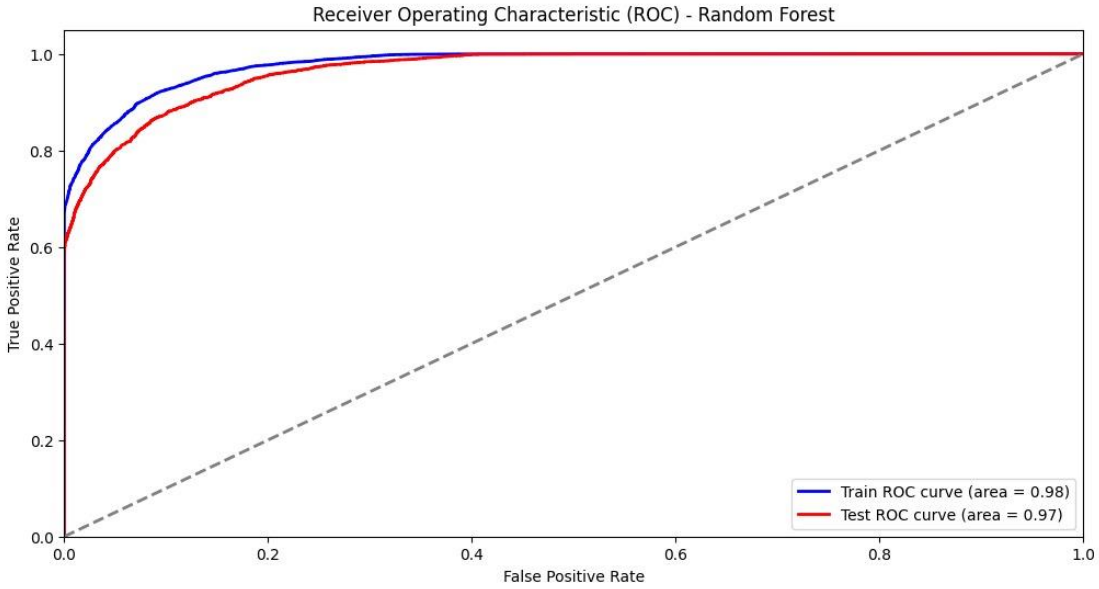
10. Learning Curve – Check for overfitting in AdaBoost model.



11.1 AUC scores – Random Forest with PCA -



11.2 AUC scores – Random Forest without PCA -



Appendix 12- AdaBoost hyperparameters chosen as learned in class:

ccp_alpha: Set to 0.0 to avoid pruning the decision tree, ensuring it fully grows according to the specified depth and other criteria.
class_weight: Set to None to treat all classes equally, as class imbalances were not a significant issue in this dataset.
criterion: Set to 'gini' to measure the quality of splits using the Gini impurity, which is computationally efficient and effective for this task.
max_depth: Limited to 2 to create shallow trees that reduce overfitting, which is suitable

given the boosting nature of AdaBoost.
max_features: Set to None to allow each tree in the ensemble to consider all features, maximizing the diversity among the trees.
max_leaf_nodes: Set to None to avoid limiting the number of leaf nodes, allowing trees to fully develop according to other constraints like depth.
min_impurity_decrease: Set to 0.0 to ensure that nodes are split as long as they improve the model, regardless of how small the improvement might be.
min_samples_leaf: Set to 3 to prevent trees from becoming too specific by ensuring each leaf has at least three
samples.
min_samples_split: Set to 2 to allow nodes to split as long as they have at least two samples, providing flexibility in tree growth.
n_estimators: Increased to 200 to boost performance by using more weak learners, which enhances the ensemble's overall predictive accuracy.
random_state: Set to 42 to ensure consistent results across different runs by controlling the randomness in the model's learning process.

Appendix 13 – Random Forest hyperparameters chosen as learned in class:
criterion: Changed from 'gini' to 'entropy' for better results with complex data relationships.
max_depth: Changed from None to 15 to prevent overfitting and balance generalization.
max_features: Changed from 'auto' to 'sqrt' to ensure consistent feature selection and reduce overfitting.
min_samples_leaf: No change; kept at 1 as increasing it did not improve performance.
min_samples_split: Changed from 2 to 5 to avoid overly specific patterns and reduce overfitting.
n_estimators: Changed from 100 to 150 to enhance model robustness and generalization.

Appendix 14 - Contribution Division

Data Exploration: Elad
Data Preprocessing: Together
Model Test Run and Evaluation: KNN and Logistic Regression: Together , Random Forest: Elad, AdaBoost: Ron
Pipeline Writing: Together
Report Writing: Together