North South University

Department of Electrical & Computer Engineering

Project Report

Course Code: CSE 445

Course Title: Machine Learning

Section: 04

Project Title:

Boston Housing Price Prediction using Machine Learning.

Date of Submission: July 27, 2025

Submitted by:

01. Joy Kumar Ghosh – 2211424 0 42

Submitted to:

Course Instructor, Dr. Mohammad Abdul Qayum (MAQm)

Abstract:

Introduction:

Data Preparation:

Boston Housing data is one of the most popular educational datasets. I downloaded the datasets from the provided link and ran the project on my local machine. First of all, let's see the dataset's description.

Table 1: Datasets Overview

Column	Null Value	Non-Null Value	Description
CRIM	20	486	per capita crime rate by town
ZN	20	486	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	20	486	proportion of non-retail business acres per town
CHAS	20	486	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	0	506	nitric oxides concentration (parts per 10 million)
RM	0	506	average number of rooms per dwelling
AGE	20	486	proportion of owner-occupied units built prior to 1940
DIS	0	506	weighted distances to five Boston employment centres
RAD	0	506	index of accessibility to radial highways
TAX	0	506	full-value property-tax rate per $10,000
PTRATIO	0	506	pupil-teacher ratio by town
B	0	506	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT	20	486	% lower status of the population
MEDV (Target)	0	506	Median value of owner-occupied homes in $1000's

The datasets overview shows that among the 506 samples, columns CRIM, ZN, INDUS, CHAS, AGE, and LSTAT contain 20 null values each. Although I checked that, the number of null values is the same but exist in different samples. As we have only 506 samples, we can't drop a sample containing a null value. Therefore, we need to fill these nulls using some appropriate function. Among these, column CHAS is not a continuous value, so I filled this column with the mode, and the rest with the mean. Furthermore, I also checked for any duplicate samples and found none.

As we need to implement a classification problem from the MEDV column, I also decided to use these classes for the stratified split. I added the column named MEDV_Class based on the quantile. I decided to have the quantile 0.25 as Low, 0.25 to 0.75 as Medium, and 0.75 as High. Finally, the dataset is ready for visualization and training.

Data Visualization:

To visualize the data, first, I have plotted a histogram for the entire dataset to see the distribution of each column. But, I didn't get enough insight from there, so I decided to plot a scatter matrix.

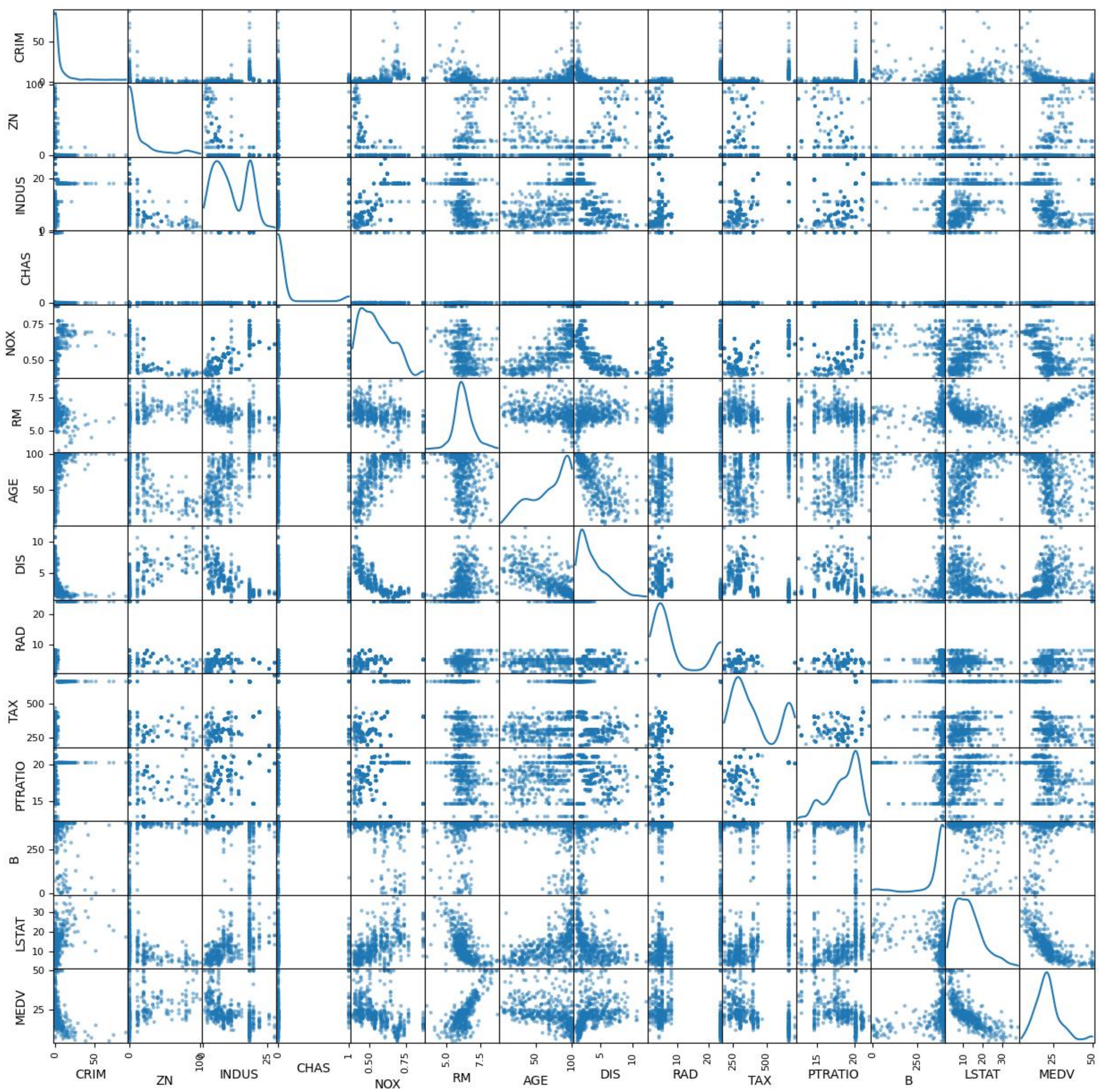

Figure 1: Scatter Matrix for entire Datasets

From the scatter matrix, we can see that RM shows a positive correlation with MEDV; on the other hand, LSTAT shows a negative correlation with MEDV. Other features do not have a reasonable correlation with the MEDV. I also checked the numerical value of the correlation with the target column.

Table 2: Correlation of each features with target column MEDV

Features	**Correlation**
MEDV (Target)	1.00
RM	0.70
ZN	0.37
B	0.33
DIS	0.25
CHAS	0.18
CRIM	-0.38
AGE	-0.38
RAD	-0.38
NOX	-0.43
TAX	-0.47
INDUS	-0.48
PTRATIO	-0.51
LSTAT	-0.72

The correlation table shows that RM has the highest positive correlation, and LSTAT has the lowest negative correlation. We can also see that CRIM, AGE, and RAD correlate very similarly to the MEDV. That means they will contribute equally, so we can keep one and discard the other two features, and the effect will be the same.

Later, I tried to find a line that fit the highest and the lowest contributing features using “regplot” from the “seaborn” package.

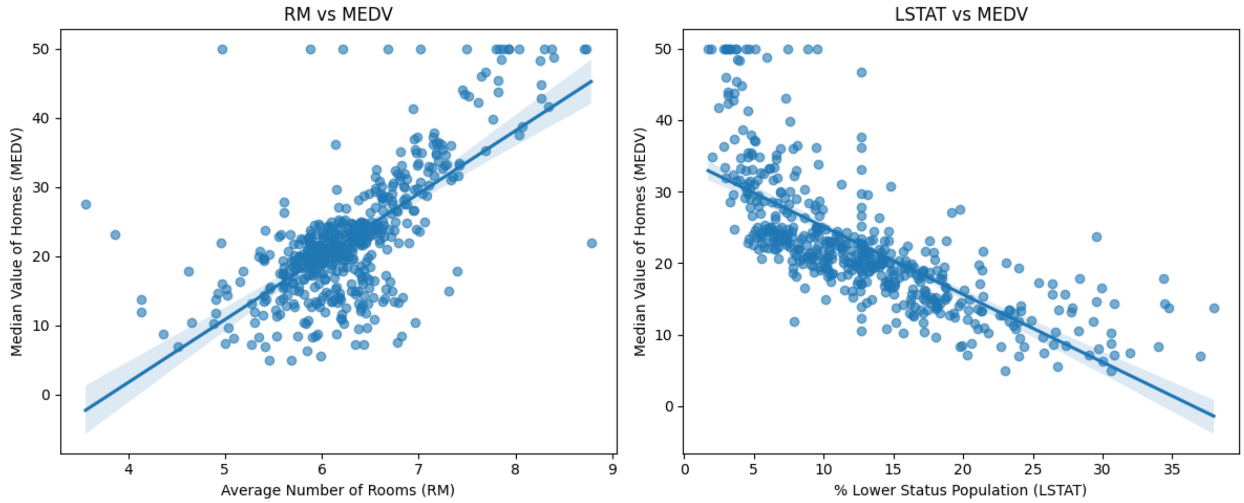

Figure 2: Regression plot for two highest contributing features

The regplot shows too many points that are far from the fit line. That means the error will be so high if we fit it as it is with linear regression. A decision tree or a random forest algorithm might reduce the error. Furthermore, I also tried to visualize whether the class distribution is linearly separable.

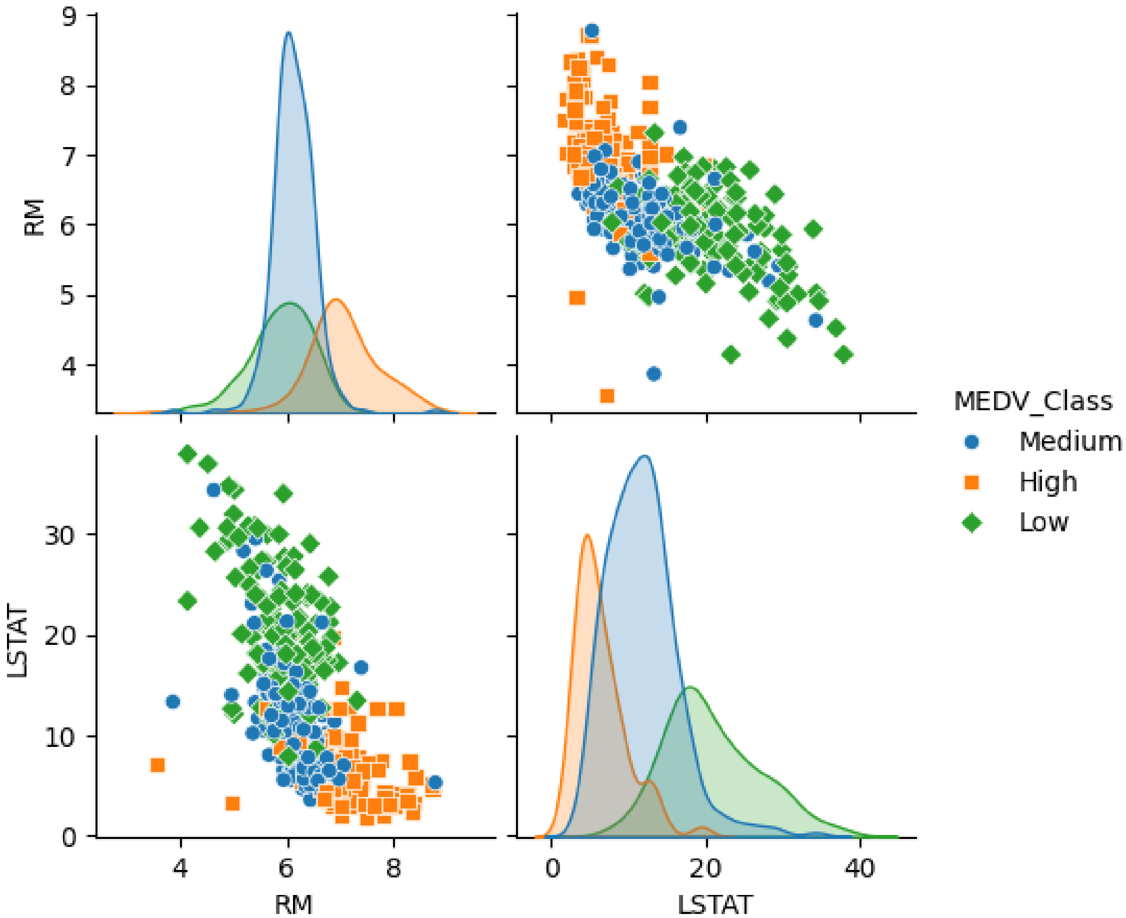

Figure 3: Pair plot to visualize the class distribution

It is a disaster; they are not linearly separable, always overlapping.

Data Split & Scaling:

For splitting the data, I used stratified splitting based on the MEDV_Class. Stratified split is a good practice, as it distributes the data rationally based on the class. So, I use it for my assignment. I split the data 80% for the training and 20% for the test. After the split, I used three different types of scaling to analyze which scaler is better for this data. Initially, I observed that the difference between the 75th % and the maximum value is too significant for CRIM, ZN, INDUS, and DIS features. So, first, I decided to use the minmax scaler. Later, I also tried the standard and robust scaler. An analysis of different scales is shown in the result analysis section.

Table 3: Stratified Train Test Split

Class	Original Sample	Train Sample	Test Sample
Low	127	101	26
Medium	255	204	51

High	124	99	25

Training Methodology:

After all the preprocessing, I started to train some machine learning algorithms. First, I trained four regressor models, such as Linear Regressor, Decision Tree Regressor, Random Forest Regressor, and Support Vector Regressor, with their default parameters. Later, I started to tune their parameters using Grid Search CV. For the Grid search, I used cross-validation (CV) value as 5, and used the 'neg_mean_squared_error' scoring. Also, I used n_jobs = -1 to maximize the core use for faster run. And use different grid parameters for other models. I kept these parameters as reasonable and minimal to overcome training time constraints.

Table 4: Grid Parameters for Grid Search

Model Name	Grid Parameters
Linear Regressor	'fit_intercept': [True, False], 'copy_X': [True, False], 'positive': [True, False]
Decision Tree Regressor	'max_depth': [None, 5, 10, 15, 25], 'min_samples_split': [2, 5, 10, 20], 'min_samples_leaf': [1, 2, 4, 8], 'max_features': ['sqrt', 'log2', None, 0.5], 'criterion': ['squared_error', 'friedman_mse'], 'min_impurity_decrease': [0.0, 0.01, 0.02], 'ccp_alpha': [0.0, 0.01, 0.02]
Support Vector Regressor	'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.001, 0.01, 0.1], 'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], 'epsilon': [0.01, 0.1, 0.2], 'degree': [2, 3, 4, 5], 'coef0': [0.0, 0.1, 1.0], 'shrinking': [True, False], 'tol': [1e-3, 1e-4]
Random Forest Regressor	'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5, 10, 15, 20], 'min_samples_leaf': [1, 2, 4, 6], 'max_features': ['sqrt', 'log2', None], 'bootstrap': [True], 'max_samples': [None, 0.8, 0.9],

	'min_impurity_decrease': [0.0, 0.01, 0.02]

I tried to use a randomized search to tune their parameters again for the assignment task. As a randomized search doesn't take much time or doesn't form all possible combinations, it randomly chooses a combination and trains the model. Here, I limited the model number to 200. And I used a robust parameter list.

Table 5: Grid Parameters for Randomized Grid Search

Model Name	**Robust Grid Parameters**
Linear Regressor	'fit_intercept': [True, False], 'copy_X': [True, False], 'positive': [True, False]
Decision Tree Regressor	'max_depth': [None, 3, 5, 8, 10, 15, 20, 25, 30, 35], 'min_samples_split': [2, 3, 5, 8, 10, 15, 20, 25, 30], 'min_samples_leaf': [1, 2, 3, 4, 6, 8, 10, 12, 15], 'max_features': ['sqrt', 'log2', None, 0.3, 0.5, 0.7, 0.8, 1.0], 'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'], 'min_impurity_decrease': [0.0, 0.005, 0.01, 0.015, 0.02, 0.03, 0.05], 'ccp_alpha': [0.0, 0.005, 0.01, 0.015, 0.02, 0.03, 0.05, 0.1], 'splitter': ['best', 'random'], 'max_leaf_nodes': [None, 10, 20, 30, 50, 100, 150, 200, 300], 'min_weight_fraction_leaf': [0.0, 0.01, 0.02, 0.05, 0.1]
Support Vector Regressor	'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.001, 0.01, 0.1], 'kernel': ['linear', 'rbf', 'poly'], 'epsilon': [0.01, 0.1, 0.2, 0.5], 'degree': [2, 3, 4], 'coef0': [0.0, 0.1, 1.0], 'shrinking': [True, False], 'tol': [1e-3, 1e-4], 'cache_size': [200], 'max_iter': [10000, 50000, 100000]
Random Forest Regressor	'n_estimators': [50, 100, 150, 200, 300, 400, 500], 'max_depth': [None, 5, 10, 15, 20, 25, 30, 40], 'min_samples_split': [2, 3, 5, 8, 10, 15, 20], 'min_samples_leaf': [1, 2, 3, 4, 5, 6, 8, 10], 'max_features': ['sqrt', 'log2', None, 0.3, 0.5, 0.7, 0.8], 'bootstrap': [True], 'max_samples': [None, 0.7, 0.8, 0.9, 0.95],

	'min_impurity_decrease': [0.0, 0.001, 0.005, 0.01, 0.015, 0.02], 'max_leaf_nodes': [None, 20, 30, 50, 100, 200, 300], 'min_weight_fraction_leaf': [0.0, 0.01, 0.02, 0.05], 'ccp_alpha': [0.0, 0.001, 0.005, 0.01, 0.015], 'criterion': ['squared_error', 'absolute_error', 'friedman_mse']

And that's the end of the training of the regressor model. After that, I started to train a classification model to predict the MEDV_Class. First, I trained three base models with their default parameters: Logistic Regression, Random Forest Classifier, and Support Vector Classifier. I also tried to make this model ensemble with the Bagging Classifier. I limited the number of ensemble models to 50 for the Bagging Classifier.

Furthermore, I also trained Gradient Boosting, AdaBoost, and LightGBM models, and used default parameters as usual. Later, I trained the stacking classifier, where I chose Logistic Regression, Random Forest Classifier, Support Vector Classifier, Gradient Boosting Classifier, Ada Boost Classifier, and LightGBM Classifier as the base models, and again Logistic Regression for the final classifier. The result analysis section gives all the performance metrics analysis, ROC, and AUC.

Result Analysis:

As mentioned above, I trained each model for different scalers. Here is the table of each model's performance:

Table 6: All Performance Metrics for Regression models

Model Name	**Parameters**	**Scaler**	**MSE**	**MAE**	**R2**
Linear Regression	Default	MinMax	20.78	3.45	0.72
		Standard	20.78	3.45	0.72
		Robust	20.78	3.45	0.72
	Grid Search	MinMax	20.78	3.45	0.72
		Standard	20.78	3.45	0072
		Robust	20.78	3.45	0.72
	Randomized Search	MinMax	20.78	3.45	0.72
		Standard	20.78	3.45	0.72
		Robust	20.78	3.45	0.72
Decision Tree Regressor	Default	MinMax	15.18	2.83	0.80
		Standard	16.71	2.98	0.78
		Robust	29.76	3.08	0.61
	Grid Search	MinMax	17.15	3.14	0.77

		Standard	17.15	3.14	0.77
		Robust	17.15	3.14	0.77
	Randomized Search	MinMax	15.96	2.92	0.79
		Standard	15.99	2.93	0.79
		Robust	15.96	2.92	0.79
Support Vector Regressor	Default	MinMax	26.07	3.26	0.65
		Standard	22.41	2.95	0.70
		Robust	24.36	3.17	0.68
	Grid Search	MinMax	7.60	2.00	0.90
		Standard	8.74	2.18	0.88
		Robust	9.81	2.30	0.87
	Randomized Search	MinMax	7.89	2.03	0.90
		Standard	8.74	2.18	0.89
		Robust	9.81	2.30	0.87
Random Forest Regressor	Default	MinMax	8.85	2.16	0.88
		Standard	8.89	2.16	0.88
		Robust	8.89	2.16	0.88
	Grid Search	MinMax	7.03	1.94	0.91
		Standard	7.07	1.95	0.91
		Robust	7.07	1.95	0.91
	Randomized Search	MinMax	8.08	2.21	0.89
		Standard	8.09	2.21	0.89
		Robust	8.08	2.21	0.89

From the table, we can see that the Random Forest Regressor outperforms all other models with the help of grid search and MinMax scaler. Its R2 score is 0.91, which is higher than others. And the closest competitor is Support Vector Regressor with grid search, which scored a 0.90 in R2. The scaler doesn't affect these models much from these analyses, but the computation time varies depending on the scaler. During the training process, I found that RobustScaler takes much more time than other scalers.

For the Linear Regression model, scaler, grid search, and randomized search do not affect the outcome. However, for the other model, we can see that grid search improved the performance. And randomized grid search is all about luck. Overall, the analysis shows that the MinMax scaler is much better than others; it improved the performance and reduced computation time.

Let's see the performance metrics of the classification problem:

Table 7: All Performance Metrics for Classification models

Model Name	Scaler	Accuracy	Precision		Recall		F1 - Score	
			Macro	Weighted	Macro	Weighted	Macro	Weighted
Logistic Regression	MinMax	0.74	0.74	0.74	0.71	0.74	0.72	0.73
	Standard	0.78	0.79	0.78	0.78	0.78	0.78	0.78
	Robust	0.80	0.81	0.80	0.79	0.80	0.80	0.80
Logistic Regression Bagging	MinMax	0.75	0.77	0.75	0.71	0.75	0.73	0.74
	Standard	0.77	0.79	0.78	0.75	0.77	0.76	0.77
	Robust	0.77	0.79	0.78	0.75	0.77	0.76	0.77
Support Vector Classifier	MinMax	0.74	0.74	0.74	0.72	0.74	0.73	0.73
	Standard	0.81	0.82	0.81	0.80	0.81	0.81	0.81
	Robust	0.75	0.78	0.77	0.73	0.75	0.75	0.75
Support Vector Classifier Bagging	MinMax	0.73	0.74	0.73	0.70	0.73	0.71	0.72
	Standard	0.75	0.78	0.76	0.72	0.75	0.74	0.75
	Robust	0.75	0.76	0.75	0.72	0.75	0.74	0.74
Random Forest Classifier	MinMax	0.86	0.87	0.87	0.85	0.86	0.86	0.86
	Standard	0.86	0.87	0.87	0.85	0.86	0.86	0.86
	Robust	0.87	0.89	0.88	0.85	0.87	0.87	0.87
Random Forest Classifier Bagging	MinMax	0.84	0.85	0.84	0.83	0.84	0.84	0.84
	Standard	0.84	0.85	0.84	0.83	0.84	0.84	0.84
	Robust	0.84	0.85	0.84	0.83	0.84	0.84	0.84
Gradient Boosting Classifier	MinMax	0.88	0.90	0.89	0.87	0.88	0.88	0.88
	Standard	0.88	0.90	0.89	0.87	0.88	0.88	0.88
	Robust	0.88	0.90	0.89	0.87	0.88	0.88	0.88
LightGBM Classifier	MinMax	0.85	0.86	0.86	0.84	0.85	0.85	0.85
	Standard	0.87	0.89	0.88	0.86	0.87	0.87	0.87
	Robust	0.84	0.86	0.85	0.83	0.84	0.84	0.84
AdaBoost Classifier	MinMax	0.78	0.78	0.78	0.79	0.78	0.78	0.78
	Standard	0.78	0.78	0.78	0.79	0.78	0.78	0.78
	Robust	0.78	0.78	0.78	0.79	0.78	0.78	0.78
Stacking	MinMax	0.87	0.89	0.88	0.85	0.87	0.87	0.87
	Standard	0.87	0.89	0.88	0.85	0.87	0.87	0.87
	Robust	0.88	0.90	0.89	0.87	0.88	0.88	0.88

Conclusion:

References: