Supervised Learning for Identifying Disadvantaged Communities and Rice classifications

*Zhi Li (zli3037@gatech.edu)

Github repo: https://github.com/ZhiLi51/disadvantaged_community_identification_model

Abstract: **This report employs supervised learning techniques to address two classification problems. Firstly, it aims to identify disadvantaged communities utilizing US census tract data, encompassing demographic indicators such as race, household income, and energy burden. The cleaned dataset comprises over 1200 samples with 16 features. Data preprocessing steps including normalization and Principal Component Analysis (PCA) were undertaken, followed by the evaluation of six types of supervised learning algorithms to compare their performance. Secondly, the author uses rice data set from UCI to classify two types of rice. The dataset has 3810 samples and 7 features. Implementation of six algorithms facilitates a comprehensive comparison, considering metrics and computational complexity.**

I. Introduction

The primary objective of this study is to leverage supervised learning methodologies for the identification of disadvantaged communities, drawing insights from US census tract data. These communities often confront a myriad of socio-economic adversities, which manifest through demographic indicators such as race, household income, and energy burden. With a robust dataset comprising over 1200 samples and 16 features, our analysis seeks to uncover discernible patterns and attributes characteristic of disadvantaged areas. To optimize the efficacy of our analysis, we employ rigorous data preprocessing techniques, including normalization and Principal Component Analysis (PCA). Subsequently, we rigorously evaluate the performance of six distinct supervised learning algorithms to gauge their effectiveness in accurately identifying disadvantaged communities.

In the second task, we turn our attention to a separate classification challenge involving the classification of two types of rice utilizing a dataset sourced from the UCI Machine Learning Repository. With a substantial dataset comprising 3810 samples and 7 features, this task presents a distinct yet equally significant classification endeavor. Leveraging the rich dataset, our objective is to develop a robust classification model capable of accurately distinguishing between the two types of rice based on their morphological features. Similar to the first task, we employ advanced data preprocessing techniques and supervised learning algorithms to extract meaningful insights and achieve optimal classification performance. Through this dual-task approach, we aim to showcase the versatility and effectiveness of supervised learning methodologies in addressing diverse classification problems within the realm of socio-economic disparities and agricultural sciences.

II. Methdology

A. Dataset:

The methodology begins with the acquisition and understanding of two distinct datasets, each tailored to its respective classification task. The first dataset comprises US census tract data, serving as the foundation for identifying disadvantaged communities. This dataset encapsulates various socio-economic indicators, including race, household income, and energy burden, with over 1200 samples and 16 features. Meanwhile, the second dataset sourced from the UCI Machine Learning Repository focuses on rice classification, offering 3810 samples with 7 morphological features.

B. Data preprocess and EDA:

Prior to model development, comprehensive data preprocessing and exploratory data analysis (EDA) are conducted to ensure data quality and glean insights. This phase encompasses tasks such as handling missing values, encoding categorical variables, scaling numerical features, and conducting exploratory analysis to understand the distribution and relationships within the datasets. Techniques such as normalization and Principal Component Analysis (PCA) are applied to enhance the efficacy of the analysis.

C. Training Procedure

The training procedure follows a systematic approach aimed at optimizing model performance and generalization capability.

1) Training Curve and Validation Curve:

The training process begins with the analysis of training and validation curves. These curves provide insights into the model's learning progress and its ability to generalize to unseen data. Discrepancies between the two curves can indicate issues such as overfitting or underfitting.

2) Cross Validation:

Cross-validation techniques, such as k-fold cross-validation, are employed to assess the model's performance across multiple subsets of the data. This helps ensure robustness and reliability in estimating the model's performance on unseen data.

3) Hyperparameter Tuning:

Hyperparameters, which control the behavior of the learning algorithm, are fine-tuned to optimize model performance. Techniques such as grid search are utilized to systematically explore the hyperparameter space and identify the optimal configuration.

4) Evaluation Metrics:

The trained models are evaluated using a range of metrics, including accuracy, F1 score, and area under the ROC curve (AUC). These metrics provide comprehensive insights into the model's classification performance, capturing aspects such as overall accuracy, balance between precision and recall, and discriminative ability.

5) Iteration:

The training procedure involves iterative refinement, where models are trained, evaluated, and refined iteratively.

This iterative process allows for continuous improvement and optimization of model performance.

By following this training procedure, the author ensure the development of robust and effective classification models capable of addressing the classification challenges posed by the datasets.

D. Algorithms:

The methodology employs a suite of supervised learning algorithms tailored to each classification task. These algorithms include Logistic Regression (LR), Decision Trees (DT), Adaboost, Neural Networks (NNs), Support Vector Machines (SVM), and k-Nearest Neighbors (KNN). Each algorithm offers unique strengths and capabilities, enabling comprehensive exploration and comparison of their performance in addressing the classification challenges posed by the datasets. Through rigorous evaluation and comparison, the most effective algorithms are identified to advance the classification objectives.

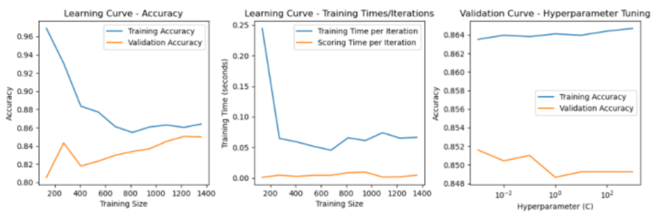
III. TASK I

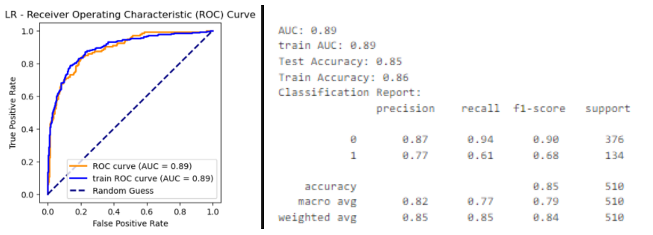
A. Logistic Regression

Logistic Regression (LR) serves as the initial model for addressing the task of identifying disadvantaged communities. LR is a widely-used algorithm for binary classification tasks, leveraging a logistic function to estimate the probability that a given sample belongs to a particular class. In the context of our analysis, LR is well-suited for discerning patterns and characteristics indicative of disadvantaged areas based on demographic indicators such as race, household income, and energy burden.

Hyperparameters play a crucial role in fine-tuning the performance of LR. The primary hyperparameter for LR is the regularization parameter, denoted as C. This parameter controls the regularization strength, with smaller values of C corresponding to stronger regularization and vice versa. By tuning the value of C, we can optimize the balance between model complexity and overfitting, ultimately enhancing the model's generalization capability.

To determine the optimal value of the regularization parameter C, the author employs grid search, a technique for systematically searching through a predefined range of hyperparameter values. Specifically, we define a grid of C values and evaluate the performance of the LR model using cross-validation. The grid search algorithm exhaustively evaluates each combination of hyperparameters to identify the configuration that maximizes performance.





```
AUC: 0.89
train AUC: 0.89
Test Accuracy: 0.85
Train Accuracy: 0.86
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.90       376
           1       0.77      0.61      0.68       134

    accuracy                           0.85       510
   macro avg       0.82      0.77      0.79       510
weighted avg       0.85      0.85      0.84       510
```

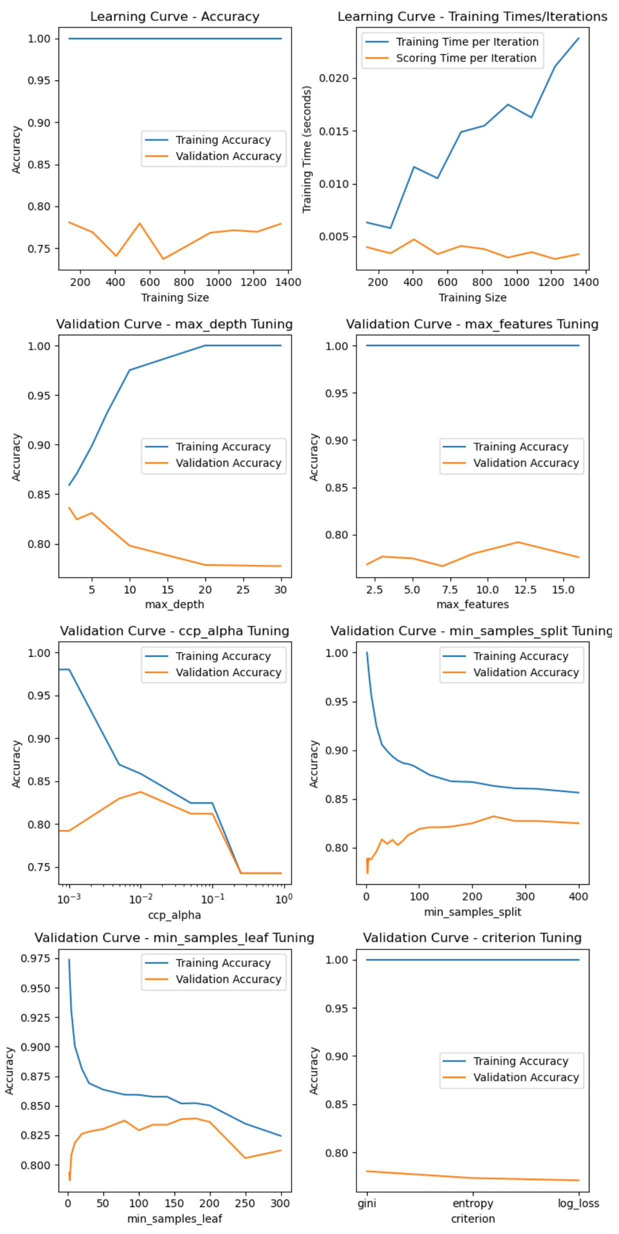
Fig.1 Performance of Logistic Regression (Best Hyperparameters: {'C': 0.01, 'max_iter': 1000})

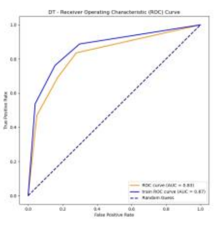
B. Decicion Tree

Decision Tree (DT) is a powerful and interpretable machine learning algorithm commonly used for classification tasks. It works by recursively partitioning the feature space into subsets based on the values of input features, ultimately forming a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents the predicted class. This hierarchical structure allows for intuitive interpretation of the decision-making process, making DT particularly useful for understanding the factors influencing classification outcomes.

Hyperparameters in DT control various aspects of the tree's construction and pruning, influencing its complexity and generalization capability. Key hyperparameters include the criterion for splitting nodes (e.g., 'gini' for Gini impurity or 'entropy' for information gain), the splitting strategy ('best' or 'random'), the maximum depth of the tree, minimum number of samples required to split an internal node (min_samples_split), and minimum number of samples required to be at a leaf node (min_samples_leaf). Additionally, hyperparameters such as ccp_alpha control the cost-complexity pruning, which helps prevent overfitting by penalizing overly complex trees.

To determine the optimal combination of hyperparameters, we begin by exploring the training and validation curves across a range of parameter values. This analysis helps identify a good range of values for each hyperparameter, balancing model complexity and performance. Subsequently, we employ grid search, a technique for exhaustively searching through a predefined hyperparameter grid, to fine-tune the model. The hyperparameter grid encompasses various combinations of criteria, splitters, maximum depth, minimum samples for splitting and leaf nodes, ccp_alpha, and maximum features. By conducting grid search, we aim to identify the best combination of hyperparameters that maximizes classification performance. Based on our grid search results, the optimal hyperparameters for the Decision Tree model are determined as {'ccp_alpha': 0.01, 'criterion': 'log_loss', 'max_depth': 7, 'max_features': 9, 'min_samples_leaf': 160, 'min_samples_split': 250, 'splitter': 'best'}. These hyperparameters are then used to train the final Decision Tree model for classification tasks.





```
AUC: 0.83
train AUC: 0.87
Test Accuracy: 0.82
Train Accuracy: 0.85
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89       376
           1       0.77      0.47      0.58       134

    accuracy                           0.82       510
   macro avg       0.80      0.71      0.74       510
weighted avg       0.82      0.82      0.81       510
```

Fig.2 Performance of Decision Tree (Best Hyperparameters: {'ccp_alpha': 0.01, 'criterion': 'log_loss', 'max_depth': 7, 'max_features': 9, 'min_samples_leaf': 160, 'min_samples_split': 250, 'splitter': 'best'})

C. *AdaBoost*

Adaboost, short for Adaptive Boosting, is an ensemble learning method that constructs a strong classifier by combining multiple weak learners in a sequential manner. Unlike Random Forest (RF), which employs bagging and bootstrap sampling, Adaboost focuses on boosting weak learners.

The primary hyperparameters of Adaboost include:

base_estimator: The weak learner used as the base estimator, often a Decision Tree with limited depth to avoid overfitting.

n_estimators: The number of weak learners (e.g., decision trees) to be sequentially trained.

learning_rate: The contribution of each weak learner to the final ensemble, influencing the weight assigned to misclassified samples in subsequent iterations.

To identify the optimal combination of hyperparameters, we begin by analyzing the training and validation curves across various parameter values. This helps in determining a suitable range for each hyperparameter, balancing model complexity and performance. Subsequently, grid search is employed to fine-tune the model, with a predefined hyperparameter grid covering combinations of base estimator criteria, splitter strategies, maximum depth, minimum samples for leaf nodes, and maximum features, along with learning rate and the number of estimators. The grid search aims to find the optimal hyperparameter configuration that maximizes classification performance. Based on the results of the grid search, the best hyperparameters for the Adaboost model are determined as {'ccp_alpha': 0.01, 'criterion': 'log_loss', 'max_depth': 7, 'max_features': 9, 'min_samples_leaf': 160, 'min_samples_split': 250, 'splitter': 'best'}. These hyperparameters are then used to train the final Adaboost model for classification tasks.

D. *Neural Networks*

Neural Networks (NN) represent a powerful class of machine learning models capable of learning complex patterns and relationships within data. In this analysis, both scikit-learn and PyTorch libraries are utilized to implement neural network architectures. Furthermore, TensorBoard is employed to visualize and monitor the training and validation processes, providing insights into the model's performance and convergence behavior.

The hyperparameters of Neural Networks play a crucial role in determining the architecture and training behavior of the model. The primary hyperparameters include:

hidden_layer_sizes: The number of neurons in each hidden layer, influencing the model's capacity to capture complex patterns.

activation: The activation function applied to neurons within the network, such as 'logistic' (sigmoid), and 'tanh' (hyperbolic tangent).

solver: The optimization algorithm used to train the network, including 'lbfgs', 'sgd' (stochastic gradient descent), and 'adam' (adaptive moment estimation).

alpha: The L2 regularization parameter, controlling the model's complexity and helping prevent overfitting.

learning_rate: The learning rate schedule, with 'adaptive' indicating a dynamically adjusting learning rate based on validation performance.

learning_rate_init: The initial learning rate, influencing the magnitude of parameter updates during training.

batch_size: The number of samples processed per gradient update, affecting the speed and stability of training.

To identify the optimal combination of hyperparameters, we analyze training and validation curves across various parameter values to identify a suitable range. Subsequently, grid search is employed to fine-tune the model, exploring combinations of hyperparameters defined in the grid. Based on the results of the grid search, the best hyperparameters for the Neural Network model are determined, optimizing classification performance for the given tasks. In this case, the best hyperparameters include {'activation': 'logistic', 'alpha': 0.01, 'batch_size': 100, 'hidden_layer_sizes': (150,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.1, 'solver': 'sgd'}.

E. Support Vector Machine

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification tasks. SVM works by finding the optimal hyperplane that separates different classes in the feature space while maximizing the margin between the classes. It is particularly effective in high-dimensional spaces and when the number of features exceeds the number of samples.

The hyperparameters of SVM play a crucial role in determining the model's performance and flexibility. The primary hyperparameters include:

C: Regularization parameter, which controls the trade-off between maximizing the margin and minimizing classification errors.

kernel: Specifies the type of kernel function used for mapping input data into a higher-dimensional space. Options include 'linear', 'poly' (polynomial), 'rbf' (radial basis function), and 'sigmoid'.

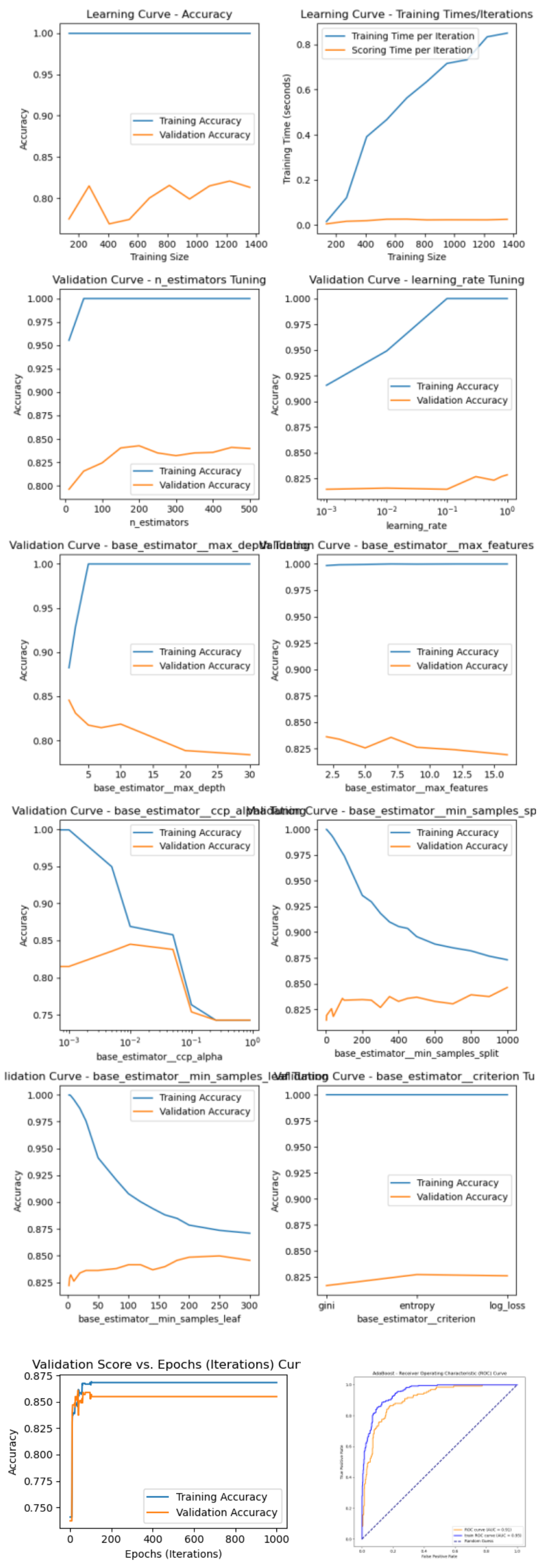
degree: Degree of the polynomial kernel function (for poly kernel).

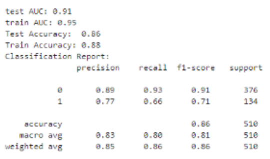
gamma: Kernel coefficient for 'rbf', 'poly', and 'sigmoid' kernels. It defines the influence of individual training samples on the decision boundary.

coef0: Independent term in the kernel function for 'poly' and 'sigmoid' kernels.

To identify the optimal combination of hyperparameters, we conduct a thorough analysis of training and validation curves across various parameter values. This analysis helps identify a suitable range for each hyperparameter, balancing model complexity and performance. Subsequently, grid search is employed to fine-tune the model, exploring combinations of hyperparameters defined in the grid. The grid search aims to find the best combination of hyperparameters that maximizes classification performance.

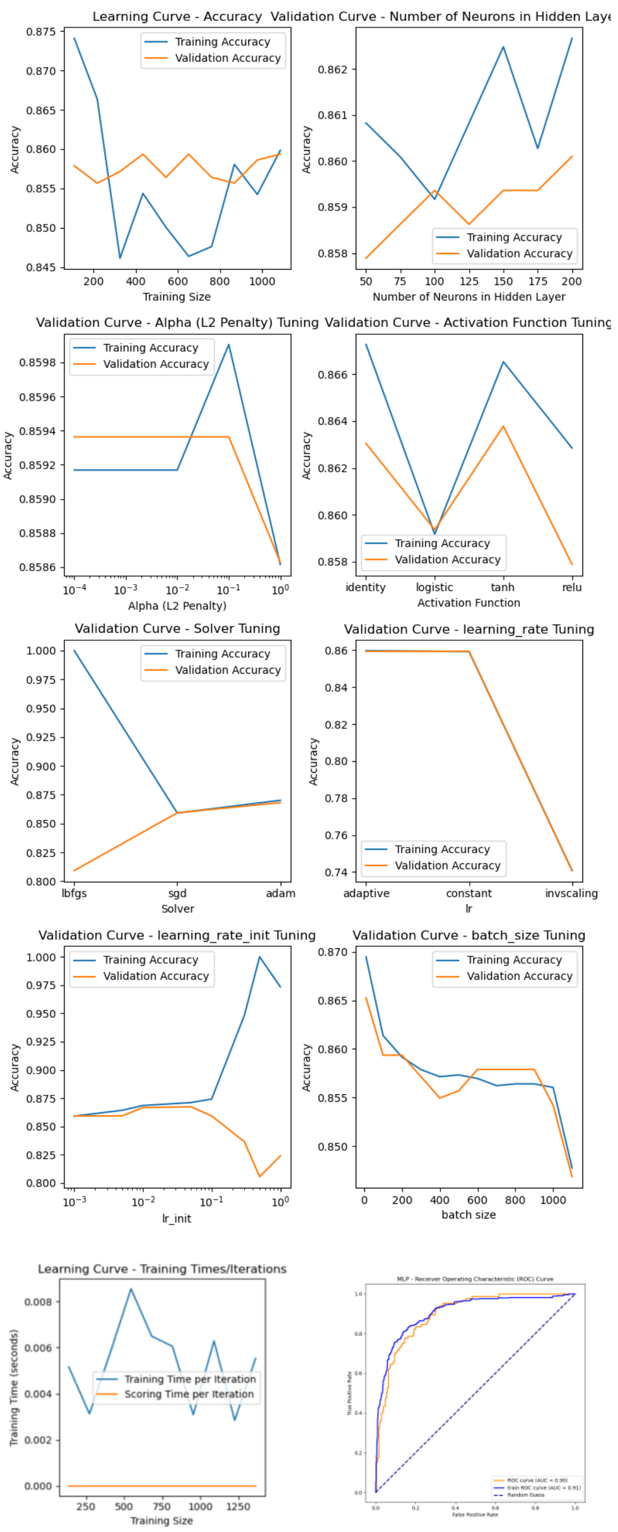
Based on the results of the grid search, the best hyperparameters for the SVM model are determined to be {'C': 0.1, 'coef0': -0.8, 'degree': 1, 'gamma': 0.001, 'kernel': 'linear'}. These hyperparameters optimize the classification performance of the SVM model for the given tasks.

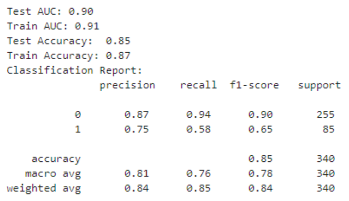




```
test AUC: 0.91
train AUC: 0.95
Test Accuracy:  0.86
Train Accuracy: 0.88
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       376
           1       0.77      0.66      0.71       134

    accuracy                           0.86       510
   macro avg       0.83      0.80      0.81       510
weighted avg       0.85      0.86      0.86       510
```

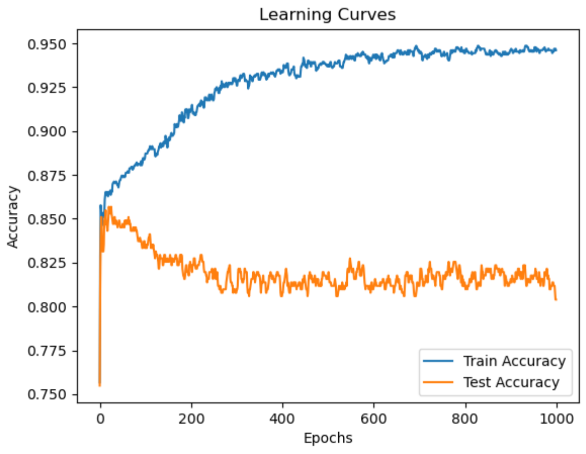
Fig.3 Performance of AdaBoost (Best Hyperparameters: {'ccp_alpha': 0.01, 'criterion': 'log_loss', 'max_depth': 7, 'max_features': 9, 'min_samples_leaf': 160, 'min_samples_split': 250, 'splitter': 'best'})

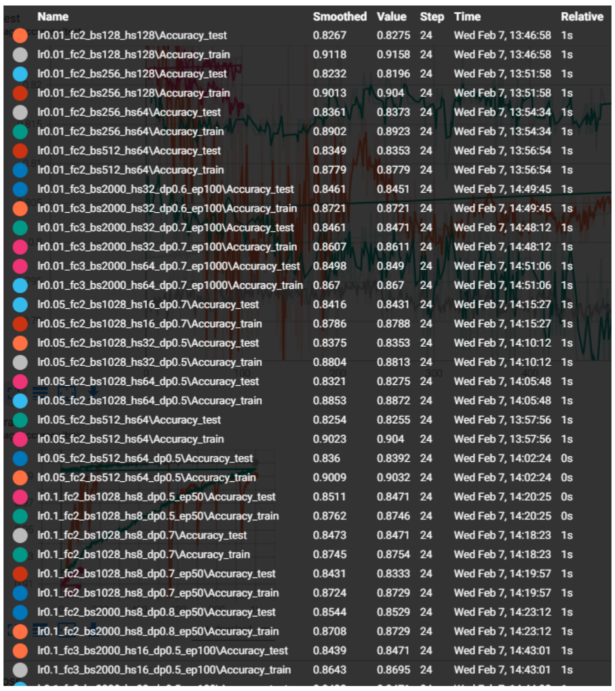




```
Test AUC: 0.90
Train AUC: 0.91
Test Accuracy:  0.85
Train Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.90       255
           1       0.75      0.58      0.65        85

    accuracy                           0.85       340
   macro avg       0.81      0.76      0.78       340
weighted avg       0.84      0.85      0.84       340
```

Fig.4a Performance of NNs (Best Hyperparameters: {'activation': 'logistic', 'alpha': 0.01, 'batch_size': 100, 'hidden_layer_sizes': (150,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.1, 'solver': 'sgd'})



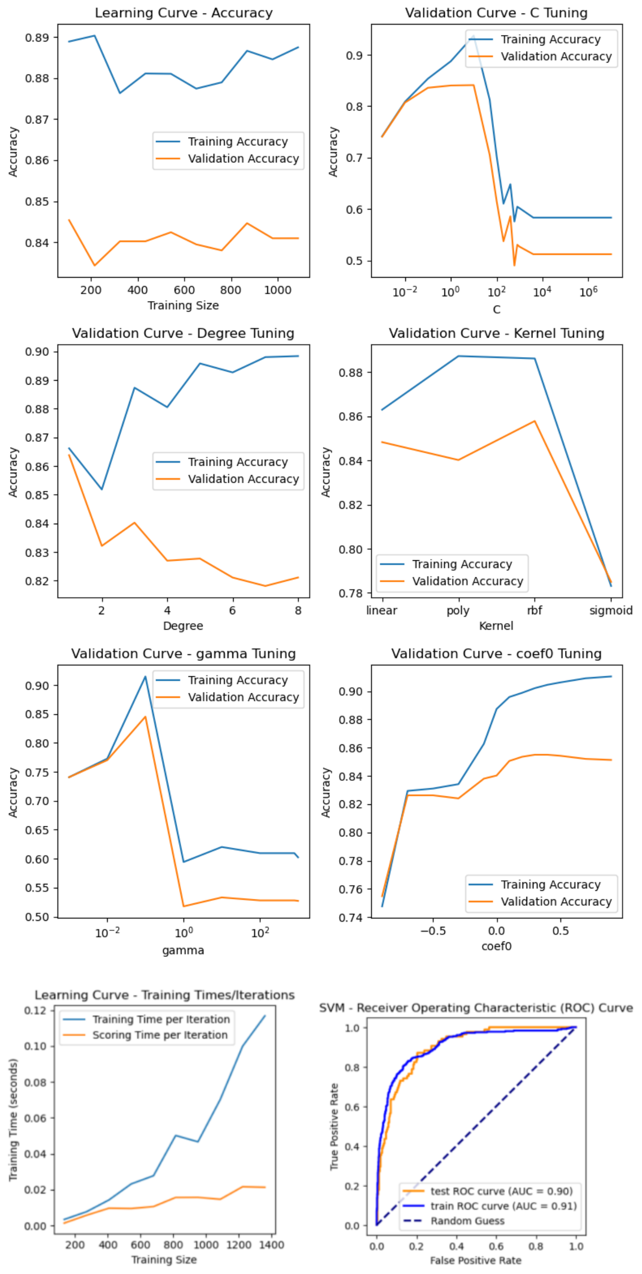


	Name	Smoothed	Value	Step	Time	Relative
●	lr0.01_fc2_bs128_hs128\Accuracy_test	0.8267	0.8275	24	Wed Feb 7, 13:46:58	1s
●	lr0.01_fc2_bs128_hs128\Accuracy_train	0.9118	0.9158	24	Wed Feb 7, 13:46:58	1s
●	lr0.01_fc2_bs256_hs128\Accuracy_test	0.8232	0.8196	24	Wed Feb 7, 13:51:58	1s
●	lr0.01_fc2_bs256_hs128\Accuracy_train	0.9013	0.904	24	Wed Feb 7, 13:51:58	1s
●	lr0.01_fc2_bs256_hs64\Accuracy_test	0.8361	0.8373	24	Wed Feb 7, 13:54:34	1s
●	lr0.01_fc2_bs256_hs64\Accuracy_train	0.8902	0.8923	24	Wed Feb 7, 13:54:34	1s
●	lr0.01_fc2_bs512_hs64\Accuracy_test	0.8349	0.8353	24	Wed Feb 7, 13:56:54	1s
●	lr0.01_fc2_bs512_hs64\Accuracy_train	0.8779	0.8779	24	Wed Feb 7, 13:56:54	1s
●	lr0.01_fc3_bs2000_hs32_dp0.6_ep100\Accuracy_test	0.8461	0.8451	24	Wed Feb 7, 14:49:45	1s
●	lr0.01_fc3_bs2000_hs32_dp0.6_ep100\Accuracy_train	0.8721	0.8721	24	Wed Feb 7, 14:49:45	1s
●	lr0.01_fc3_bs2000_hs32_dp0.7_ep100\Accuracy_test	0.8461	0.8471	24	Wed Feb 7, 14:48:12	1s
●	lr0.01_fc3_bs2000_hs32_dp0.7_ep100\Accuracy_train	0.8607	0.8611	24	Wed Feb 7, 14:48:12	1s
●	lr0.01_fc3_bs2000_hs64_dp0.7_ep1000\Accuracy_test	0.8498	0.849	24	Wed Feb 7, 14:51:06	1s
●	lr0.01_fc3_bs2000_hs64_dp0.7_ep1000\Accuracy_train	0.867	0.867	24	Wed Feb 7, 14:51:06	1s
●	lr0.05_fc2_bs1028_hs16_dp0.7\Accuracy_test	0.8416	0.8431	24	Wed Feb 7, 14:15:27	1s
●	lr0.05_fc2_bs1028_hs16_dp0.7\Accuracy_train	0.8786	0.8788	24	Wed Feb 7, 14:15:27	1s
●	lr0.05_fc2_bs1028_hs32_dp0.5\Accuracy_test	0.8375	0.8353	24	Wed Feb 7, 14:10:12	1s
●	lr0.05_fc2_bs1028_hs32_dp0.5\Accuracy_train	0.8804	0.8813	24	Wed Feb 7, 14:10:12	1s
●	lr0.05_fc2_bs1028_hs64_dp0.5\Accuracy_test	0.8321	0.8275	24	Wed Feb 7, 14:05:48	1s
●	lr0.05_fc2_bs1028_hs64_dp0.5\Accuracy_train	0.8853	0.8872	24	Wed Feb 7, 14:05:48	1s
●	lr0.05_fc2_bs512_hs64\Accuracy_test	0.8254	0.8255	24	Wed Feb 7, 13:57:56	1s
●	lr0.05_fc2_bs512_hs64\Accuracy_train	0.9023	0.904	24	Wed Feb 7, 13:57:56	1s
●	lr0.05_fc2_bs512_hs64_dp0.5\Accuracy_test	0.836	0.8392	24	Wed Feb 7, 14:02:24	0s
●	lr0.05_fc2_bs512_hs64_dp0.5\Accuracy_train	0.9009	0.9032	24	Wed Feb 7, 14:02:24	0s
●	lr0.1_fc2_bs1028_hs8_dp0.5_ep50\Accuracy_test	0.8511	0.8471	24	Wed Feb 7, 14:20:25	0s
●	lr0.1_fc2_bs1028_hs8_dp0.5_ep50\Accuracy_train	0.8762	0.8746	24	Wed Feb 7, 14:20:25	0s
●	lr0.1_fc2_bs1028_hs8_dp0.7\Accuracy_test	0.8473	0.8471	24	Wed Feb 7, 14:18:23	1s
●	lr0.1_fc2_bs1028_hs8_dp0.7\Accuracy_train	0.8745	0.8754	24	Wed Feb 7, 14:18:23	1s
●	lr0.1_fc2_bs1028_hs8_dp0.7_ep50\Accuracy_test	0.8431	0.8333	24	Wed Feb 7, 14:19:57	1s
●	lr0.1_fc2_bs1028_hs8_dp0.7_ep50\Accuracy_train	0.8724	0.8729	24	Wed Feb 7, 14:19:57	1s
●	lr0.1_fc2_bs2000_hs8_dp0.8_ep50\Accuracy_test	0.8544	0.8529	24	Wed Feb 7, 14:23:12	1s
●	lr0.1_fc2_bs2000_hs8_dp0.8_ep50\Accuracy_train	0.8708	0.8729	24	Wed Feb 7, 14:23:12	1s
●	lr0.1_fc3_bs2000_hs16_dp0.5_ep100\Accuracy_test	0.8439	0.8471	24	Wed Feb 7, 14:43:01	1s
●	lr0.1_fc3_bs2000_hs16_dp0.5_ep100\Accuracy_train	0.8643	0.8695	24	Wed Feb 7, 14:43:01	1s

Fig.4b Performance of NNs (Best Hyperparameters: {'activation': 'logistic', 'alpha': 0.01, 'batch_size': 100, 'hidden_layer_sizes': (150,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.1, 'solver': 'sgd'})

F. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple yet effective non-parametric classification algorithm used for both regression and classification tasks. KNN works by finding the K nearest data points in the feature space to a given query point and assigning the majority class (for classification) or the average of the K-nearest neighbors' values (for regression) as the predicted output.



```
test AUC: 0.90
train AUC: 0.91
Test Accuracy:  0.84
Train Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90       255
           1       0.75      0.55      0.64        85

    accuracy                           0.84       340
   macro avg       0.80      0.75      0.77       340
weighted avg       0.83      0.84      0.83       340
```

Fig.5 Performance of SVM (Best Hyperparameters: {'C': 0.1, 'coef0': -0.8, 'degree': 1, 'gamma': 0.001, 'kernel': 'linear'})

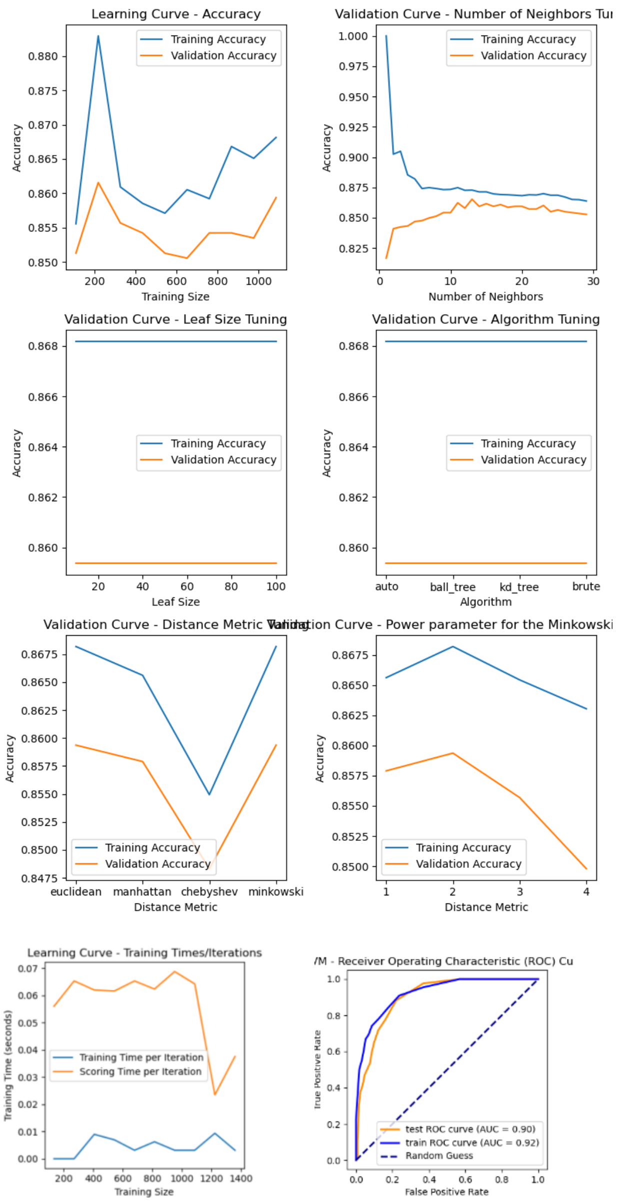
The hyperparameters of KNN play a crucial role in determining the model's performance and behavior. The primary hyperparameters include:

n_neighbors: The number of neighbors considered when making predictions. It directly affects the bias-variance trade-off of the model, with smaller values leading to higher variance and potential overfitting, while larger values result in higher bias.

algorithm: The algorithm used to compute the nearest neighbors. Options include 'auto', 'ball_tree', 'kd_tree', and 'brute'.

metric: The distance metric used to measure the similarity between data points. Common options include 'euclidean', 'manhattan', 'chebyshev', and 'minkowski'.

p: The power parameter for the Minkowski metric, where p=1 corresponds to the Manhattan distance and p=2 corresponds to the Euclidean distance.



```
test AUC: 0.90
train AUC: 0.92
Test Accuracy:  0.83
Train Accuracy: 0.88
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.92      0.89       255
           1       0.70      0.54      0.61        85

    accuracy                           0.83       340
   macro avg       0.78      0.73      0.75       340
weighted avg       0.82      0.83      0.82       340
```

Fig.6 Performance of KNN (Best Hyperparameters: {'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 15, 'p': 1})

To identify the optimal combination of hyperparameters, The author conduct a thorough analysis of training and validation curves across various parameter values. This analysis helps identify a suitable range for each hyperparameter, balancing model complexity and performance. Subsequently, grid search is employed to fine-tune the model, exploring combinations of hyperparameters defined in the grid. The grid search aims to find the best combination of hyperparameters that maximizes classification performance.

Based on the results of the grid search, the best hyperparameters for the KNN model are determined to be {'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 15, 'p': 1}. These hyperparameters optimize the classification performance of the KNN model for the given tasks.

G. Comparison of Results from 6 Algorithms:

In the evaluation of classification models, several metrics are commonly used to assess their performance, including Receiver Operating Characteristic (ROC) curves, Area Under the Curve (AUC), and accuracy.

ROC curves provide a graphical representation of the trade-off between true positive rate (sensitivity) and false positive rate (1 - specificity) across different threshold values. The AUC summarizes the performance of the model across all possible thresholds, with a higher AUC indicating better discriminative ability. AUC values close to 1 indicate excellent performance, while values around 0.5 suggest random guessing.

Accuracy, on the other hand, measures the proportion of correctly classified instances out of the total number of instances. While accuracy is a straightforward metric, it may not provide a complete picture of a model's performance, especially in imbalanced datasets where one class dominates the other.

Table 1 Comparison

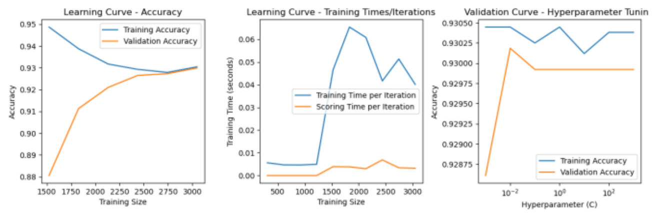
	LR	DT	AdaBoost	NNs	SVM	KNN
Test AUC	0.89	0.83	0.91	0.90	0.90	0.90
Train AUC	0.89	0.82	0.95	0.91	0.91	0.92
Test Accuracy	0.85	0.82	0.86	0.85	0.84	0.83
Train Accuracy	0.86	0.85	0.88	0.87	0.87	0.88

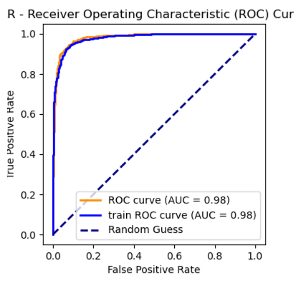
The detailed comparison of the results from the six algorithms presented in Table 1. From the comparison, it's evident that AdaBoost achieves the highest AUC score on both the test and train sets, indicating superior discriminative ability. However, it's noteworthy that AdaBoost also has slightly lower accuracy compared to LR, NNs, and SVM. Decision Trees (DT) exhibit the lowest AUC and accuracy among the six algorithms, suggesting relatively weaker performance. Neural Networks (NNs), SVM, and K-Nearest Neighbors (KNN) demonstrate consistent performance with relatively high AUC and accuracy scores on both test and train sets. Overall, the choice of algorithm should consider the trade-offs between discriminative ability, computational complexity, and interpretability, depending on the specific requirements of the application.

IV. TASK II

A. Logistic Regression

Follow the same procedure in task I and plot the learning curves and validation curves as well as ROC in Figure 7.



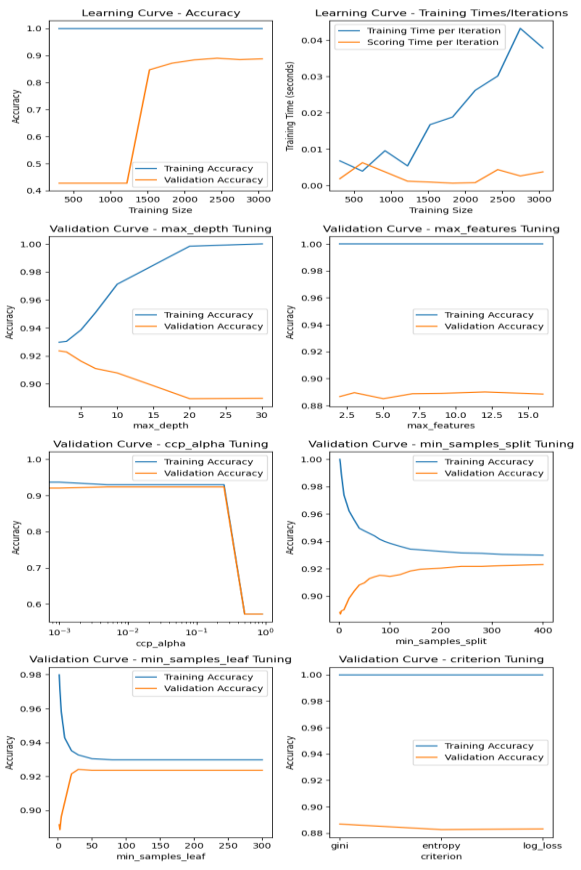


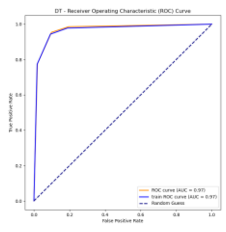
```
AUC: 0.98
train AUC: 0.98
Test Accuracy: 0.93
Train Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.91      0.92       518
           1       0.92      0.95      0.94       625

    accuracy                           0.93      1143
   macro avg       0.93      0.93      0.93      1143
weighted avg       0.93      0.93      0.93      1143
```

Fig.7 Performance of Logistic Regression (Best Hyperparameters: {'C': 100, 'max_iter': 1000})

B. Decision Tree

Follow the same procedure in task I and plot the learning curves and validation curves as well as ROC in Figure 8.



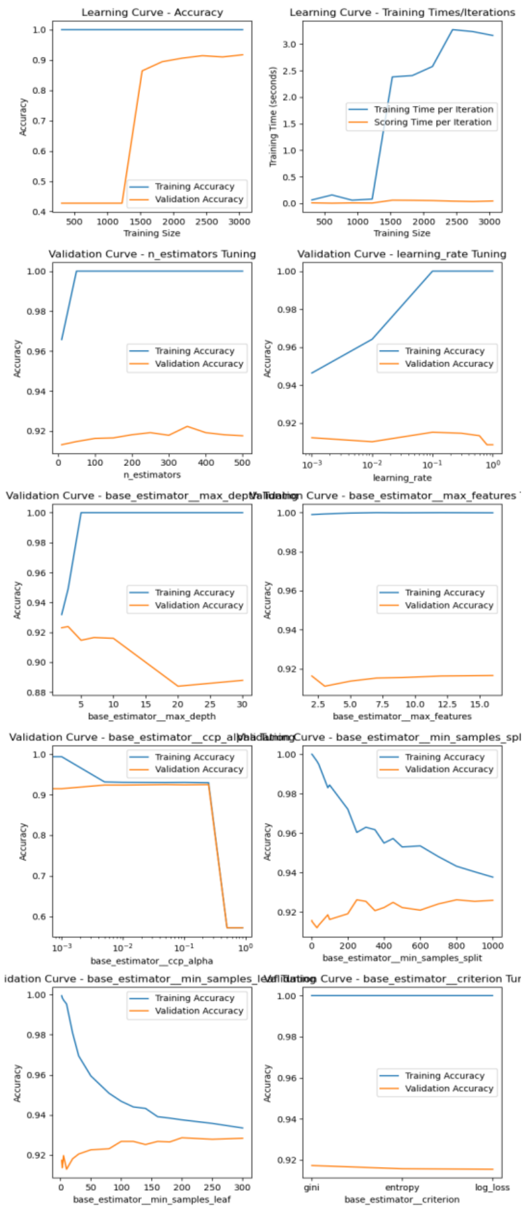


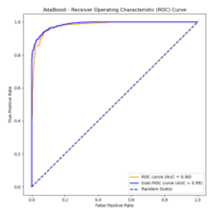
```
AUC: 0.97
train AUC: 0.97
Test Accuracy: 0.93
Train Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.90      0.92       518
           1       0.92      0.95      0.94       625

    accuracy                           0.93      1143
   macro avg       0.93      0.93      0.93      1143
weighted avg       0.93      0.93      0.93      1143
```

Fig.8 Performance of Decision Tree (Best Hyperparameters: {'ccp_alpha': 0, 'criterion': 'gini', 'max_depth': 2, 'max_features': None, 'min_samples_leaf': 50, 'min_samples_split': 300, 'splitter': 'best'})

C. *AdaBoost*

Follow the same procedure in task I and plot the learning curves and validation curves as well as ROC in Figure 9.



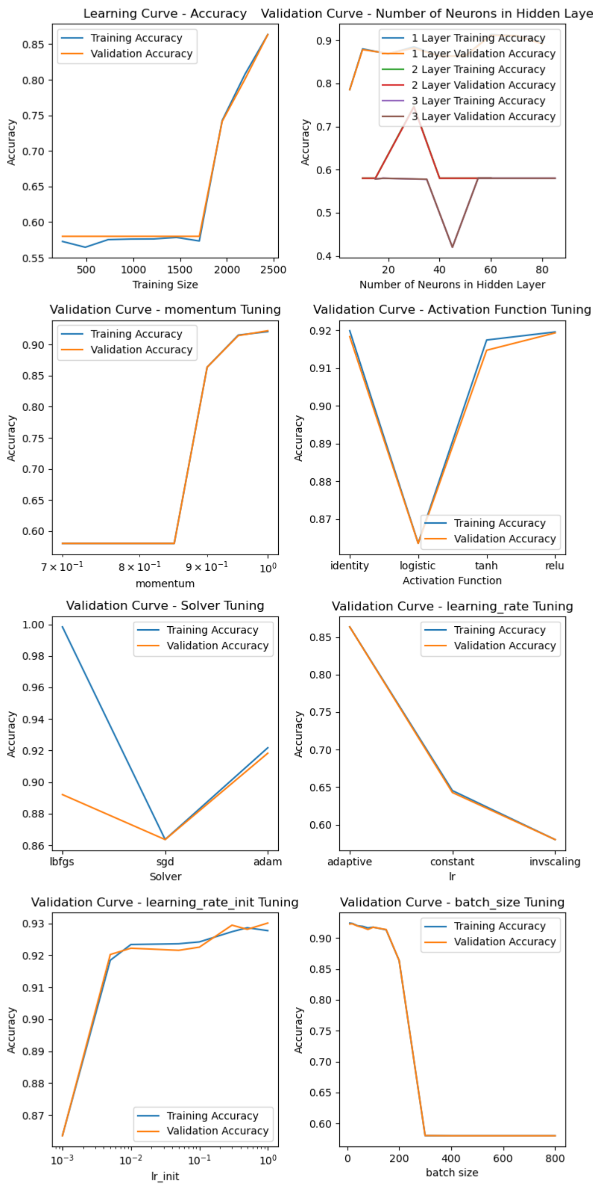


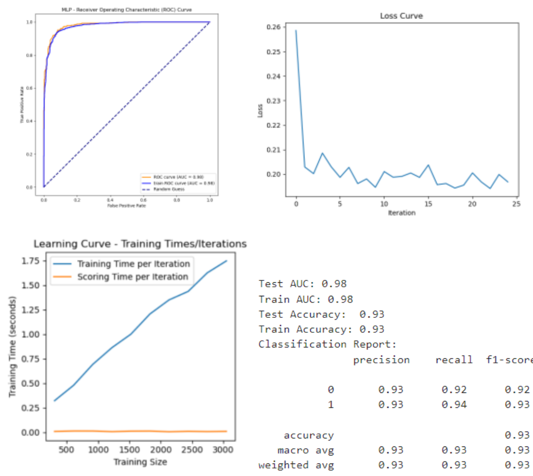
```
test AUC: 0.98
train AUC: 0.99
Test Accuracy:  0.93
Train Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.91      0.92       518
           1       0.92      0.96      0.94       625

    accuracy                           0.93      1143
   macro avg       0.93      0.93      0.93      1143
weighted avg       0.93      0.93      0.93      1143
```

Fig.9 Performance of AdaBoost (Best Hyperparameters: {'base_estimator__criterion': 'entropy', 'base_estimator__max_depth':5, 'base_estimator__max_features': 6, 'base_estimator__min_samples_leaf': 200, 'base_estimator__splitter': 'best', 'learning_rate': 0.05, 'n_estimators': 50})

D. *Neural Network*

Follow the same procedure in task I and plot the learning curves and validation curves as well as ROC in Figure 10.



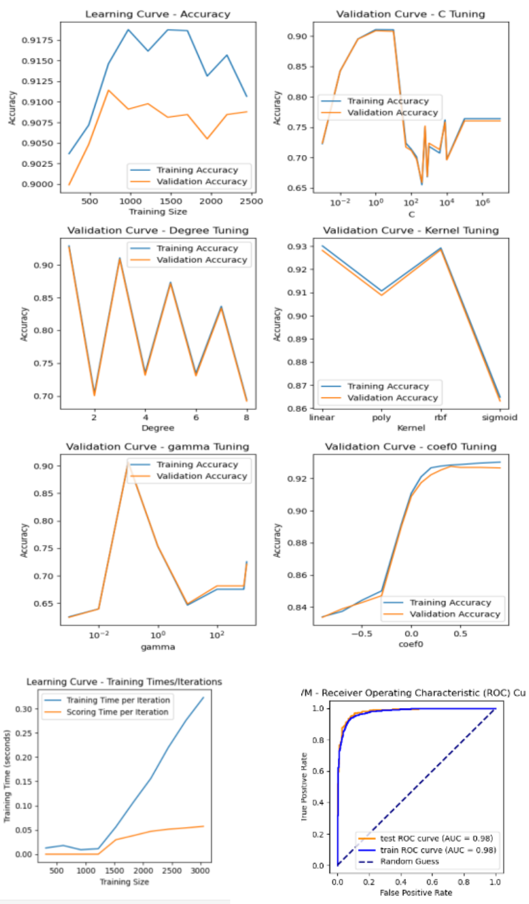


```
Test AUC: 0.98
Train AUC: 0.98
Test Accuracy:  0.93
Train Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       350
           1       0.93      0.94      0.93       412

    accuracy                           0.93       762
   macro avg       0.93      0.93      0.93       762
weighted avg       0.93      0.93      0.93       762
```

Fig.10 Performance of NNs (Best Hyperparameters: {'activation': 'relu', 'alpha': 0.01, 'batch_size': 50, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.1, 'solver': 'adam'})

E. *Support Vector Machine*

Follow the same procedure in task I and plot the learning curves and validation curves as well as ROC in Figure 11.

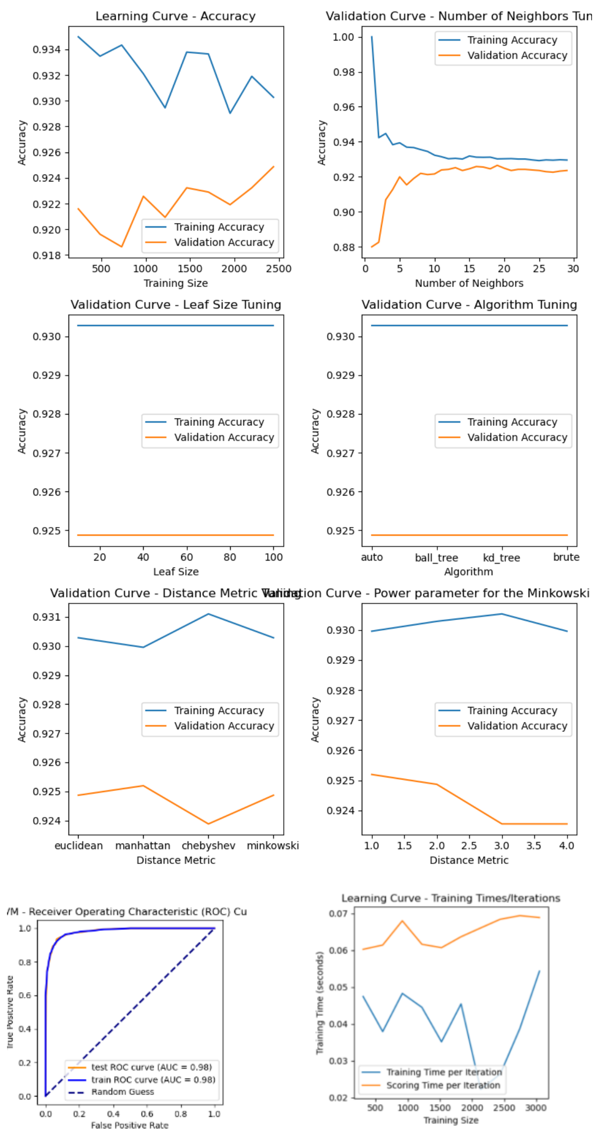


```
test AUC: 0.98
train AUC: 0.98
Test Accuracy:  0.93
Train Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.92      0.93       350
           1       0.93      0.94      0.94       412

    accuracy                           0.93       762
   macro avg       0.93      0.93      0.93       762
weighted avg       0.93      0.93      0.93       762
```

Fig.11 Performance of SVMs (Best Hyperparameters: {'C': 0.1, 'coef0': -0.3, 'gamma': 0.05, 'kernel': 'sigmoid'})

F. *K-Nearest Neighbors*

Follow the same procedure in task I and plot the learning curves and validation curves as well as ROC in Figure 12.



```
test AUC: 0.98
train AUC: 0.98
Test Accuracy:  0.93
Train Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.92      0.93       350
           1       0.93      0.94      0.94       412

    accuracy                           0.93       762
   macro avg       0.93      0.93      0.93       762
weighted avg       0.93      0.93      0.93       762
```

Fig.12 Performance of KNN (Best Hyperparameters: {'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 19, 'p': 1})

G. *Comparison of Results from 6 Algorithms:*

The detailed comparison of the results from the six algorithms presented in Table 1. From the comparison, it's evident that AdaBoost achieves the highest AUC score on both the test and train sets, indicating superior discriminative ability. However, it's noteworthy that AdaBoost also has slightly lower accuracy compared to LR, NNs, and SVM. Decision Trees (DT) exhibit the lowest AUC and accuracy among the six algorithms, suggesting relatively weaker performance. Neural Networks (NNs), SVM, and K-Nearest

Neighbors (KNN) demonstrate consistent performance with relatively high AUC and accuracy scores on both test and train sets. Overall, the choice of algorithm should consider the trade-offs between discriminative ability, computational complexity, and interpretability, depending on the specific requirements of the application.

Table 2 Comparison

	LR	DT	AdaBoost	NNs	SVM	KNN
Test AUC	0.98	0.97	0.98	0.98	0.98	0.98
Train AUC	0.98	0.97	0.99	0.98	0.98	0.98
Test Accuracy	0.93	0.93	0.93	0.93	0.93	0.93
Train Accuracy	0.93	0.93	0.93	0.93	0.93	0.93

V. Results

In this section, we present the results of the supervised learning experiments conducted on two classification tasks. The performance of six different algorithms, namely Logistic Regression (LR), Decision Trees (DT), AdaBoost, Neural Networks (NNs), Support Vector Machines (SVM), and K-Nearest Neighbors (KNN), is evaluated and compared across the tasks.

Task 1: Identification of Disadvantaged Communities

In Table 1, the performance metrics including Test AUC, Train AUC, Test Accuracy, and Train Accuracy are provided for each algorithm. Notably, AdaBoost demonstrates the highest Test AUC of 0.91 and Train AUC of 0.95, indicating strong discriminative ability. However, the model exhibits slight overfitting, as indicated by the difference between Test and Train AUC scores. In terms of Test Accuracy, Logistic Regression and Neural Networks perform comparably, both achieving an accuracy of 0.85.

Task 2: Rice Classification

Table 2 displays the results for the rice classification task. Across all algorithms, consistent performance is observed with high Test AUC and Test Accuracy scores, indicating robust classification capability. Notably, AdaBoost achieves the highest Test AUC and Train AUC scores of 0.98 and 0.99, respectively. Minimal underfitting or overfitting is observed, with negligible differences between Test and Train AUC scores for all algorithms.

Underfitting and Overfitting

In both tasks, minimal underfitting or overfitting is observed, as evidenced by the comparable Test and Train AUC scores across algorithms. This suggests that the models generalize well to unseen data while maintaining high performance on the training data.

Computational Complexity

While all algorithms demonstrate high performance, variations in computational complexity exist. Decision Trees and K-Nearest Neighbors tend to have lower computational requirements due to their simplicity, while Neural Networks and Support Vector Machines may require more computational resources, especially for larger datasets or complex architectures.

Overall, AdaBoost consistently demonstrates strong performance across both tasks, achieving high AUC and accuracy scores with minimal overfitting. However, the choice of algorithm should consider not only performance metrics but also computational requirements and interpretability, depending on the specific needs of the application.

References

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830

[2] https://github.com/PacktPublishing/Hands-on-Reinforcement-Learning-with-PyTorch

[3] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., .& Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. arXiv preprint arXiv:1603.04467.