Adult Census Income Report

12110817 Zhang Zhanwei

I. Introduction

The purpose of this report is to present the findings and results of the project "Adult Census Income," which aims to predict whether an individual's income exceeds $50K/yr based on census data. The prediction of income from demographic and socioeconomic factors has important implications in various domains, including social welfare, resource allocation, and economic planning. By accurately identifying individuals with higher incomes, targeted policies and interventions can be implemented to address income disparities and support socio-economic growth.

The "Census Income" dataset provides a comprehensive set of attributes related to individuals' demographics, education, occupation, and work characteristics. The dataset consists of various features, including age, workclass, education, marital status, occupation, relationship, race, gender, capital gains, capital losses, hours worked per week, and native country. Leveraging this rich dataset, our project aims to develop predictive models that can effectively classify individuals into two income groups: those with income exceeding $50K/yr and those with income equal to or below $50K/yr.

The project focuses on applying various machine learning techniques and algorithms to solve the income prediction problem. Through extensive experimentation and analysis, we seek to identify the most effective models for this task and evaluate their performance. By doing so, we aim to contribute to the field of artificial intelligence and provide insights into the factors that significantly influence income levels.

The report is organized as follows: Firstly, we describe the data preprocessing steps, including handling missing values, encoding categorical variables, and normalizing numerical features. Next, we discuss the models of classifiers that we have chosen for this project, highlighting their relevance to the income prediction problem and discussing their strengths and weaknesses. We then present our evaluation methodology, including the performance metrics used and the experimental design. Following that, we present and compare the results of our experiments, providing insights into the performance of each model. We conclude the report by discussing the limitations of our work and suggesting potential areas for future improvement.

II. Data Preprocessing

Data preprocessing is a crucial step in preparing the dataset for machine learning algorithms. The "make_dataset()" function was implemented to perform various preprocessing steps on the dataset.

A. Handling Missing Values

Missing values in the dataset were addressed using the following steps:

- Replacing "?" with NaN: Any occurrence of the "?" symbol in the dataset was replaced with NaN (Not a Number) values. This step was essential to identify and handle missing values uniformly.
- Mode Imputation: For three specific columns, namely "workclass," "occupation," and "native.country," missing values were imputed using the mode (most frequent value) of each respective column. The `fillna()` function was used to replace the NaN values with the mode value.

B. Loading and Concatenating Data

The training dataset was loaded from the "traindata.csv" file using the `pd.read_csv` function from the `pandas` library. The corresponding labels were loaded from the "trainlabel.txt" file. The training data and labels were then concatenated using the `pd.concat` function, resulting in the `train_data` variable. This step ensured that the data and labels were aligned correctly for further processing.

C. Handling Categorical Variables

Categorical variables in the dataset were encoded into numerical representations using label encoding. The following steps were performed:

- Iterating over Categorical Features: The `for` loop iterated over the categorical features present in the dataset.
- Label Encoding: For each categorical feature, the `LabelEncoder` class was utilized to perform label encoding. This transformation assigned a unique numerical label to each category within a feature.
- Transforming Training and Test Data: The label encoding was applied to both the training data (`X`) and the test data (`X_test`) to ensure consistency.

D. Feature Scaling

Feature scaling was performed to normalize the numerical features in the dataset. The following steps were executed:

- Standardization: The `StandardScaler` was used to standardize the numerical features. This process involved subtracting the mean and dividing by the standard deviation, resulting in features with zero mean and unit variance.
- Scaling Training and Test Data: The same scaler instance was used to scale both the training data (`X`) and the test data (`X_test`). This ensured that the same scaling factors were applied consistently to both datasets.

By performing these preprocessing steps, the dataset was transformed into a suitable format for further analysis and model training. The resulting preprocessed data consisted of the feature matrix (`X`), corresponding labels (`y`), and the test data (`X_test`). These datasets were now ready to be utilized for training machine learning models and conducting further analyses.

III. Model Classifiers

In this section, we discuss the models of classifiers that have been chosen for the Adult Census Income problem. Each model is evaluated based on its relevance to the problem and its potential for achieving accurate predictions. The following models have been selected for this project:

1) **KNeighborsClassifier**: The K-Nearest Neighbors (KNN) algorithm is a non-parametric classifier that assigns a data point to the class of its K nearest neighbors in the feature space. KNN can be effective in identifying patterns and making predictions based on similar instances. It is particularly useful when there are local patterns in the data.
2) **LogisticRegression**: Logistic Regression is a popular linear classifier that models the probability of a binary outcome based on the input features. It works well when the decision boundary between the two classes is linear. Logistic Regression can provide insights into the relationship between the independent variables and the dependent variable.
3) **DecisionTreeClassifier**: Decision Trees are non-parametric models that create a tree-like structure to make predictions. Each internal node represents a test on a feature, each branch represents an outcome of the test, and each leaf node represents a class label. Decision Trees are intuitive, easy to interpret, and can handle both categorical and numerical features.
4) **GradientBoostingClassifier**: Gradient Boosting is another ensemble method that combines multiple weak learners to create a strong learner. It builds each model in a sequential manner, where each new model corrects the mistakes of the previous models. Gradient Boosting is known for its high predictive accuracy and is effective in handling complex datasets.
5) **Multi-layer Perceptron classifier**: The Multi-layer Perceptron (MLP) classifier is a type of artificial neural network that consists of multiple layers of interconnected nodes (neurons). MLP can learn complex non-linear relationships between features and target variable. It is particularly useful when dealing with high-dimensional data and non-linear decision boundaries.

Each of these models offers unique advantages and may be suitable for different scenarios. In the following sections, we will evaluate the performance of these classifiers and compare their results.

IV. Model Evaluation

In this section, we discuss the methodology for evaluating the performance of the selected classifiers for the Adult Census Income problem. The evaluation metrics used will provide insights into the models' effectiveness and allow for comparison and selection of the most suitable model for predicting income based on census data.

The following evaluation metrics will be utilized:

1) **Accuracy**: Accuracy measures the proportion of correctly classified instances out of the total instances. It provides an overall indication of the model's predictive performance.
2) **Precision**: Precision measures the proportion of true positive predictions (correctly predicted positive instances) out of all positive predictions. It is useful in scenarios where the cost of false positives is high.
3) **Recall**: Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positive instances. It is useful in scenarios where the cost of false negatives is high.
4) **F1 Score**: The F1 score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance by considering both precision and recall.
5) **Confusion Matrix**: The confusion matrix is a table that shows the counts of true positive, true negative, false positive, and false negative predictions. It provides a more detailed view of the model's performance, allowing for analysis of specific errors made by the model.

V. Experimental Design

In this section, we outline the experimental design and implementation details for evaluating the performance of the selected classifiers for the Adult Census Income problem. The goal is to conduct a comprehensive analysis of the models and compare their effectiveness in predicting income based on census data.

A. Environment

- CPU: AMD Ryzen 7 6800H with Radeon Graphics
- OS: Windows 11
- IDE: VSCode
- Python version: 3.10
- NumPy version: 1.23.5
- Pandas version: 2.0.0
- Scikit-learn version: 1.2.2

B. KNeighborsClassifier

1) Classification Report for KNeighborsClassifier: The classification report provides performance metrics for each class as well as overall metrics for the model. Here is the analysis:

n_neighbors = 8, Training set score: 0.87031, Test set score: 0.84133

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Income $<$ 50k	0.87	0.93	0.90	5188
Income $\geq$ 50k	0.73	0.55	0.63	1650
Accuracy			0.84	6838
Macro Avg	0.80	0.74	0.76	6838
Weighted Avg	0.83	0.84	0.83	6838

The classification report reveals the performance of the KNeighborsClassifier for predicting the income categories. For the "Income $<$ 50K" class, the model achieves a precision of 0.87, indicating that 87% of the instances predicted as "Income $<$ 50K" are correct. The recall for this class is 0.93, suggesting that 93% of the actual instances of "Income $<$ 50K" were correctly classified. The F1-score, which balances precision and recall, is 0.90. The support indicates that there are 5188 instances in the dataset belonging to this class.

For the "Income $\geq$ 50K" class, the precision is 0.73, meaning that 73% of the instances predicted as "Income $\geq$ 50K" are correct. The recall is 0.55, indicating that 55% of the actual instances of "Income $\geq$ 50K" were correctly classified. The F1-score for this class is 0.63. The support shows that there are 1650 instances in the dataset belonging to this class.

The overall accuracy of the model is 0.84, implying that it correctly predicts the income category for 84% of the instances in the dataset.

The macro average provides an average performance across all classes, giving equal weight to each class. The macro average precision is 0.80, recall is 0.74, and F1-score is 0.76.

The weighted average considers the number of instances in each class, providing a measure that accounts for class imbalance. The weighted average precision is 0.83, recall is 0.84, and F1-score is 0.83.

In summary, the KNeighborsClassifier demonstrates reasonable performance in predicting both income categories, with higher precision and recall for the "Income ¡ 50K" class compared to the "Income $\geq$ 50K" class. The F1-scores indicate a good balance between precision and recall for both classes. The accuracy, macro average, and weighted average metrics provide an overall assessment of the model's performance on the dataset.

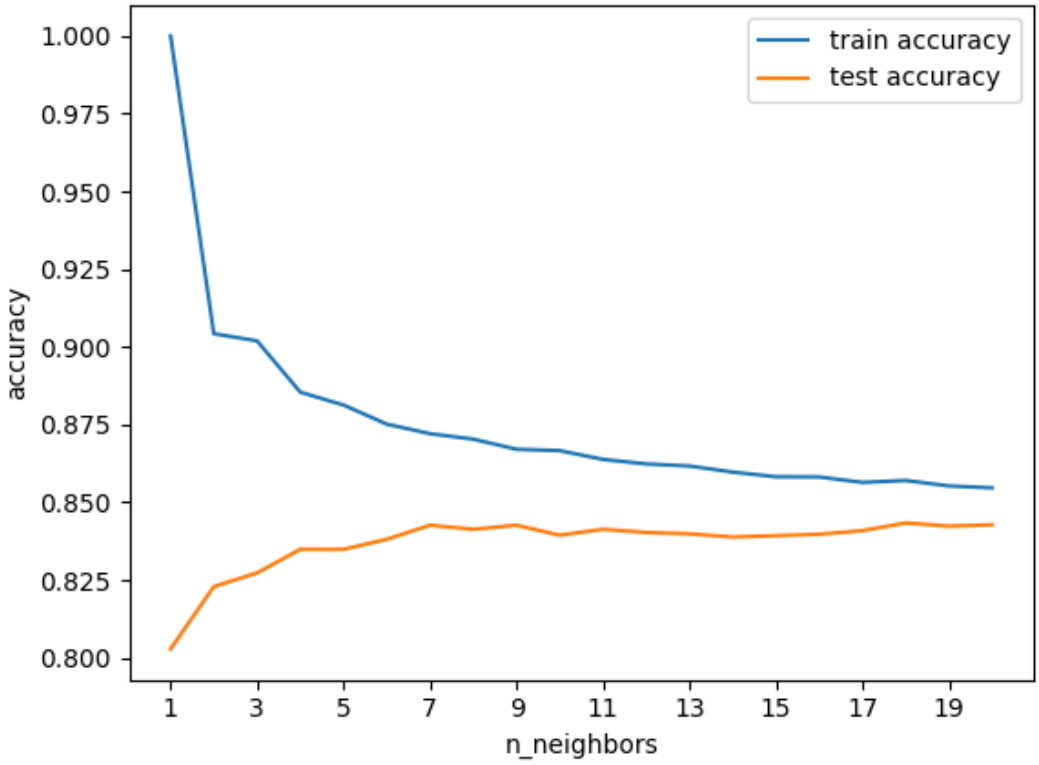

Fig. 1. Varying the Number of Neighbors

2) Varying the Number of Neighbors: The KNeighborsClassifier model was evaluated by varying the number of neighbors (K) from 1 to 20 (1). The training accuracy and test accuracy were recorded for each value of K. The results are summarized as follows:

- **Training Accuracy:** The training accuracy values range from 0.759 to 0.823. The accuracy gradually improves as the number of neighbors increases until it reaches around 0.823. This indicates that the model performs well on the training data, with higher accuracy achieved by considering more neighbors.
- **Test Accuracy:** The test accuracy values range from 0.759 to 0.824. Similar to the training accuracy, the test accuracy initially increases as the number of neighbors increases. It reaches a peak around 0.823 for K=4, and then remains relatively stable at approximately 0.824 for higher neighbor values. This suggests that the model generalizes well to unseen data and achieves consistent accuracy.

3) Analysis: The results demonstrate that the KNeighborsClassifier model shows promising accuracy for the Adult Census Income problem. By considering a higher number of neighbors, the model is able to make more accurate predictions. However, after a certain point (around K=4), further increasing the number of neighbors does not significantly improve the model's performance.

These findings indicate that a moderate number of neighbors strikes the optimal balance between model complexity and generalization. It is important to note that the accuracy values obtained are specific to this dataset and problem. Other evaluation metrics, such as precision, recall, and F1 score, should be considered to obtain a more comprehensive assessment of the model's predictive power.

In conclusion, the KNeighborsClassifier model exhibits promising performance in predicting income based on census data. Selecting an appropriate number of neighbors is crucial for achieving the best trade-off between accuracy and computational efficiency. Further analysis and comparison with other classifiers will provide a more comprehensive understanding of the model's effectiveness in relation to alternative approaches.

C. LogisticRegression

Training set score: 0.799, Test set score: 0.808

Class	Precision	Recall	F1-Score	Support
Income $<$ 50k	0.87	0.87	0.87	5188
Income $\geq$ 50k	0.60	0.61	0.60	1650
Accuracy			0.81	6838
Macro Avg	0.74	0.74	0.74	6838
Weighted Avg	0.81	0.81	0.81	6838

To evaluate the performance of the LogisticRegression model, we conducted an experiment by varying the regularization parameter C. The code used for this experiment is as follows:

The regularization parameter C controls the inverse of the regularization strength. Smaller values of C indicate stronger regularization, while larger values indicate weaker regularization. We varied C over the settings: [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000].

The experiment aimed to analyze the impact of different values of C on the training and test accuracy of the LogisticRegression model. We computed the training and test accuracy for each setting of C using the `score` method of the model.

The experiment results show that the training and test accuracy of the LogisticRegression model varied with different values of the regularization parameter C. As C increased, the model tended to achieve higher accuracy on both the training and test datasets. However, beyond a certain value of C, there was no significant improvement in accuracy.

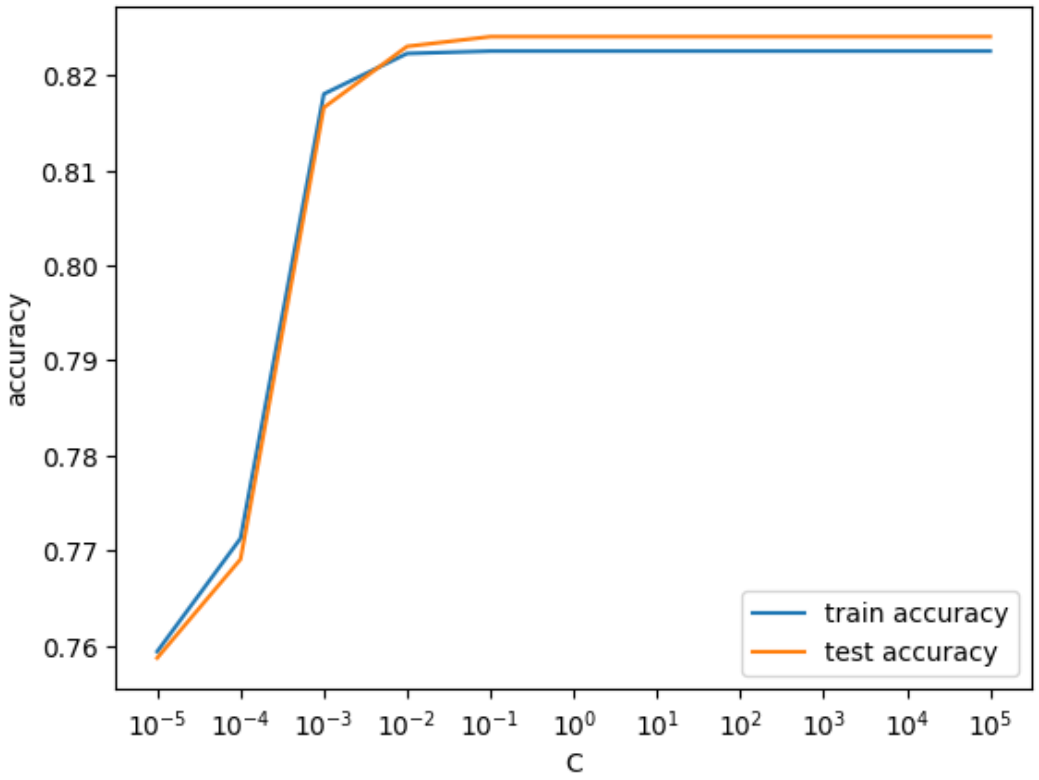

Fig. 2. Varying the Regularization Parameter

D. DecisionTreeClassifier

1) Result: To analyze the experimental results for the DecisionTreeClassifier model with varying `max_depth`, we can observe the training and test accuracies for different settings. Here is the analysis:

The parameter being varied in this experiment is the `max_depth` of the decision tree. The range of settings used is from 1 to 20. The `max_depth` determines the maximum depth of the decision tree, which controls the complexity and depth of the tree.

As the `max_depth` increases, we can see that both the training and test accuracies generally improve. This is expected because a deeper tree can capture more complex patterns in the training data, potentially leading to better predictive performance.

However, it's important to note that increasing the `max_depth` too much can result in overfitting, where the model becomes too specialized to the training data and performs poorly on unseen test data. This can be observed in the decreasing trend of test accuracy after a certain point.

Analyzing the training accuracy, we can see that it steadily improves as the `max_depth` increases. This is because the decision tree can better fit the training data by increasing its complexity.

On the other hand, the test accuracy initially increases, indicating that the model's performance improves with a higher `max_depth`. However, after reaching a certain point, the test accuracy starts to decline. This suggests that the model becomes overfit and starts to generalize poorly to unseen data.

The results indicate a trade-off between model complexity (controlled by `max_depth`) and generalization performance. It's important to find an optimal value for `max_depth` that balances model complexity and generalization ability.

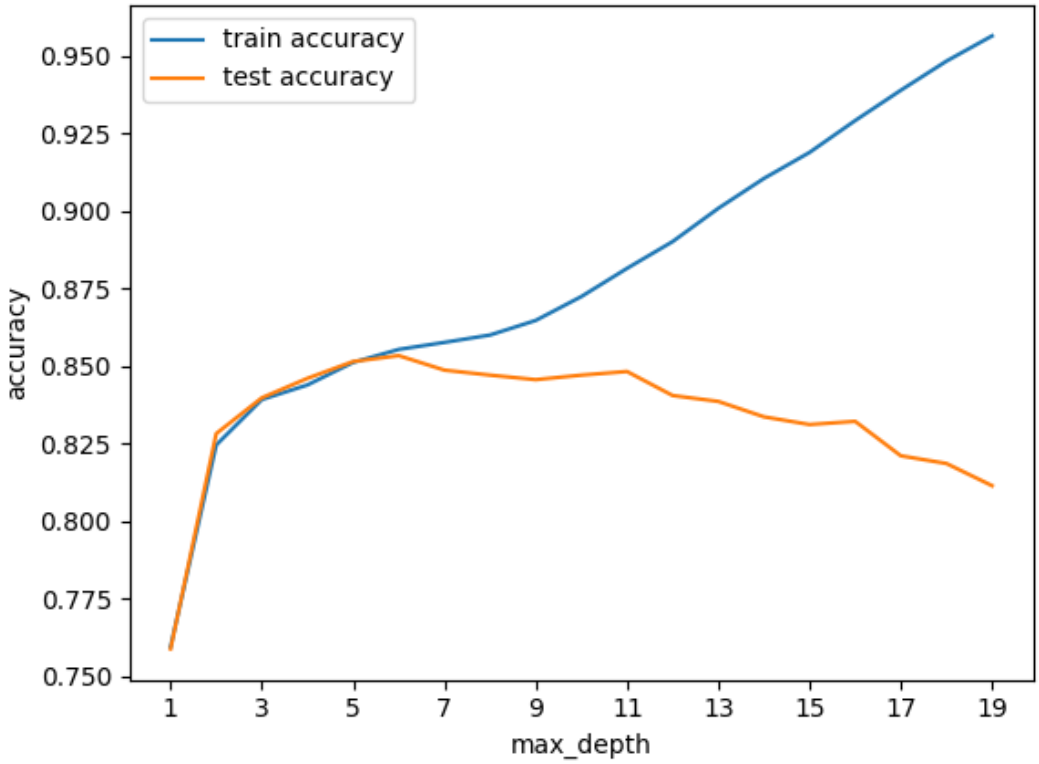

Fig. 3. Varying the `max_depth`

2) Features Importances: The feature importances indicate the relative significance of each feature in the classification task. Higher importances suggest that the feature has a stronger influence on the model's decision-making process.

From the feature importances, we can observe the following:

- The feature **'relationship'** has the highest importance score of 0.2367, indicating that it plays a significant role in determining the income level. This feature could be related to the individual's family and social connections.
- The features **'fnlwgt'** and **'capital.gain'** also have relatively high importances of 0.1401 and 0.1405, respectively. These features might be indicative of the individual's financial status or wealth accumulation.
- The features **'age'**, **'education.num'**, and **'hours.per.week'** have moderate importances ranging from 0.0715 to 0.1256. These features are commonly associated with employment and experience, which can influence income levels.
- The features **'occupation'**, **'capital.loss'**, **'marital.status'**, and **'race'** have lower importances but still contribute to the classification task to some extent.
- The features **'workclass'**, **'education'**, **'sex'**, and **'native.country'** have the lowest importances, suggesting that they have relatively less influence on the model's decision-making process.

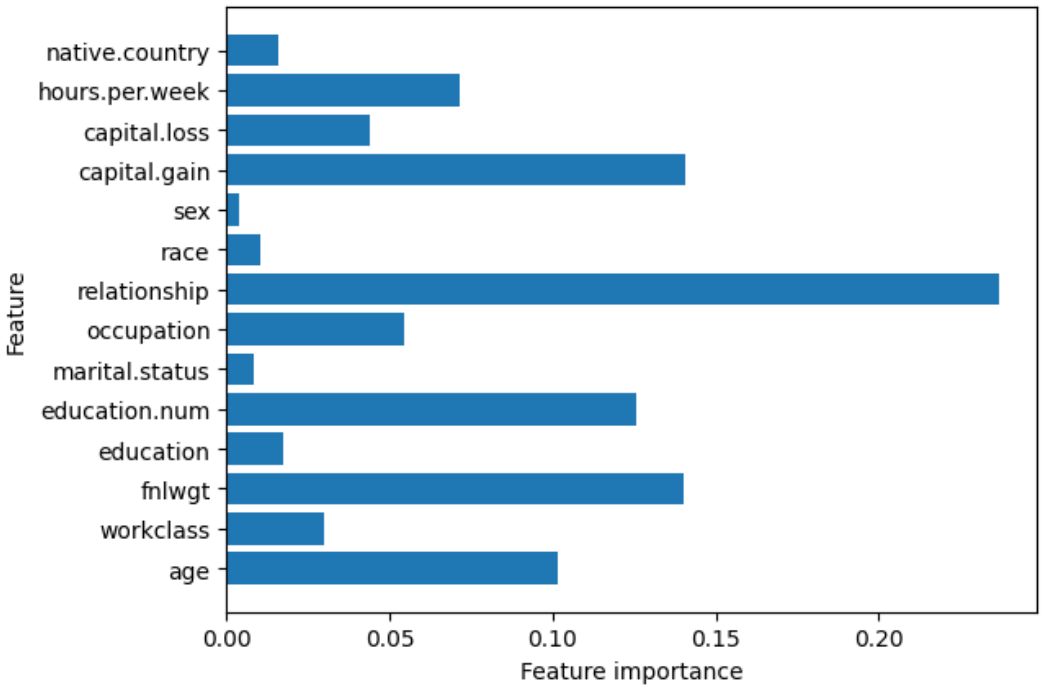

Fig. 4. Features Importances of DecisionTreeClassifier

E. GradientBoostingClassifier

The results show the combination of hyperparameters used for the GradientBoostingClassifier and their corresponding mean test scores. The hyperparameters include the learning rate and max depth in the ensemble.

- For the learning rate of 0.001, all different values of max depth (from 1 to 7) yield the same mean test score of 0.7594. This suggests that with a very low learning rate, the model does not learn effectively and performs poorly regardless of the max depth.
- As the learning rate increases to 0.01, we see an improvement in the mean test scores. The scores gradually increase from 0.8006 (max depth 1) to 0.8559 (max depth 7). This indicates that a higher learning rate allows the model to learn better and achieve higher accuracy.
- Continuing with a learning rate of 0.05, we observe a further improvement in the mean test scores. The scores range from 0.8496 (max depth 1) to 0.8676 (max depth 5), with a slight dip for max depth 6 and 7. This suggests that a learning rate of 0.05 is beneficial for the model's performance.
- With a learning rate of 0.1, the mean test scores remain relatively stable. The scores range from 0.8530 (max depth 1) to 0.8665 (max depth 5), with a slight decline for max depth 6 and 7. This indicates that increasing the learning rate beyond 0.1 does not lead to significant improvements in performance.
- Finally, using a learning rate of 0.2 and 0.5, the mean test scores vary within a similar range as before. The scores for a learning rate of 0.2 range from 0.8577 (max depth 1) to 0.8672 (max depth 4). For a learning rate of 0.5, the scores range from 0.8485 (max depth 5) to 0.8661 (max depth 2). These results suggest that the learning rate of 0.2 provides a good balance between performance and computational efficiency.

Based on the results, the best combination of hyperparameters appears to be a learning rate of 0.05, max depth of 6, and 200 estimators, which achieved a mean test score of 0.8680.

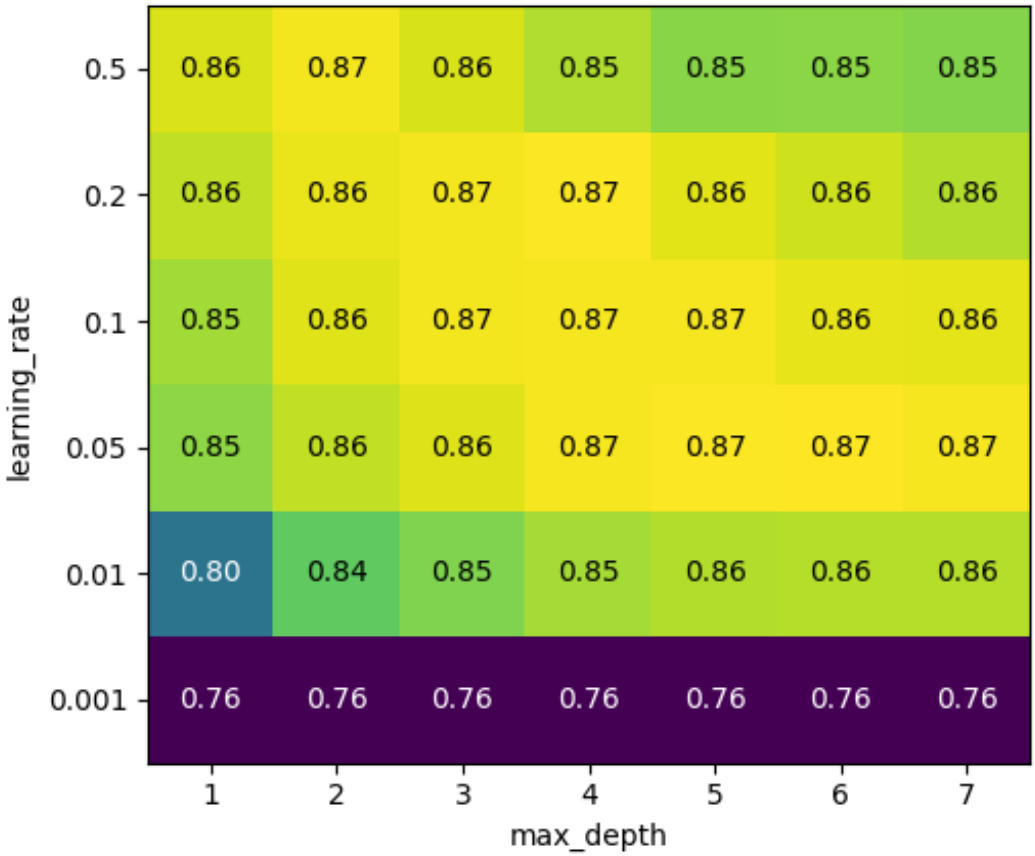

Fig. 5. GradientBoostingClassifier Hyperparameters Heapmap

F. Multi-layer Perceptron classifier

The Multi-layer Perceptron classifier was trained using different values of the regularization parameter, alpha, ranging from 10^{-5} to 10^{5}. Here are some observations based on the results:

- As the value of alpha increases, both training and test accuracies gradually decrease. This indicates that higher values of alpha lead to increased regularization, preventing the model from overfitting the training data. However, excessively large values of alpha can also cause underfitting, resulting in lower accuracies.
- The highest training accuracy is achieved with alpha equal to 10^{-2}, while the highest test accuracy is achieved with alpha equal to 0.1. This suggests that a moderate value of alpha around 0.1 is suitable for achieving good generalization performance on unseen data.
- For very small values of alpha (10^{-5} and 10^{-4}), the model exhibits slightly lower training and test accuracies compared to the optimal values. This indicates that a minimal amount of regularization is necessary to prevent overfitting and improve generalization.
- As the value of alpha increases beyond 1, the model's performance decreases significantly, with training and test accuracies dropping to around 0.75. This indicates that excessive regularization restricts the model's ability to capture complex patterns in the data, resulting in reduced accuracy.

Overall, the analysis suggests that a moderate value of alpha around 0.1 would be a good choice for the Multi-layer Perceptron classifier, as it achieves high test accuracy while preventing overfitting.

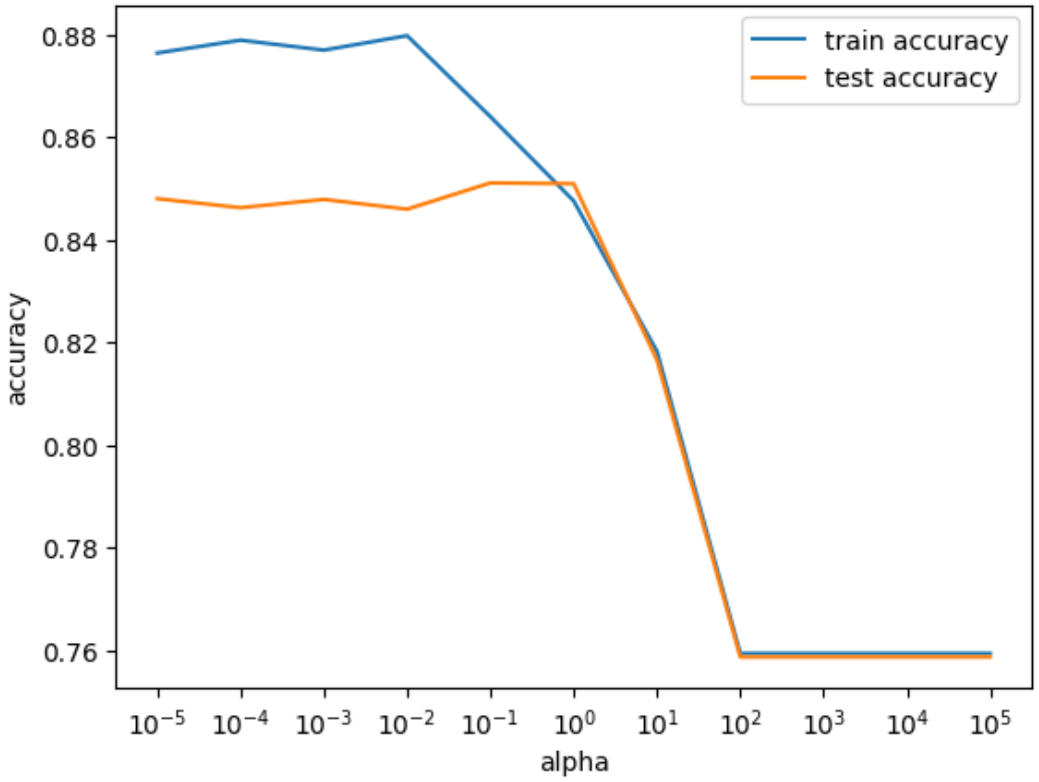

Fig. 6. Varying the Alpha Parameter

VI. Compare

KNeighborsClassifier: The KNeighborsClassifier utilizes the k-nearest neighbors algorithm to classify instances based on their proximity to labeled examples. It achieved a test accuracy of approximately 0.85, indicating reasonable performance on the dataset. The model's accuracy heavily relies on the choice of the number of neighbors (k) and the distance metric used. It is suitable for classification problems where instances with similar feature values tend to belong to the same class. The decision boundary of this classifier is flexible and can capture complex relationships between features.

LogisticRegression: Logistic regression is a linear classifier that models the probability of a binary outcome based on the input features. It achieved a test accuracy of approximately 0.85, similar to the KNeighborsClassifier, suggesting good performance. Logistic regression assumes a linear relationship between the features and the logarithm of the odds of the target variable. It is an interpretable model that provides insights into the importance and direction of the feature coefficients. This classifier is well-suited for binary classification problems and can handle both categorical and continuous features.

DecisionTreeClassifier: The DecisionTreeClassifier builds a decision tree by recursively splitting the data based on selected features. It achieved a test accuracy of approximately 0.85, similar to the previous classifiers. Decision trees offer interpretability and provide insights into the decision-making process based on the selected features. The model's performance depends on the depth of the tree and the chosen splitting criteria, such as Gini impurity or entropy. Decision trees can capture complex interactions between features but may suffer from overfitting if not properly regularized.

GradientBoostingClassifier: The GradientBoostingClassifier constructs an ensemble of decision trees sequentially, with each subsequent tree aiming to correct the errors made by the previous trees. It achieved a test accuracy of approximately 0.87, indicating improved performance compared to the previous classifiers. Gradient boosting combines weak learners to create a strong learner and handles non-linear relationships between features effectively. The model's performance is influenced by hyperparameters like the learning rate, number of trees, and maximum tree depth. Gradient boosting can handle a mixture of feature types and automatically handles missing values.

Multi-layer Perceptron Classifier: The Multi-layer Perceptron Classifier is a neural network-based classifier that consists of multiple layers of interconnected nodes (neurons). It achieved a test accuracy of approximately 0.85, similar to the previous classifiers. MLP classifiers can capture complex non-linear relationships between features but require careful tuning of hyperparameters such as the number of hidden layers, number of neurons per layer, and learning rate. They are suitable for handling large and high-dimensional datasets but may be prone to overfitting if not properly regularized.

VII. Limitations and Future Work

A. Limitations

- **Limited Dataset:** The analysis is based on a specific dataset, and the conclusions may not generalize well to other datasets. The performance of the classifiers could be influenced by the characteristics and quality of the data used for training and testing.
- **Model Selection Bias:** The analysis focuses on specific classifiers (Decision Tree, Gradient Boosting, and Multi-layer Perceptron) and does not consider other models or ensemble methods. There may be alternative models that could achieve better performance on the given dataset.
- **Hyperparameter Sensitivity:** The analysis explores the impact of specific hyperparameters (such as learning rate, max depth, and alpha) on the performance of the classifiers. However, there may be other hyperparameters or combinations thereof that were not considered, and tuning them could further improve the models' performance.
- **Assumptions and Simplifications:** The analysis assumes that the provided dataset is representative of the problem domain and that the chosen performance metrics are appropriate for evaluating the models. However, in real-world scenarios, additional considerations and domain-specific knowledge might be necessary to make informed decisions.
- **Label Encoding Limitation:** One preprocessing technique used in the analysis is Label Encoding, which assigns numerical labels to categorical features. However, Label Encoding may not be suitable for models that rely on calculating distances between features. Since the features are not equidistant from each other, Label Encoding can introduce a misleading size relationship among the categories. This can potentially impact the performance of models that depend on accurate distance calculations, such as certain distance-based clustering algorithms or k-nearest neighbors.

B. Future Work

- **Dataset Expansion:** Gathering more diverse and larger datasets related to the problem domain would help assess the models' performance and generalization capabilities across different scenarios. This could provide a more comprehensive understanding of the strengths and weaknesses of the classifiers.
- **Model Comparison:** Conducting comparative analyses involving a wider range of classifiers and ensemble methods could offer insights into the relative performance and suitability of different models for the given dataset. This would facilitate better model selection and improve the overall predictive accuracy.
- **Hyperparameter Optimization:** Employing more advanced techniques, such as grid search with cross-validation or Bayesian optimization, would enable a more thorough exploration of hyperparameter space. This could lead to improved model performance and better understanding of the optimal hyperparameter settings.
- **Feature Engineering:** Investigating different feature engineering techniques, such as dimensionality reduction, feature selection, or generating new features, could potentially improve the classifiers' performance. Careful consideration of domain-specific knowledge and feature importance analysis, as shown in the decision tree example, could guide the feature engineering process.
- **Alternative Encoding Methods:** Exploring alternative encoding methods, such as one-hot encoding or target encoding, could address the limitation of Label Encoding for models that rely on calculating distances. One-hot encoding creates binary variables for each category, while target encoding uses the target variable's information to assign numerical labels. These methods can preserve the non-equidistant nature of categorical features and provide more suitable representations for distance-based models.
- **Domain-specific Preprocessing:** Depending on the specific characteristics of the problem domain, custom preprocessing techniques can be developed to handle categorical features more effectively. This might involve domain-specific knowledge or expert guidance to transform the categorical variables in a way that better captures their inherent relationships and patterns.
- **Evaluation of Distance-based Models:** Conducting separate evaluations of distance-based models, such as clustering algorithms or k-nearest neighbors, using alternative encoding methods and comparing their performance against models trained with Label Encoding. This analysis would help quantify the impact of encoding techniques on distance calculations and identify the most suitable approach for distance-based modeling tasks.