Home Assignment:

Maor Nave – ID.313603391, Tel – 0543121144

Date: 08/05/2024

**Code Guide: ML course final EX "decision\_making\_law" and "cars" datasets: analysis, fitting the best models and evaluating.**

The code created in this research represents two designed programs that are adapted to handle the data files that were captured in the work, the first input is called decision\_making\_law and the prediction task that exists in it is classification, the second input is called cars in which the prediction task is based on regression. The program is built to receive, process and predict the information inside the csv file. This includes visualizing the data before cleaning, cleaning and refining the data and issuing new insights from it, moreover, methods were used in the code to complete missing data and issue synthetic data, and various methods for selecting relevant features for solving the prediction tasks. In addition, the program uses and compares all the models available in the skitlearn package adapted to the solution of the specific problem (classification / regression) to analyze the information. The output of the program includes the information about all the optimal models after continuously using the GridSearchCV object. The models were trained based on relevant metrics and tested based on metrics that we were asked to test with in this exercise. The program outputs the results of all tested models to json, text and png and CSV files.

1. Key Considerations:
2. The evaluation scores of the models showcased in this exercise are grounded in runs conducted on my personal computing hardware. It is crucial to acknowledge that the work environment utilized for data processing is relatively outdated, potentially affecting the resultant model scores. Furthermore, the hyperparameters chosen for this study are tuned to ensure a reasonable computational time, allowing for the exploration of all pertinent hyperparameters across the various models being assessed. The resolution to this predicament lies in systematically addressing these challenges and subsequently selecting the most optimal model based on comprehensive evaluation metrics.
3. In the pursuit of optimizing regression and classification models for this study, the negative mean squared error (neg\_mean\_squared\_error) and the f1-score were chosen as the primary scoring metrics for grid search and cross-validation. This decision was based on their effectiveness in evaluating model performance across various tasks discussed within this study.
4. It is worth noting that within the scikit-learn framework, adjustments to these metrics may be necessary to enhance the accuracy and r2 results of the models. This optimization process is crucial for ensuring the reliability and robustness of the regression analyses conducted in this research.
5. Random\_state was used in each model construction in order to obtain consistency in the results.
6. Libraries:
7. Numpy (np): Provides functions for working with arrays and mathematical computing.
8. pandas (pd): Offers data manipulation and analysis tools using DataFrames.
9. random: Generates pseudo-random numbers and is often used for random sampling and shuffling data.
10. Matplotlib (plt): A plotting library for creating static, animated, and interactive visualizations in Python, widely used for data visualization.
11. Seaborn (sns): Data visualization library built on top of Matplotlib, providing high-level functions for statistical plotting.
12. Os: Library for interacting with the operating system, managing files and directories.
13. Skitlearn.model\_selecton: Part of scikit-learn, includes tools for model selection, such as GridSearchCV for hyperparameter tuning, cross-validation, and KFold for splitting data into k-folds for validation.
14. Skitlearn.metrics: Provides evaluation metrics like R-squared (r2), F1-score, mean squared error (MSE), accuracy, and tools for creating confusion matrices to assess classification performance.
15. Yaml: a Python library for working with YAML files, often used for configuration and data serialization.
16. Joblib: provides utilities for saving and loading Python objects to and from disk, often used for model persistence.
17. Json: Library for handling JSON data format, used for data interchange and storage.
18. Skitlearn.feature\_selection: Offers methods like SelectKBest for selecting top k features and Recursive Feature Elimination (RFE) for feature ranking.
19. Skitlearn.stats: Includes statistical tools such as Variance Inflation Factor (VIF) for multicollinearity assessment and standard deviation (Sdv) for measuring variability.
20. Sdv: The GaussianCopulaSynthesizer in sdv.single\_tableof SDV library generates synthetic data by modeling joint distribution using a Gaussian copula.
21. Skitlearn.preprocessing: Provides data preprocessing techniques like StandardScaler for feature scaling and LabelEncoder for encoding categorical variables.
22. Imblearn: Library for dealing with imbalanced datasets, with techniques like Synthetic Minority Over-sampling Technique (SMOTE) for oversampling minority classes to address class imbalance.
23. Maor\_Nave\_313603391\_EX3\_models\_Final a py file assigning all models candidates and their hyperparams tuning for the relevant assignment.
24. Functions:
25. General Functions EX 1+2:
26. save\_txt\_files (data, path):

* DataFrame data and saves its information, such as data types and column names, to a text file specified by the path argument. It first creates a buffer using StringIO to capture the DataFrame information, converts it to a string, and then writes this string to the specified file. The function ensures that the file is closed properly after writing the information.

1. load\_yaml(path):

* The "load\_yaml" function loads YAML data from a specified file path using a safe loader to prevent potential security vulnerabilities. It opens the file, reads its contents, and then closes the file. Finally, it returns the loaded YAML data.

1. save\_json(path, data):

* writes the provided data to a JSON file located at the specified path. It opens the file in write mode with UTF-8 encoding, converts the data into a JSON string, and then writes this JSON string to the file. The file is automatically closed after the data is written.

1. drop\_multy\_cols(df, cols\_list):

* takes a DataFrame and a list of column names as input. It drops the columns specified in the column list from the DataFrame and returns the modified DataFrame.

1. add\_id\_col\_to\_df(df) – EX1:

* adds an ID column to the DataFrame based on the total number of IDs calculated as one-third the length of the DataFrame. It then generates ID values ranging from 1 to the total number of IDs, assigning each row an ID based on its position in the DataFrame. The modified DataFrame with the added ID column is returned. That’s because the data is organized in a way the each 3 rows are the same person sampling.

1. EDA functions:
2. check\_unique\_vals(df, paths\_dict):

* analyzes the DataFrame to find unique values in each column and saves this information to a JSON file. It creates a dictionary data\_dict\_unique to store the unique values for each column. The function then ensures that the output directory specified in paths\_dict exists or creates it if it doesn't exist already. Finally, it saves the data\_dict\_unique dictionary as a JSON file named 'unique\_data\_after\_columns\_clean.json' in the output directory using the save\_json function.

1. visualize\_corr\_and\_hist(df, output\_folders):

* Designed to visualize the correlation and histograms of features in the DataFrame and save the plots in specified output folders. It first sets up the required directory structure for saving the visualizations. The function then generates histograms for each column in the DataFrame and saves them as individual PNG files in the 'hist' folder within the main visualization folder. Next, it plots the relationship between each column and the dependent variable using count plots and saves these plots in the 'relationship' folder.
* Additionally, the function plots the correlation between each feature column and the dependent variable using box plots. For numerical columns, it plots the feature column on the x-axis and target on the y-axis, while for categorical columns, it plots target on the x-axis and the feature column on the y-axis. These correlation plots are saved in the 'corr' folder within the main visualization folder. Overall, the function provides a comprehensive visualization of data distribution, relationships, and correlations for exploratory data analysis.

1. Data manipulations functions:
2. fill\_data\_random(df) – EX1:

* addresses missing data by filling it randomly in the DataFrame. It iterates through each column, excluding the 'Feelings' column. If the column contains the value 'DATA\_EXPIRED', it randomly replaces 'DATA\_EXPIRED' instances with other unique values from that column for each unique 'ID' and that’s because each 3 rows in the df are have the same ID (same data on a specific person and different STNumber reading). If there are rare (outliers) values in a column (occurring less than 3/sum of value counts), those values are replaced with random selections from the remaining non-rare values in the column. The function then returns the DataFrame with filled missing data.

1. convert\_data\_to\_num\_cat\_vals(df) – EX1:

* converts categorical data to numerical categorical values using LabelEncoder. It first creates a LabelEncoder object. For each column in the DataFrame, if the column is 'Feelings', it transforms the data into one-hot encoded values and concatenates them with the DataFrame, dropping the original 'Feelings' column afterward. For other columns that are of type 'object' or 'bool', LabelEncoder is used to convert them into numerical categorical values. The function then separates the 'Decision' column, encodes the remaining DataFrame columns, and concatenates them with 'Decision' to maintain the original order. Finally, the modified DataFrame with numerical categorical values is returned.

1. balance\_data(ds\_df\_after\_hot\_vector) - EX1:

* uses Synthetic Minority Over-sampling Technique (SMOTE) to balance the data by generating synthetic samples for the minority class. It first separates the features and target columns from the DataFrame ds\_df\_after\_hot\_vector. Then, it initializes SMOTE with parameters such as k\_neighbors=10 and random\_state=42. SMOTE is applied to the features and target columns to generate synthetic samples, resulting in balanced data. The function returns a DataFrame ds\_df\_balanced containing the balanced data with synthetic samples added to the minority class.

1. heat\_map\_corr(ds\_df\_balanced, output\_path\_kwargs) – EX1:

* generates a heatmap of the correlation matrix for the balanced DataFrame ds\_df\_balanced. It first prepares the DataFrame by dropping the 'Feelings' columns and calculates the correlation matrix. The correlation matrix is then saved as a CSV file in the specified output path. Next, the function filters correlations based on a mean threshold and identifies columns with higher-than-mean correlations. It plots a heatmap of these high correlation values and saves it as an image. The function also sets a threshold for Variance Inflation Factor (VIF) and removes columns with VIF exceeding the threshold.

1. clean\_multy\_cat\_data(df, test=False) – EX2:

* responsible for cleaning categorical data in a DataFrame. It first converts specific columns like 'Exterior\_Color', 'Interior\_Color', 'Transmission', and others to lowercase using the .str.lower() method and that for preventing situations of duplicate value in lower and upper cases. Then, it calls the fill\_cat\_data\_random function to fill missing categorical data randomly for each specified column and with that the fill\_cat\_data\_random is mapping uneccecery duplicates values to uniquely formed ones. The function repeats this process for several categorical columns such as 'Drivetrain', 'Engine', 'State', 'Seller\_Type', and 'MPG'. If the test parameter is set to True, it also converts the 'Model' column to lowercase and fills missing data randomly for that column as well. Overall, this function enhances the data quality by preprocessing categorical data, ensuring consistency by converting text to lowercase, and addressing missing values using random filling techniques.

1. clean\_multy\_num\_data(df, test=False) – EX2:

* designed to clean numerical data in a DataFrame. It begins by determining the list of numerical columns based on whether the function is operating in a test scenario or not. For testing (test=True), the function focuses on specific rating columns related to different aspects of a product. In contrast, for non-testing scenarios (test=False), it includes additional numerical columns like 'Price'.

For each numerical column in the determined list (col\_list), the function calculates the mean and standard deviation from non-missing values. If the column is 'Price', it generates random values from a normal distribution for rows where the value is 0, simulating plausible prices. Otherwise, for rating columns, it generates random values from a normal distribution within a restricted range (1 to 5) and fills missing values with these random ratings.

Overall, this function enhances the data quality by addressing missing values in numerical columns through random value generation based on statistical measures like mean and standard deviation.

1. convert\_data\_to\_num\_cat\_vals(df) - EX2:

* converts categorical data to numerical values using the LabelEncoder from scikit-learn. It initializes a LabelEncoder object and then iterates through each column in the DataFrame. If a column has an object data type (indicating it contains categorical data), the LabelEncoder is used to transform the categorical values into numerical labels. The transformed DataFrame is then returned with categorical data converted to numerical values.

1. gen\_syn\_data(df) – EX2:

* This function gen\_syn\_data(df) generates synthetic data using the Gaussian Copula Synthesizer from the SDV (Synthetic Data Vault) library. It begins by creating a SingleTableMetadata object and detecting metadata from the input DataFrame df. Then, it initializes a GaussianCopulaSynthesizer object using the detected metadata and fits the synthesizer to the input data. The function generates synthetic data with a specified number of rows (here, 5000) using the sample method of the synthesizer. It concatenates the original DataFrame df with the synthetic data to create a combined DataFrame full\_data.

Finally, the combined DataFrame is saved as a CSV file named 'cars\_clean\_with\_syn\_data.csv' in the 'input' directory, and the function returns this combined DataFrame.

1. Main function 🡪 walkthrough by EX:

Starting the program, a configuration file is employed to determine whether a particular model has undergone training or if it necessitates retraining, chosen upon user-defined parameters specified in GridSearchCV. if retraining is necessary for each model, an algorithm is initiated to ascertain optimal parameters, focusing on those hyperparameters deemed most impactful and pertinent. Additionally, the program commences with a data preprocessing stage, segregating information into training and testing sets as dictated by the exercise criteria.

1. EX1 🡪 Classification:

* Load configuration settings from a YAML file named 'Maor\_Nave\_313603391\_EX3\_Config\_Final.yaml':

This step reads the configuration file that contains various settings and flags to control the execution of different functions and processes.

* Load and perform Exploratory Data Analysis (EDA) on the data stored in the CSV file 'input\decision\_making\_law.csv':

The data is loaded from the specified CSV file, and exploratory data analysis techniques are applied to gain insights into the data distribution, missing values, outliers, etc.

* Configure output paths based on the loaded configuration settings:

The output paths for saving visualizations, cleaned data, model results, and other outputs are determined based on the configuration file.

* Check for unique values in the dataset and save the results if specified in the configuration (config['EX1']['functions']['check\_unique\_vals']):

If enabled in the configuration, this step checks for unique values in each column of the dataset and saves the results to a JSON file for further analysis or validation.

* Visualize correlations and histograms for features and save the plots if specified in the configuration (config['EX1']['functions']['visualize\_corr\_and\_hist']):

If enabled, this step generates correlation matrices and histograms to visualize relationships between features and saves these plots in the specified output folder.

* Clean and preprocess the data if specified in the configuration (config['EX1']['functions']['clean\_and\_save\_data']):

If the cleaning and preprocessing flag is set in the configuration, several data cleaning steps are performed, including adding an ID column, filling missing data, converting categorical data to numerical values, balancing the data, and removing highly correlated features.

* Extract model parameters and split the data into training and testing sets:

This step involves extracting model parameters specified in the configuration file and splitting the dataset into training and testing sets for model training and evaluation.

* Scale the data using StandardScaler and perform hyperparameter tuning using GridSearchCV to find the best model based on F1 score:

The data is scaled using StandardScaler, and hyperparameter tuning is performed using GridSearchCV to find the best model configuration based on the F1 score metric.

The method for finding optimal hyperparameter values involves an iterative process of passing ordered lists to each hyperparameter with typical values relevant to the specific task. Initially, default values are used, and further tuning is done by observing tendencies and performance metrics, adding new values to the hyperparameter lists until optimal values are determined. During these iterations with GridSearchCV, cross-validation is not performed, with hyperparameters set to 'None', allowing for a focused exploration of configurations to find the best model hyperparameters.

* Evaluate the best model's performance on the test set and generate a confusion matrix:

The best model obtained from hyperparameter tuning is evaluated on the test set, and performance metrics such as accuracy, F1 score, and confusion matrix are calculated and saved for analysis.

* Save model accuracies and confusion matrices to CSV files in the specified output directory:

The model accuracies, along with the confusion matrices for each model, are saved in CSV format for documentation and analysis purposes.

* Train with K-Folds Cross Validation and evaluate accuracy and F1 scores:

This section of the code checks if the configuration flag for training with K-Folds Cross Validation is set to True (config['EX1']['functions']['train\_cv']).

If the flag is True, it initializes a Random Forest Classifier chosen\_model with specific hyperparameters that are presumably chosen as the best metric model based on F1 score and accuracy.

The data is then scaled using StandardScaler to normalize the features.

Recursive Feature Elimination (RFE) is applied to select the most important features using the chosen\_model.

K-Folds Cross Validation with 5 splits (n\_splits=5) is performed to train and evaluate the model.

Cross-validated predictions (cv\_pred) are generated using the trained model on the scaled and selected features (X\_scaled[selected\_features]) and the target variable (y).

F1 scores are calculated both for individual classes (average=None) and weighted average (average='weighted') to assess overall model performance.

Accuracy is also computed using accuracy\_score to evaluate classification accuracy.

The results, including F1 scores, weighted F1 score, and accuracy, are stored in a DataFrame (cv\_scores\_df) and saved to a CSV file named 'best\_RandomForestClassifier\_Kfold.csv' in the specified output directory (output\_path\_kwargs['path\_main\_model']).

* Train without the 'STNumber' column and test accuracy and F1 scores:

This section of the code checks if the configuration flag for training without the 'STNumber' column is set to True (config['EX1']['functions']['train\_cv\_no\_ST']).

If the flag is True, the same Random Forest Classifier chosen\_model and data preprocessing steps are applied as in the previous section.

However, in this case, after feature selection, the 'STNumber' column is removed from the selected features if it exists.

K-Folds Cross Validation, predictions, and evaluation metrics computation are performed similarly to the previous section.

The results are saved to a CSV file named 'best\_RandomForestClassifier\_Kfold\_noSTNumber.csv' in the specified output directory.

1. EX2 🡪 Regression:

* Load and Preprocess Data:

The code first loads a CSV file named 'cars.csv' into a pandas DataFrame (cars\_df).

It then loads the configuration file ('Maor\_Nave\_313603391\_EX3\_Config\_Final.yaml') using load\_yaml.

* Output path settings are extracted from the configuration file (output\_path\_kwargs).
* Clean and Prepare Data:

If specified in the configuration (config['EX2']['functions']['clean\_and\_save\_data']), data cleaning processes are initiated.

Certain columns ('VIN', 'Consumer\_Review\_#', 'Stock\_#') are dropped from the DataFrame (cars\_df) using drop\_multy\_cols.

Categorical data is cleaned using clean\_multy\_cat\_data, and the resulting DataFrame is saved to 'cars\_clean\_cat.csv'.

The cleaned categorical and numerical data are further processed using clean\_multy\_num\_data and saved to 'cars\_clean\_cat\_num.csv'.

* Generate Synthetic Data (Optional):

If enabled in the configuration (config['EX2']['functions']['create\_syn\_data']), synthetic data is generated using gen\_syn\_data based on the cleaned and processed data.

* Load Model Configuration and Define Models:

Model configurations are loaded from a custom module Maor\_Nave\_313603391\_EX3\_models\_Final.models\_ex2, as specified in the configuration file.

The features and target variable ('Price') are extracted from the preprocessed data (cars\_df\_after\_label\_encoding).

* Train and Evaluate Models:

The code iterates over each model specified in the configurations and performs hyperparameter tuning using GridSearchCV.

The method for finding optimal hyperparameter values involves an iterative process of passing ordered lists to each hyperparameter with typical values relevant to the specific task. Initially, default values are used, and further tuning is done by observing tendencies and performance metrics, adding new values to the hyperparameter lists until optimal values are determined. During these iterations with GridSearchCV, cross-validation is not performed, with hyperparameters set to 'None', allowing for a focused exploration of configurations to find the best model hyperparameters.

Feature selection methods such as SelectKBest or Recursive Feature Elimination (RFE) are applied based on the model type, as indicated in the configuration.

The best-performing model is selected based on the negative mean squared error metric ('neg\_mean\_squared\_error') and saved using joblib.

Model performance metrics such as R-squared scores (r2\_train, r2\_test) are calculated and stored in a DataFrame (report\_df).

* Cross-Validation and Model Testing:

Optionally, if specified in the configuration (config['EX2']['functions']['train\_cv']), the best model is loaded (e.g., 'trained\_model\_best\_XGBRegressor.joblib').

The data is scaled using StandardScaler and prepared for cross-validated predictions.

Cross-validated predictions are made, and the R-squared score is computed for each fold.

The R-squared scores are saved to 'best\_XGBREG\_Kfold.csv' or 'best\_XGBREG\_Kfold\_cars\_evaluate\_r2.csv' based on the context.

* Prediction on New Data:

If enabled in the configuration (config['EX2']['functions']['test\_on\_new\_data']), new data ('cars\_evaluate.csv') is preprocessed similarly to the training data.

The best model is loaded, and predictions are made on the new data to estimate prices.

Predicted prices are saved to 'cars\_evaluate\_after\_price.csv'.

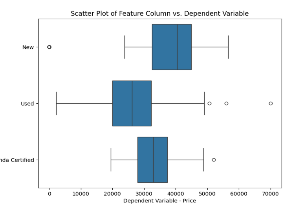
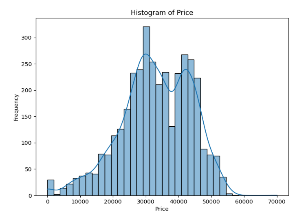
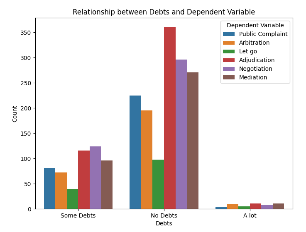
* Check R2 Scores for Evaluation Data:

If specified in the configuration (config['EX2']['functions']['check\_new\_ev']), R2 scores are computed for evaluation data ('cars\_evaluate\_after\_price.csv').

The evaluation data is preprocessed, scaled, and prepared for cross-validated predictions.

Cross-validated R2 scores are calculated using K-Folds Cross Validation and saved to 'best\_XGBREG\_Kfold\_cars\_evaluate\_r2.csv'.

1. Models evaluation:

* In this report section, I'll outline the process of generating predictions on test data and evaluating model performance using metrics like accuracy and R-squared score (R2). These metrics help assess the predictive accuracy and goodness of fit for each model. It's worth noting that the choice of objective function during model training varies based on the nature of the predictive task (classification or regression). For classification tasks, the focus is on maximizing F1 scores across all classes, while regression tasks aim to minimize Mean Squared Error (MSE) between predicted and actual data.
* This section covers the data from various models and their respective storage locations in different directories, including the ouput files from exploratory data analysis (EDA) processes. Towards the end of this section, there's a comparative table presenting the performance of different models across various prediction tasks.
* Generated data (after cleaning and scaling) location:
* input\cars\_clean\_cat\_num.csv
* input\cars\_clean\_with\_syn\_data.csv
* input\cars\_ev\_clean\_cat\_num.csv
* input\cars\_evaluate\_after\_price.csv 🡪 outputted csv file for ex2 last section.
* input\decision\_making\_law\_clean.csv
* visualization data location:
* output\vis\corr 🡪 all visualizations outputted to check correlation between features to target values.
* output\vis\df\_info 🡪 each input df raw data information.
* output\vis\hist 🡪 data histograms representations of each column in each dataset to understand data scattering and balance.
* output\vis\relationship 🡪 visualizations showing the distribution of the data according to the relationship they have in the target column space.
* תמונה שמכילה צילום מסך, דפוס, צבעוני, קו

  התיאור נוצר באופן אוטומטיoutput\corr 🡪 generate visualizations and create a CSV file listing the features with the highest VIF (Variance Inflation Factor) values. High VIF values indicate significant multicollinearity, suggesting that some columns in the dataset should be addressed during data cleaning. These columns will be removed to mitigate multicollinearity issues.
* Best models scores locations:
* output\models\models\_r2\_finding\_the\_best\_model\_v3.csv
* output\models\model\_accuracies\_report\_v18\_f1\_score\_searching for best\_model\_v5.csv
* Best models locations:
* output\models\ trained\_model\_best\_XGBRegressor.joblib – best model is noted for example the rest of models are saved as well.
* Confusion matrices:
* Each confusion matrix is outputted to train and test sets.
* output\models\confusion\_matrix\_Random\_Forest\_train.csv
* output\models\confusion\_matrix\_Random\_Forest\_test.csv
* Best models cv train and scores location:
* output\models\best\_RandomForestClassifier\_Kfold.csv
* output\models\best\_RandomForestClassifier\_Kfold\_noSTNumber.csv
* output\models\best\_XGBREG\_Kfold.csv
* output\models\best\_XGBREG\_Kfold\_cars\_evaluate\_r2.csv
* Classification results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Best\_Model | ACC\_train | ACC\_test | F1\_train | F1\_test |
| Random\_Forest | 1 | 0.57 | [1. 1. 1. 1. 1. 1.] | [0.44776119 0.58823529 0.85393258 0.43373494 0.4952381 0.62222222] |
| Random\_Forest\_with\_cv | - | 0.57 | - | |  | | --- | | [0.46125461254612543 | | 0.5792811839323466 | | 0.7607476635514019 | | 0.4444444444444444 | | 0.4541577825159915 | | 0.5256410256410257] | |
| Random\_Forest\_with\_cv\_  No\_STNimber | - | 0.51 | - | |  | | --- | | [0.4339796860572484 | | 0.5379023883696781 | | 0.7511737089201879 | | 0.4157955865272938 | | 0.45100105374077976 | | 0.49411764705882355] | |

* Regression results:

|  |  |  |
| --- | --- | --- |
| Best\_Model | R2\_train | R2\_test |
| XGBRegressor | 0.97 | 0.93 |
| XGBRegressor\_with\_cv | - | 0.95 |
| XGBRegressor\_with\_cv\_on evaluation\_data\_after\_predictions | - | 0.99 |

* Conclusions:
* Classification:
* The best-performing model, both in terms of accuracy and F1 score for the classification task, is the Random Forest model. This model excels due to its simplicity, making it effective in handling imbalanced data by transferring information effectively. It also exhibits strong performance in capturing the complex relationships among different features in the dataset, especially in predicting the output for each record accurately.
* The models trained specifically for the classification problem include a range of algorithms and techniques aimed at optimizing classification accuracy. These models undergo rigorous training and evaluation processes to identify the most suitable one for the given dataset and task. The models are: Logistic\_Regression, Random\_Forest, SVM, XGBoost, AdaBoost Decision\_Tree, KNN, SGDClassifier, MLPclassifier.
* Upon additional training using the KFolds method, it's noticeable that the test values remain relatively stable. This stability can be attributed to the model's ability to generalize well to new data, indicating a robust and reliable predictive capacity that extends beyond the training data.
* Analysis of the F1 scores reveals that the model performs exceptionally well in identifying certain classes, particularly classes 2, 3, and 5, showcasing a high level of precision and recall for these specific categories. This performance disparity among classes underscores the complexity and nuances within the dataset, especially concerning the diverse configurations present in the feature space.
* Examining the impact of the STNumber column on model performance unveils intriguing insights. The absence of information in this column significantly affects the model's ability to predict accurately, leading to noticeable drops in F1 scores and accuracy metrics. This underscores the importance of feature selection and the role of essential features like STNumber in enhancing model performance.
* The selection of STNumber as a crucial feature in the Recursive Feature Elimination (RFE) method for model training aligns with the observed results post-column removal. This reaffirms the significance of feature engineering and the iterative process of refining the feature set to improve model performance and interpretability.
* Despite employing techniques like balancing data using SMOTE, discrepancies between training and test performance may persist due to the inherent complexities within the feature space. These complexities arise from the intricate combinations of features that influence the model's ability to accurately predict outcomes, highlighting the ongoing challenges in achieving optimal model generalization and performance.
* Regression:
* The XGBRegressor stands out as the top-performing model based on R-squared (R2) metric for regression tasks. Its effectiveness stems from the gradient boosting technique, adept at handling non-linear relationships within data. This iterative approach enhances predictive accuracy by addressing model weaknesses, making it robust for optimizing R2 scores.
* In the context of regression, various models were trained and evaluated and are as follows - Linear\_Regression, Lasso\_Regression, Ridge\_Regression, SVR, Gradient\_Boosting, Random\_Forest\_Regressor, KNN\_Regressor, SGDRegressor, XGBRegressor
* Upon additional training using the KFolds method, minimal changes were observed in test values. This stability indicates that the model has effectively captured relevant patterns from the data, minimizing overfitting.
* Analysis of R2 values reveals optimal model convergence on the dataset. The absence of significant deviations post-CV demonstrates the model's consistency and reliability in predicting outcomes.
* Notably, R2 values tend to increase when applying the model to new datasets. This improvement is attributed to data cleaning processes, pipeline adjustments tailored to the dataset, and appropriate data completion strategies (e.g., ensuring normal distributions and removing duplicate features). Furthermore, the utilization of feature selection methods like SelectKbest and RFE contributed positively to performance, resulting in enhanced R2 scores on new data.

1. Project Folders:
2. input 🡪 all data sets including after cleaning and scaling.
3. output 🡪 stores all models results and plots.
4. Code\_guide.docx 🡪 this file.
5. Maor\_Nave\_313603391\_EX3\_1\_ML\_Final.py 🡪 main python program file EX1.
6. Maor\_Nave\_313603391\_EX3\_2\_ML\_Final.py 🡪 main python program file EX2.
7. Maor\_Nave\_313603391\_EX3\_models\_Final.py 🡪 main python models configuration file.
8. Maor\_Nave\_313603391\_EX3\_Config\_Final.yaml 🡪 main config file for functions activations and params.
9. Bibliography:
10. <https://pandas.pydata.org/>
11. <https://matplotlib.org/>
12. <https://scikit-learn.org/stable/supervised_learning.html>
13. <https://scikit-learn.org/>
14. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html>
15. <https://xgboost.readthedocs.io/en/stable/parameter.html#general-parameters>
16. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>
17. <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>
18. <https://scikit-learn.org/stable/modules/model_evaluation.html>