|  |  |
| --- | --- |
|  |  |
| Machine LearningProject |  |
|  |  |
|  | By: |
|  | Belal Khaled |

**Table Of Contents:**

**Topics: Page:**

## **1. Introduction …………………………………….. 3**

## **2. Part 1 …………………………………………… 4-6**

## **Data Visualization ……………………………………. 4**

## **Correlations, Patterns …………………………………. 5**

## **Outlier ……………………………………………………. 6**

## **3. Part 2 …………………………………………………… 7-8**

## **Forward Selection Strategy ……………………………. 7**

## **Identifying The Best Model ……………………………… 8**

## **4. Part 3 ……………………………………………………. 9-22**

## **Classifications:**

## **Support Vector Machine …………………………….. 9-12**

* ***Decision Tree ………………………………………. 13-16***
* ***K Nearest Neighbor …………………………………. 17-20***
* ***Notes ……………………………………………………. 21***
* ***Identifying The Best Classifier ……………………….. 22***

***5.* References ………………………………………………… 23**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Introduction: **In this report, we will talk about data visualization using Scatter Plot Matrix,**  **regression, and classification methods such as SVM, DT, and KNN:**  **Part 1: Data Visualization using Scatter Plot Matrix:**  **In this section, we will use The California Housing Prices dataset.**  **Then, we will explore and visualize the data, trying to find correlations among**  **features, outliers, or any pattern in the data.**  **Part 2: Regression:**  **We will use the same data (The California Housing Prices dataset)**  **to construct regression models considering the forward selection strategy, and**  **for each model, we will report the R2 score and Mean Squared Error.**  **Then, we will identify the best model.**  **Part 3:** **Classification (SVM, Decision Trees, K-Nearest Neighbors):**    **We will classify our dataset based on the 10-Fold Cross Validation and**  **The Hold-Out Method.**  **Then, we will report the confusion matrix and compare between them.**    **We will explain any preprocessing techniques that we have used.**    **Finally, we will identify the best classifier.** | | | | |
|  | PART 1:**Data Visualization :****Code:**  **import pandas as pd**  **import matplotlib.pyplot as plt**  **data1 = pd.read\_csv("housingdata5.csv")**  **print(data1.info())**  **print(data1.describe())**  **#Visualize data using scatter plot matrix**  **pd.plotting.scatter\_matrix (data1, figsize = (26,24), diagonal = '')**  **plt.show()** **Output:**    **latitude**  **longitude**  **housing\_median\_age**  **median\_house\_**  **value**  **median\_income**  **households**  **Population**    **total\_rooms** PART 1 NOTES (1): **data1.corr()** **Correlations, Patterns:** **Indicates that as one variable increases,**  **the other variable tends to increase as well.**  **Positive Correlation:**    **Very Weak to Weak: Moderate to Strong:**    **No Relation**  **No Relation**  **Strong to Very Strong:**    **Positive**  **Linear Relationship**  **Indicates that as one variable increases,**  **the other variable tends to decrease**.  **Negative Correlation:**  **Very Weak to Weak: Moderate to Strong:**    **There is no**  **‘Moderate Negative Correlation’**  **in this scatter plot matrix.**  **No Relation**  **Strong to Very Strong:**  **Negative**  **Linear Relationship** PART 1 NOTES (2):**Outlier ( ):** |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | PART 2:**Regression:****Code:** **import pandas as pd**  **from sklearn.linear\_model import LinearRegression**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn.metrics import r2\_score, mean\_squared\_error**  **import numpy as np**  **data2 = pd.read\_csv("housingdata5.csv")**  **X = data2[['housing\_median\_age','total\_rooms','population','households','median\_income']]**  **y = data2['median\_house\_value']**  **# Split the data into training and testing sets**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)**  **(test\_size=0.3 means that 30% of the data will be used for testing, and the remaining 70% will be used for training the model)** **Using Forward Selection Strategy:** **selected\_features = [] 🡺 (Initialize an empty list to store selected features)**  **model\_performance = {} 🡺 (create an empty dictionary to store model performance(R2 ,MSE))**  **#Loop through each feature**  **for feature in X\_train.columns: 🡺 (.columns: is an attribute of a Pandas DataFrame object that returns**  **the column labels (names) of the DataFrame)**  **#Add the feature to the selected features list**  **selected\_features.append(feature)**  **#Train the model using the selected features**  **model = linear\_model.LinearRegression()**  **model.fit(X\_train[selected\_features], y\_train)**  **#Make predictions on the test set**  **y\_pred = model.predict(X\_test[selected\_features])**  **#Calculate the R-squared score and mean squared error**  **r2 = r2\_score(y\_test, y\_pred)**  **mse = mean\_squared\_error(y\_test, y\_pred)**  **#Store the model performance in the dictionary**  **model\_performance[", ".join(selected\_features)] = (r2, mse)**  **#Remove the last added feature if it does not improve the model performance**  **if len(selected\_features) > 1 and r2 < max(model\_performance.values())[0]:**  **selected\_features.pop()**  **(max(model\_performance.values())[0]: This returns the maximum value among the performance metrics. Since we are interested in the R-squared score, it will return the maximum R-squared score)** **R2 score and Mean Squared Error for each model.****Then identifying the best model :** **#Initialize variables to store the best model's information**  **best\_features = ""**  **best\_r2 = 0**  **best\_mse = 1000**    **#Loop through each model performance**  **for features, performance in model\_performance.items():**  **#Print the selected features and their corresponding model performance**  **print("Selected Features:", features)**  **print("R-squared Score:", performance[0])**  **print("Mean Squared Error:", performance[1])**  **print('-------------------------------------------------------------------------------------------')**  **#Check if the current model has better performance than the current best model**  **if performance[0] > best\_r2:**  **best\_features = features**  **best\_r2 = performance[0]**  **best\_mse = performance[1]**  **#Print the best model's information**  **print("Best Model is:")**  **print("Selected Features:", best\_features)**  **print("R-squared Score:", best\_r2)**  **print("Mean Squared Error:", best\_mse)** **Output:**  **Selected Features: housing\_median\_age**  **R-squared Score: 0.009860112758160677**  **Mean Squared Error: 12980042026.164888**  **-------------------------------------------------------------------------------------------**  **Selected Features: housing\_median\_age, total\_rooms**  **R-squared Score: 0.04474694242349009**  **Mean Squared Error: 12522700067.669453**  **-------------------------------------------------------------------------------------------**  **Selected Features: housing\_median\_age, total\_rooms, population**  **R-squared Score: 0.11002925498580662**  **Mean Squared Error: 11666894568.32274**  **-------------------------------------------------------------------------------------------**  **Selected Features: housing\_median\_age, total\_rooms, population, households**  **R-squared Score: 0.11003928183559408**  **Mean Squared Error: 11666763123.328646**  **-------------------------------------------------------------------------------------------**  **Selected Features: housing\_median\_age, total\_rooms, population, households, median\_income**  **R-squared Score: 0.5394759349996792**  **Mean Squared Error: 6037148684.531719**  **-------------------------------------------------------------------------------------------**  **Best Model is:**  **Selected Features: housing\_median\_age, total\_rooms, population, households, median\_income**  **R-squared Score: 0.5394759349996792**  **Mean Squared Error: 6037148684.531719** PART 3:**Classification:****Support Vector Machine (SVM):****Code:** **import pandas as pd**  **from sklearn.preprocessing import LabelEncoder**  **from sklearn.metrics import classification\_report, confusion\_matrix**, **accuracy\_score** |  |  |
|  | **dataset = pd.read\_csv("online\_shoppers\_intention\_Dataset 5.csv")**  **print(dataset.info())**  **print(dataset.shape)🡺 (12330, 18).**  **dataset.head()**  **#We will delete an unnecessary attribute like the month column which has a data type object**  **dataset = dataset.drop(['Month'], axis= 1)🡺** (**The axis=1 it indicates that the column should be dropped).**  **print(dataset.shape) (12330, 17).**  **dataset.head()** **Data Preprocessing:****Handling Missing Data:** **#Check for null values in data**  **nullcount = dataset.isnull().sum()**  **print('Total number of null values in dataset:', nullcount.sum())🡺 Total number of null values in dataset: 0** **Converting String Value To Integer Type:** **dataset['VisitorType'] = LabelEncoder().fit\_transform(dataset['VisitorType'])**  **dataset['Weekend'] = LabelEncoder().fit\_transform(dataset['Weekend'])**  **dataset['Revenue'] = LabelEncoder().fit\_transform(dataset['Revenue'])**  **dataset.head()** **Explaining:**  * **There is no missing value to handle.** * **We can’t have text in our data if we’re going to run any kind of model on it. So before we can run a model, we need to make this data ready for the model.**   **And to convert this kind of categorical text data into model-understandable numerical data,**  **we used the LabelEncoder class.** **Separating Features And Labels:** **#Assign values to the X and y variables:**  **X = dataset.iloc[:, :-1]🡺** (**X will contain all rows and all columns of the dataset except for the last column).**  **y = dataset.iloc[:, -1] 🡺 (y will contain all rows but only the values from the last column of the dataset).**  **print(X.shape)🡺 (12330, 16).**  **X.head()**  **y.head()** **Data Standardization:** **#Scale the data to be between -1 and 1**  **from sklearn.preprocessing import StandardScaler**  **scaler = StandardScaler()**  **X = scaler.fit\_transform(X) 🡺 The fit\_transform() method performs both fitting and transforming in a single step.** **10-Fold Cross Validation:** **from sklearn.svm import SVC**  **from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict**  **svc = SVC(kernel='linear')**  **scores = cross\_val\_score(svc, X, y, cv=10)**  **#Perform cross-validation predictions**  **y\_pred = cross\_val\_predict(svc, X, y, cv=10)**  **Confusion\_Matrix:**  **[[9978 279]**  **[1171 902]]**  **True Positives(TP) = 9978**  **True Negatives(TN) = 902**  **False Positives(FP) = 279**  **False Negatives(FN) = 1171**  **#Compute and print confusion matrix**  **cm = confusion\_matrix(y, y\_pred)**  **print('Confusion\_Matrix\n\n', cm)**  **print('\nTrue Positives(TP) = ' , cm[0,0])**  **print('\nTrue Negatives(TN) = ' , cm[1,1])**  **print('\nFalse Positives(FP) = ' , cm[0,1])**  **print('\nFalse Negatives(FN) = ' , cm[1,0])**  **print('-------------------------------------------------------')**  **print(classification\_report(y, y\_pred))**  **Score: [0.8783455 0.87915653 0.87672344 0.8864558 0.8864558 0.87996756**  **0.88807786 0.88726683 0.88077859 0.88077859]**  **Average Score: 0.8824006488240066**  **print('-------------------------------------------------------')**  **print("Score:", scores)**  **print("Average Score:", scores.mean())**  **print('-------------------------------------------------------')**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Classes** | **precision** | **recall** | **f1-score** | **support** | | **0 (TP,FP)** | **0.89** | **0.97** | **0.93** | **10257** | | **1 (TN,FN)** | **0.76** | **0.44** | **0.55** | **2073** |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **accuracy** |  |  | **0.88** | **12330** | | **macro avg** | **0.83** | **0.70** | **0.74** | **12330** | | **weighted avg** | **0.87** | **0.88** | **0.87** | **12330** |    **Splitting Dataset Into Training Set And Testing Set:** **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)**  **(test\_size=0.2 means that 20% of the data will be used for testing,**  **and the remaining 80% will be used for training the model)** **Hold-Out Method:** **#Standardize features by removing mean and scaling to unit variance:**  **scaler = StandardScaler()**  **scaler.fit(X\_train)**  **X\_train = scaler.transform(X\_train)**  **X\_test = scaler.transform(X\_test)**  **#Train the SVM classifier**  **classifier = SVC(kernel='linear')**  **classifier.fit(X\_train, y\_train)**  **#Predict on the test set**  **Confusion\_Matrix:**  **[[2010 59]**  **[ 220 177]]**  **True Positives(TP) = 2010**  **True Negatives(TN) = 177**  **False Positives(FP) = 59**  **False Negatives(FN) = 220**  **accuracy= 0.8868613138686131**  **y\_pred = classifier.predict(X\_test)**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print('Confusion\_Matrix\n\n', cm)**  **print('\nTrue Positives(TP) = ' , cm[0,0])**  **print('\nTrue Negatives(TN) = ' , cm[1,1])**  **print('\nFalse Positives(FP) = ' , cm[0,1])**  **print('\nFalse Negatives(FN) = ' , cm[1,0])**  **print('-------------------------------------------------------')**  **print(classification\_report(y\_test, y\_pred))**  **print('-------------------------------------------------------')**  **print("accuracy=", accuracy\_score(y\_test, y\_pred))**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Classes** | **precision** | **recall** | **f1-score** | **support** | | **0 (TP,FP)** | **0.90** | **0.97** | **0.94** | **2069** | | **1 (TN,FN)** | **0.75** | **0.45** | **0.56** | **397** |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **accuracy** |  |  | **0.89** | **2466** | | **macro avg** | **0.83** | **0.71** | **0.75** | **2466** | | **weighted avg** | **0.88** | **0.89** | **0.87** | **2466** |  **Comparing 10-Fold Cross Validation And Hold-Out Method:** **By comparing the classification results obtained from :**  **10-Fold Cross-Validation and The Holdout Method, we can observe the following:**  **10-Fold Cross Validation:**  **Accuracy: 0.88**  **Precision: Class 0 - 0.89, Class 1 - 0.76**  **Recall: Class 0 - 0.97, Class 1 - 0.44**  **F1-score: Class 0 - 0.93, Class 1 - 0.55**  **Holdout Validation:**  **Accuracy: 0.89**  **Precision: Class 0 - 0.90, Class 1 - 0.75**  **Recall: Class 0 - 0.97, Class 1 - 0.45**  **F1-score: Class 0 - 0.94, Class 1 - 0.56**  **Comparing the results, we can observe that:**  **The accuracy of the holdout method (0.89) is slightly higher than**  **that of 10-fold cross-validation (0.88).**  **The precision for class 0 is similar in both methods (0.89 in 10-fold CV and 0.90 in holdout), while the precision for class 1 is slightly higher in the holdout method (0.75) compared to 10-fold CV (0.76).**  **The recall for class 0 is the same in both methods (0.97),**  **but the recall for class 1 is slightly higher in the holdout method (0.45) compared to 10-fold CV (0.44).**  **The F1-score for class 0 is similar in both methods (0.93 in 10-fold CV and 0.94 in holdout), while the F1-score for class 1 is slightly higher in the holdout method (0.56) compared to 10-fold CV (0.55).**  **Overall, the performance metrics between the two methods are relatively close, with slight variations in precision, recall, and F1-score. The holdout method shows a slightly better performance in terms of accuracy, precision, and recall for class 1. However, it's important to note that these differences might be influenced by the specific train-test split in the holdout method, and 10-fold cross-validation provides a more comprehensive evaluation by considering multiple train-test splits.** PART 3:**Classification:** ***Decision Tree(DT):*** **Code:** **import pandas as pd**  **from sklearn.preprocessing import LabelEncoder**  **from sklearn.metrics import classification\_report, confusion\_matrix,** **accuracy\_score**  **dataset = pd.read\_csv("online\_shoppers\_intention\_Dataset 5.csv")**  **print(dataset.info())**  **print(dataset.shape)🡺 (12330, 18).**  **dataset.head()**  **#We will delete an unnecessary attribute like the month column which has a data type object**  **dataset = dataset.drop(['Month'], axis= 1)🡺** (**The axis=1 it indicates that the column should be dropped).**  **print(dataset.shape) (12330, 17).**  **dataset.head()** **Data Preprocessing:****Handling Missing Data:** **#Check for null values in data**  **nullcount = dataset.isnull().sum()**  **print('Total number of null values in dataset:', nullcount.sum()) 🡺 Total number of null values in dataset: 0** **Converting String Value To Integer Type:** **dataset['VisitorType'] = LabelEncoder().fit\_transform(dataset['VisitorType'])**  **dataset['Weekend'] = LabelEncoder().fit\_transform(dataset['Weekend'])**  **dataset['Revenue'] = LabelEncoder().fit\_transform(dataset['Revenue'])**  **dataset.head()** **Explaining:**  * **There is no missing value to handle.** * **We can’t have text in our data if we’re going to run any kind of model on it. So before we can run a model, we need to make this data ready for the model.**   **And to convert this kind of categorical text data into model-understandable numerical data,**  **we used the LabelEncoder class.** **Separating Features And Labels:** **#Assign values to the X and y variables:**  **X = dataset.iloc[:, :-1]🡺** (**X will contain all rows and all columns of the dataset except for the last column).**  **y = dataset.iloc[:, -1] 🡺 (y will contain all rows but only the values from the last column of the dataset).**  **print(X.shape)🡺 (12330, 16).**  **X.head()**  **y.head()** **Data Standardization:** **#Scale the data to be between -1 and 1**  **from sklearn.preprocessing import StandardScaler**  **scaler = StandardScaler()**  **X = scaler.fit\_transform(X)🡺 The fit\_transform() method performs both fitting and transforming in a single step.** **10-Fold Cross Validation:** **from sklearn.tree import DecisionTreeClassifier**  **from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict**  **#Create a Decision Tree classifier**  **classifier = DecisionTreeClassifier()**  **#Perform 10-fold cross-validation**  **scores = cross\_val\_score(classifier, X, y, cv=10)**  **#Perform cross-validation predictions**  **Confusion\_Matrix:**  **[[10243 14]**  **[ 10 2063]]**  **True Positives(TP) = 10243**  **True Negatives(TN) = 2063**  **False Positives(FP) = 14**  **False Negatives(FN) = 10**  **y\_pred = cross\_val\_predict(classifier, X, y, cv=10)**  **#Compute and print confusion matrix**  **cm = confusion\_matrix(y, y\_pred)**  **print('Confusion\_Matrix\n\n', cm)**  **print('\nTrue Positives(TP) = ' , cm[0,0])**  **print('\nTrue Negatives(TN) = ' , cm[1,1])**  **print('\nFalse Positives(FP) = ' , cm[0,1])**  **print('\nFalse Negatives(FN) = ' , cm[1,0])**  **print('-------------------------------------------------------')**  **print(classification\_report(y, y\_pred))**  **Score: [1. 0.99756691 0.99837794 0.99756691 0.99918897 0.99837794 0.99837794 0.99432279 0.99918897 0.99594485]**  **Average Score: 0.9978913219789132**  **print('-------------------------------------------------------')**  **print("Score:", scores)**  **print("Average Score:", scores.mean())**  **print('-------------------------------------------------------')**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Classes** | **precision** | **recall** | **f1-score** | **support** | | **0 (TP,FP)** | **1.00** | **1.00** | **1.00** | **10257** | | **1 (TN,FN)** | **0.99** | **1.00** | **0.99** | **2073** |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **accuracy** |  |  | **1.00** | **12330** | | **macro avg** | **1.00** | **1.00** | **1.00** | **12330** | | **weighted avg** | **1.00** | **1.00** | **1.00** | **12330** |  **Splitting Dataset Into Training Set And Testing Set:** **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)**  **(test\_size=0.2 means that 20% of the data will be used for testing,**  **and the remaining 80% will be used for training the model)** **Hold-Out Method:** **#Standardize features by removing mean and scaling to unit variance:**  **scaler = StandardScaler()**  **scaler.fit(X\_train)**  **X\_train = scaler.transform(X\_train)**  **X\_test = scaler.transform(X\_test)**  **#Use the DT classifier to fit data:**  **classifier = DecisionTreeClassifier()**  **classifier.fit(X\_train, y\_train)**  **Confusion\_Matrix:**  **[[2060 9]**  **[ 1 396]]**  **True Positives(TP) = 2060**  **True Negatives(TN) = 396**  **False Positives(FP) = 9**  **False Negatives(FN) = 1**  **accuracy= 0.9959448499594485**  **#Predict y data with classifier:**  **y\_predict = classifier.predict(X\_test)**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print('confusion\_matrix\n\n', cm)**  **print('\nTrue Positives(TP) = ' , cm[0,0])**  **print('\nTrue Negatives(TN) = ' , cm[1,1])**  **print('\nFalse Positives(FP) = ' , cm[0,1])**  **print('\nFalse Negatives(FN) = ' , cm[1,0])**  **print('-------------------------------------------------------')**  **print(classification\_report(y\_test, y\_pred))**  **print('-------------------------------------------------------')**  **print("accuracy=", accuracy\_score(y\_test, y\_pred))**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Classes** | **precision** | **recall** | **f1-score** | **support** | | **0 (TP,FP)** | **1.00** | **1.00** | **1.00** | **2069** | | **1 (TN,FN)** | **0.98** | **1.00** | **0.99** | **397** |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **accuracy** |  |  | **1.00** | **2466** | | **macro avg** | **0.99** | **1.00** | **0.99** | **2466** | | **weighted avg** | **1.00** | **1.00** | **1.00** | **2466** |  **Comparing 10-Fold Cross Validation And Hold-Out Method:** **By comparing the classification results obtained from :**  **10-Fold Cross-Validation and The Holdout Method, we can observe the following:**  **Accuracy:**  **Both 10-fold cross-validation and the hold-out method achieved high accuracy, with values of 1.00. This means that the model correctly classified all instances in both cases.**  **Precision, Recall, and F1-score:**  **For class 0, the Precision, Recall, and F1-score are consistently high, with values of 1.00 in both evaluation methods. This indicates that the model performed exceptionally well in correctly identifying negative instances.**  **For class 1, the Precision, Recall, and F1-score are also high, with values close to 1.00 in both cases. This indicates that the model performed well in correctly identifying positive instances, although there is a slight decrease in Precision and F1-score in**  **the hold-out method compared to the cross-validation.**  **Overall performance:**  **The macro average and weighted average scores are high for both evaluation methods, indicating strong overall performance of the model.**  **However, the macro average scores are slightly lower in**  **the hold-out method compared to 10-fold cross-validation, mainly due to the decrease in Precision and F1-score for class 1.**  **In summary, both 10-fold cross-validation and the hold-out method resulted in high accuracy and Precision, indicating the model's ability to correctly classify instances.**  **The hold-out method showed a slight decrease in Precision**  **and F1-score for class 1 compared to 10-fold cross-validation, but overall, the model demonstrated excellent performance in both evaluation methods.** PART 3:**Classification:** ***K Nearest Neighbor(KNN):*** **Code:** **import pandas as pd**  **from sklearn.preprocessing import LabelEncoder**  **from sklearn.metrics import classification\_report, confusion\_matrix,** **accuracy\_score**  **dataset = pd.read\_csv("online\_shoppers\_intention\_Dataset 5.csv")**  **print(dataset.info())**  **print(dataset.shape)🡺 (12330, 18).**  **dataset.head()**  **#We will delete an unnecessary attribute like the month column which has a data type object**  **dataset = dataset.drop(['Month'], axis= 1)🡺** (**The axis=1 it indicates that the column should be dropped).**  **print(dataset.shape) (12330, 17).**  **dataset.head()** **Data Preprocessing:****Handling Missing Data:** **#Check for null values in data**  **nullcount = dataset.isnull().sum()**  **print('Total number of null values in dataset:', nullcount.sum()) 🡺 Total number of null values in dataset: 0** **Converting String Value To Integer Type:** **dataset['VisitorType'] = LabelEncoder().fit\_transform(dataset['VisitorType'])**  **dataset['Weekend'] = LabelEncoder().fit\_transform(dataset['Weekend'])**  **dataset['Revenue'] = LabelEncoder().fit\_transform(dataset['Revenue'])**  **dataset.head()** **Explaining:**  * **There is no missing value to handle.** * **We can’t have text in our data if we’re going to run any kind of model on it. So before we can run a model, we need to make this data ready for the model.**   **And to convert this kind of categorical text data into model-understandable numerical data,**  **we used the LabelEncoder class.** **Separating Features And Labels:** **#Assign values to the X and y variables:**  **X = dataset.iloc[:, :-1]🡺** (**X will contain all rows and all columns of the dataset except for the last column).**  **y = dataset.iloc[:, -1] 🡺 (y will contain all rows but only the values from the last column of the dataset).**  **print(X.shape)🡺 (12330, 16).**  **X.head()**  **y.head()** **Data Standardization:** **#Scale the data to be between -1 and 1**  **from sklearn.preprocessing import StandardScaler**  **scaler = StandardScaler()**  **X = scaler.fit\_transform(X)🡺 The fit\_transform() method performs both fitting and transforming in a single step.** **10-Fold Cross Validation:** **from sklearn.neighbors import KNeighborsClassifier**  **from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict**  **#Create K-nearest neighbors classifier**  **classifier = KNeighborsClassifier()**  **#Perform 10-fold cross-validation**  **scores = cross\_val\_score(classifier, X, y, cv=10)**  **#Perform cross-validation predictions**  **Confusion\_Matrix:**  **[[10195 62]**  **[ 70 2003]]**  **True Positives(TP) = 10195**  **True Negatives(TN) = 2003**  **False Positives(FP) = 62**  **False Negatives(FN) = 70**  **y\_pred = cross\_val\_predict(classifier, X, y, cv=10)**  **#Compute and print confusion matrix**  **cm = confusion\_matrix(y, y\_pred)**  **print('Confusion\_Matrix\n\n', cm)**  **print('\nTrue Positives(TP) = ' , cm[0,0])**  **print('\nTrue Negatives(TN) = ' , cm[1,1])**  **print('\nFalse Positives(FP) = ' , cm[0,1])**  **print('\nFalse Negatives(FN) = ' , cm[1,0])**  **print('-------------------------------------------------------')**  **print(classification\_report(y, y\_pred))**  **Score: [0.99107867 0.98540146 0.99432279 0.98621249 0.99107867 0.99107867**  **0.98945661 0.98783455 0.99270073 0.9837794] Average Score: 0.9892944038929439**  **print('-------------------------------------------------------')**  **print("Score:", scores)**  **print("Average Score:", scores.mean())**  **print('-------------------------------------------------------')**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Classes** | **precision** | **recall** | **f1-score** | **support** | | **0 (TP,FP)** | **0.99** | **0.99** | **0.99** | **10257** | | **1 (TN,FN)** | **0.97** | **0.97** | **0.97** | **2073** |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **accuracy** |  |  | **0.99** | **12330** | | **macro avg** | **0.98** | **0.98** | **0.98** | **12330** | | **weighted avg** | **0.99** | **0.99** | **0.99** | **12330** |  **Splitting Dataset Into Training Set And Testing Set:** **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)**  **(test\_size=0.2 means that 20% of the data will be used for testing,**  **and the remaining 80% will be used for training the model)** **Hold-Out Method:** **#Standardize features by removing mean and scaling to unit variance:**  **scaler = StandardScaler()**  **scaler.fit(X\_train)**  **X\_train = scaler.transform(X\_train)**  **X\_test = scaler.transform(X\_test)**  **# Use the KNN classifier to fit data:**  **classifier = KNeighborsClassifier(n\_neighbors=5)**  **classifier.fit(X\_train, y\_train)**  **Confusion\_Matrix:**  **[[2065 19]**  **[ 22 360]]**  **True Positives(TP) = 2065**  **True Negatives(TN) = 360**  **False Positives(FP) = 19**  **False Negatives(FN) = 22**  **accuracy= 0.9833738848337389**  **#Predict y data with classifier:**  **y\_predict = classifier.predict(X\_test)**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print('confusion\_matrix\n\n', cm)**  **print('\nTrue Positives(TP) = ' , cm[0,0])**  **print('\nTrue Negatives(TN) = ' , cm[1,1])**  **print('\nFalse Positives(FP) = ' , cm[0,1])**  **print('\nFalse Negatives(FN) = ' , cm[1,0])**  **print('-------------------------------------------------------')**  **print(classification\_report(y\_test, y\_pred))**  **print('-------------------------------------------------------')**  **print("accuracy=", accuracy\_score(y\_test, y\_pred))**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Classes** | **precision** | **recall** | **f1-score** | **support** | | **0 (TP,FP)** | **0.99** | **0.99** | **0.99** | **2084** | | **1 (TN,FN)** | **0.95** | **0.94** | **0.95** | **382** |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **accuracy** |  |  | **0.98** | **2466** | | **macro avg** | **0.97** | **0.97** | **0.97** | **2466** | | **weighted avg** | **0.98** | **0.98** | **0.98** | **2466** |  **Comparing 10-Fold Cross Validation And Hold-Out Method:** **By comparing the classification results obtained from :**  **10-Fold Cross-Validation and The Holdout Method, we can observe the following:**  **Accuracy:**  **The accuracy achieved with 10-fold cross-validation is 0.99, while the accuracy with the hold-out method is 0.98. This means that the model's performance was slightly better with cross-validation, as it achieved a higher overall accuracy.**  **Precision, Recall, and F1-score:**  **For class 0 (negative class), both methods achieved high Precision, Recall, and F1-score (all above 0.99), indicating excellent performance in correctly classifying negative instances.**  **For class 1 (positive class), the performance is slightly lower.**  **The hold-out method achieved Precision, Recall, and F1-score of 0.95, 0.94, and 0.95, respectively, while the 10-fold cross-validation achieved slightly higher scores of 0.97, 0.97, and 0.97, respectively.**  **This suggests that the model performed better in correctly classifying positive instances with cross-validation.**  **Overall performance:**  **Both methods achieved high performance with similar patterns in Precision, Recall, and F1-score for each class.**  **The macro average and weighted average scores are also close, indicating consistent performance across both methods.**  **In summary, both 10-fold cross-validation and the hold-out method yielded good classification results,**  **with slightly higher performance in terms of accuracy and positive class classification achieved through 10-fold cross-validation.** **NOTES:** **Precision:**  **Precision is the ratio of true positives (TP) to the sum of true positives and false positives (FP).**  **It measures the model's ability to correctly identify positive instances**  **out of the total instances predicted as positive.**  **High Precision indicates a low rate of false positives.**  **Recall:**  **Recall is the ratio of true positives (TP) to the sum of true positives and false negatives (FN).**  **It measures the model's ability to correctly identify positive instances out of the total actual positive instances. High recall indicates a low rate of false negatives.**  **The F1-score is the harmonic mean of Precision and Recall.**  **It provides a balance between precision and recall and is a single metric that combines both measures. The F1-score is useful when you want to consider both precision and Recall simultaneously.**  **Support:**  **Support represents the number of instances of each class in the actual data.**  **It indicates the prevalence of each class in the dataset.**  **Accuracy:**  **Accuracy is the overall accuracy of the model, which is the ratio of correct predictions to the total number of instances.**  **It provides a general measure of how well the model performs on the entire dataset.**  **The classification\_report function displays these metrics for each class individually,**  **as well as an average across all classes.**  **It helps in evaluating the performance of a classification model by providing insights into precision, Recall, F1-score, and Support for each class, along with an overall accuracy score.**  **Macro Average (macro avg):**  **The macro average calculates the unweighted average of the performance metrics (such as Precision, Recall, and F1-score) for each class.**  **It treats each class equally and gives equal importance to the metrics of each class.**  **The macro average provides an overall performance measure that is not influenced by class imbalance.**  **Weighted Average (weighted avg):**  **The weighted average calculates the average of the performance metrics, weighted by the Support (number of instances) of each class.**  **It takes into account the class imbalance by giving more weight to the metrics of the classes with a larger number of instances.**  **The weighted average provides an overall performance measure that considers the contribution of each class based on its prevalence in the dataset.** |  |  |
|  | **Identifying The Best Classifier:** **Based on the provided accuracy scores,**  **the Decision Tree (DT) classifier**  **has the highest accuracy of 0.9959.**  **Decision Trees are known for their interpretability,**  **ability to capture complex relationships,**  **and high accuracy in many scenarios.**  **Interpretability:**  **Decision Trees provide a clear and intuitive representation of the decision-making process. The resulting tree structure can be easily understood and interpreted.**  **Handling Mixed Data Types:**  **Decision Trees can handle both numerical and categorical features without requiring extensive preprocessing or feature engineering. They can handle missing values and categorical variables directly, which simplifies the data preparation process.**  **Robustness to Outliers:**  **Decision Trees are generally robust to outliers and noise in the data. Outliers have minimal impact on the tree structure, as the splits are determined based on information gain or other impurity measures.** |  |  |

## **References:**

## **Part 1:**

[**https://mode.com/blog/python-data-visualization-libraries/**](https://mode.com/blog/python-data-visualization-libraries/)

## **Part 2:**

**Allen, D. M. [1971], “Mean square error of prediction as a criterion for**

**selecting variables,” *Technometrics*, 13, 469–475.**

[**https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2\_score.html**](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html)

## **Part 3:**

**Ristianini, N., Shawe-Taylor, J.: An Introduction to Support VectorMachines**

**and other kernel-based learning methods. Cambridge University Press,**

**Cambrigde Book website:**[**www.support-vector.net**](http://www.support-vector.net/)**(2000).**

**Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. (1984).**

***Classification and regression trees*. Pacific Grove: Wadsworth & Brooks.**

**https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn**