**ASSIGNMENT-8**

**KNN Model:**

1. **Connecting to the Google Drive:** This part connects to Google Drive so that the file can be accessed. It imports the "drive" module from the "google.colab" package and mounts the Google Drive at "/content/drive".
2. **Importing the Libraries:** This section imports necessary libraries like numpy, pandas, and matplotlib. These libraries are widely used for data manipulation and visualization.
3. **Importing the Dataset:** In this section, the dataset is imported using the pandas module. It loads the "Social\_Network\_Ads.csv" file from Google Drive into a Pandas DataFrame. The third and fourth columns of the dataset are extracted as feature variables (x) and target variables (y), respectively.
4. **Splitting the Data into Train and Test Sets:** The train\_test\_split function from the sklearn.model\_selection module is used to split the data into training and testing sets. The test size is set to 0.25, meaning that 25% of the data will be used for testing. The random\_state parameter is set to 0 for reproducibility.
5. **Feature Scaling:** Feature scaling is an essential step in many machine learning algorithms. Here, the StandardScaler function from the sklearn.preprocessing module is used to normalize the data. It scales the features so that they have zero mean and unit variance.
6. **Training the KNN Model on the Training Set:** The KNeighborsClassifier function from the sklearn.neighbors module is used to train the KNN model on the training set. The n\_neighbors parameter is set to 5, which means that the five nearest neighbors will be considered for each new data point. The metric parameter is set to "minkowski", and the p parameter is set to 2, which corresponds to the Euclidean distance metric.
7. **Predicting the Results:** The trained KNN model is used to predict the class of a new data point. Here, a sample data point with age 30 and estimated salary 87000 is passed through the model, and the predicted class is outputted.
8. **Predicting the Results on Test Set:** The predict method of the KNeighborsClassifier object is used to predict the class of each data point in the test set. The predicted values and true values are concatenated into a single array and printed.
9. **Making the Confusion Matrix:** The confusion\_matrix function from the sklearn.metrics module is used to create a confusion matrix. This matrix displays the number of true positives, true negatives, false positives, and false negatives for each class.
10. **Visualizing the Results from the Training Set:** The results of the KNN model on the training set are visualized using matplotlib. The data is first transformed back to the original scale using the inverse\_transform method of the StandardScaler object. Then, a meshgrid is created over the range of the data, and the predicted class is plotted as a contour plot.
11. **Visualizing the Results from the Test Set:** The same procedure as step 10 is repeated for the test set. The meshgrid is created using the range of the test data, and the predicted class is plotted as a contour plot.

**Support Vector Machine:**

This code performs Support Vector Machine (SVM) algorithm on a dataset named "Social\_Network\_Ads.csv" using Scikit-learn library. The SVM algorithm is used to classify whether a person will purchase a product based on his/her age and estimated salary.

1. **Connecting to the google drive:** This segment imports the drive module from google.colab and mounts the Google Drive to access the file.
2. **Importing the libraries:** This segment imports the necessary libraries such as numpy, pandas, and matplotlib which are required to implement SVM.
3. **Importing the dataset:** This segment reads the Social\_Network\_Ads.csv file using pd.read\_csv() function and stores the data in a DataFrame object named data. Then, it extracts the independent variable (X) and dependent variable (y) from the DataFrame using data.iloc and stores them in variables x and y respectively.
4. **Splitting the data into train and test sets:** This segment splits the data into training and testing datasets using train\_test\_split() function. It randomly splits the data into 75% training data and 25% testing data.
5. **Feature Scaling:** This segment scales the feature variables x\_train and x\_test using StandardScaler() from Scikit-learn library to normalize the values of X to have mean = 0 and standard deviation = 1.
6. **Training the SVM model on the training set:** This segment trains the SVM model on the training dataset using SVC() function from Scikit-learn library. It uses the linear kernel function to classify the data points into two classes.
7. **Predicting the Results:** This segment predicts whether a person will purchase a product or not using the trained SVM model on a single data point. It uses the predict() function of the classifier.
8. **Predicting the Results on the test set:** This segment predicts the result on the testing dataset using the trained SVM model and compares the predicted result with the actual result.
9. **Making the confusion matrix:** This segment creates the confusion matrix using confusion\_matrix() function from Scikit-learn library. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives.
10. **Visualizing the result from the training set:** This segment visualizes the results of the training dataset. It creates a meshgrid of the data points and classifies each point using the trained SVM model. Then, it plots the decision boundary and the data points on a graph.
11. **Visualizing the result from the test set:** This segment visualizes the results of the testing dataset. It creates a meshgrid of the data points and classifies each point using the trained SVM model. Then, it plots the decision boundary and the data points on a graph.

**Decision Tree Classification:**

This code is for implementing a decision tree classifier model on a dataset called 'Social\_Network\_Ads.csv' using the scikit-learn library in Python. The code performs the following functions:

1. **Connecting to the Google Drive**: This segment of the code connects to the Google Drive account of the user by mounting it to access a file called 'Social\_Network\_Ads.csv', which is stored in the drive. The drive is mounted using the 'drive.mount()' function from the 'google.colab' module.
2. **Importing the Libraries**: This segment of the code imports the required Python modules and libraries for machine learning, namely 'numpy', 'pandas', and 'matplotlib.pyplot'. These libraries are used to perform mathematical operations, manipulate data, and visualize the results.
3. **Importing the Dataset**: This segment of the code imports the 'Social\_Network\_Ads.csv' dataset using the 'pd.read\_csv()' function from the 'pandas' module. It then extracts the independent variables (x) and dependent variable (y) from the dataset using the 'iloc' method. Finally, the code prints the dataset, x, and y.
4. **Splitting the Data into Train and Test Sets**: This segment of the code splits the data into a training set and a test set using the 'train\_test\_split()' function from the 'sklearn.model\_selection' module. The split is made by taking 75% of the data as the training set and 25% as the test set. The split is done randomly using a seed value of 0.
5. **Feature Scaling**: This segment of the code scales the independent variables in both the training and test sets using the 'StandardScaler()' function from the 'sklearn.preprocessing' module. This step is performed to bring all the independent variables to a common scale, making the model more efficient.
6. **Training the Decision Tree Model on the Training Set**: This segment of the code trains the decision tree model on the training set using the 'DecisionTreeClassifier()' function from the 'sklearn.tree' module. The criterion for splitting the tree is set to 'entropy', and the random state is set to 0.
7. **Predicting the Results**: This segment of the code predicts the results for a new data point, [30, 87000], using the 'predict()' method of the 'DecisionTreeClassifier()' object. It then prints the predicted result.
8. **Predicting the Results on the Test Set**: This segment of the code predicts the results for the test set using the 'predict()' method of the 'DecisionTreeClassifier()' object. It then prints the concatenated arrays of predicted and actual results using 'np.concatenate()' function from the 'numpy' module.
9. **Making the Confusion Matrix**: This segment of the code calculates the confusion matrix and the accuracy score of the model using the 'confusion\_matrix()' and 'accuracy\_score()' functions from the 'sklearn.metrics' module. It then prints the confusion matrix.
10. **Visualizing the Result from the Training Set**: This segment of the code visualizes the decision boundary of the decision tree model on the training set. It does this by creating a meshgrid of points and then applying the 'predict()' method of the 'DecisionTreeClassifier()' object to these points. The result is then plotted using the 'contourf()' function from the 'matplotlib.colors' module.
11. **Visualizing the Result from the Test Set**: This segment of the code visualizes the decision boundary of the decision tree model on the test set. It follows the same procedure as the previous segment but with the test set.

**Random Forest Classification:**

1. **Connecting to the Google Drive:** The code starts by importing the necessary libraries, including pandas, numpy, and matplotlib. Then it connects to Google Drive to access the dataset that is stored in the drive.
2. **Importing the Libraries**: The required libraries for machine learning like numpy, pandas, and matplotlib are imported.
3. **Importing the Dataset**: Using the pandas module, the data from the CSV file is read and the 3rd and 4th columns of the dataset are extracted as the feature variables (x) and the target variable (y), respectively. The dataset is then printed along with the feature variables and target variables.
4. **Splitting the Data into Train and Test Sets**: The dataset is split into training and testing sets using the train\_test\_split() function from the sklearn.model\_selection module. The split is done with a test size of 25% and a random state of 0. The x and y training and testing sets are then printed.
5. **Feature Scaling**: Feature scaling is done using the StandardScaler() function from the sklearn.preprocessing module. The x\_train and x\_test sets are transformed using the fit\_transform() and transform() methods of the StandardScaler object, respectively.
6. **Training the Random Forest Model on the Training Set**: The RandomForestClassifier() function from the sklearn.ensemble module is used to create the random forest model. The hyperparameters are set such that there are 10 decision trees in the forest, the criterion is set to entropy, and the random state is set to 0. The fit() method is used to train the model on the training set.
7. **Predicting the Results**: The predict() method of the trained classifier is used to predict whether a person with age 30 and estimated salary 87000 will purchase the product or not.
8. **Predicting the Results on Test Set**: The predict() method of the trained classifier is used to predict the target variable on the test set. The predicted and actual values are printed for comparison.
9. **Making the Confusion Matrix**: The confusion\_matrix() function and accuracy\_score() function from the sklearn.metrics module are used to calculate the confusion matrix and accuracy score for the test set.
10. **Visualizing the Result from the Training Set**: The decision boundary of the trained classifier on the training set is visualized using the matplotlib module. The inverse\_transform() method of the StandardScaler object is used to convert the scaled feature variables to their original values. A contour plot is created with a red-green color scheme representing the two classes. The scatter plot shows the training set with red dots representing the customers who did not buy the product, and green dots representing the customers who did buy the product.
11. **Visualizing the Result from the Test Set**: The decision boundary of the trained classifier on the test set is visualized in the same way as the training set, but this time using the data from the test set.