MANISA CELAL BAYAR UNIVERSITY

FACULTY OF ENGINEERING
DEPARTMENT OF CIVIL ENGINEERING

INTERNSHIP PROGRAM

FINAL REPORT

PREPARED BY

150303055 Burak Mustafa Karakaya

SUPERVISOR

Arş. Gör. Dr. Aybike ÖZYÜKSEL ÇİFTÇİOĞLU

1. Table of Contents

2. Figures

3. Tables

4. Support Vector Machine

Support vector machine, typically used for classification problems, is a supervised learning method. It draws a line on a plane to separate the points located on it. Its goal is to ensure that the line is at maximum distance from the points of both classes. It is suitable for small to medium-sized datasets that are complex in nature.

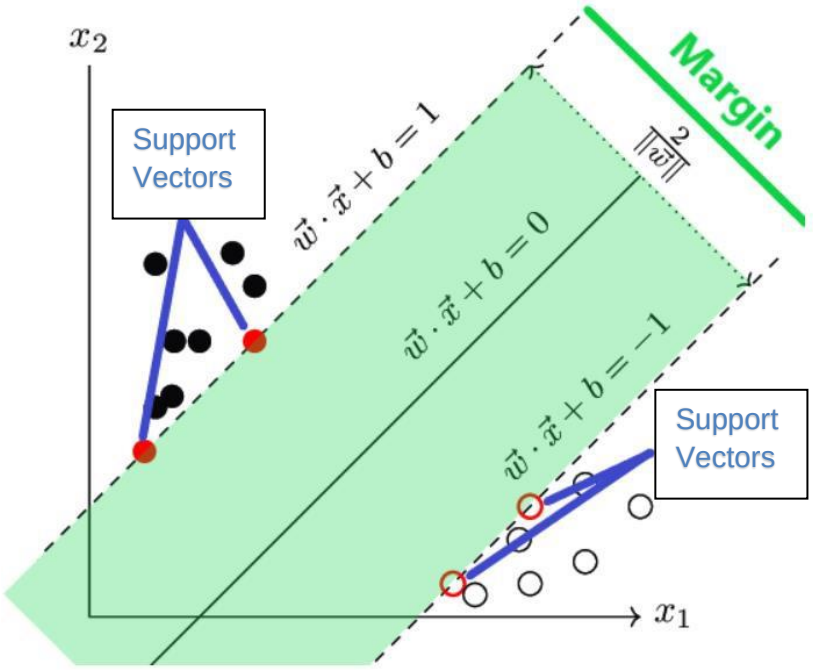

Figure 1 An Example of SVM

The table contains two different classes, black and white. The main goal in classification problems is to determine in which class the future data will be located. To achieve this classification, a line that separates the two classes is drawn and the green area between -1 and +1 on this line is called Margin. The wider the Margin, the better the separation between two or more classes. Support Vector Machine (SVM) is a supervised learning method that is suitable for complex but small to medium-sized datasets.

Application of SVM

10 girişten 1-10 aralığındakiler Filtre

Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Figure 2 SVM Data

Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Figure 3 SVM Library

Importing the dataset

```
[2]  dataset = pd.read_csv('Position_Salaries.csv')
     X = dataset.iloc[:, 1:-1].values
     y = dataset.iloc[:, -1].values
```

```
[3]  print(X)
```

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
```

```
[4]  print(y)
```

```
[  45000   50000   60000   80000  110000  150000  200000  300000  500000
 1000000]
```

```
[5]  y = y.reshape(len(y),1)
```

Figure 4 SVM Data Insertion

Feature Scaling

```
[7]  from sklearn.preprocessing import StandardScaler
     sc_X = StandardScaler()
     sc_y = StandardScaler()
     X = sc_X.fit_transform(X)
     y = sc_y.fit_transform(y)
```

```
[8]  print(X)
```

```
[[-1.5666989 ]
 [-1.21854359]
 [-0.87038828]
 [-0.52223297]
 [-0.17407766]
 [ 0.17407766]
 [ 0.52223297]
 [ 0.87038828]
 [ 1.21854359]
 [ 1.5666989 ]]
```

```
[9]  print(y)
```

```
[[-0.72004253]
 [-0.70243757]
 [-0.66722767]
 [-0.59680786]
 [-0.49117815]
```

Figure 5 SVM Data Scaling

Training the SVR model on the whole dataset

```
[10] from sklearn.svm import SVR
     regressor = SVR(kernel = 'rbf')
     regressor.fit(X, y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was e
  y = column_or_1d(y, warn=True)
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Figure 6 SVM Training

Predicting a new result

```
[11] sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])))
```

```
array([170370.0204065])
```

Figure 7 SVM Prediction

▾ Visualising the SVR results

```
[12] plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
     plt.plot(sc_X.inverse_transform(X), sc_y.inverse_transform(regressor.predict(X)), color = 'blue')
     plt.title('Truth or Bluff (SVR)')
     plt.xlabel('Position level')
     plt.ylabel('Salary')
     plt.show()
```

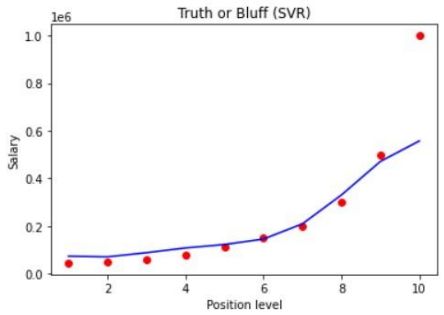

Figure 8 SVM Visual Results

▾ Visualising the SVR results (for higher resolution and smoother curve)

```
[13] X_grid = np.arange(min(sc_X.inverse_transform(X)), max(sc_X.inverse_transform(X)), 0.1)
     X_grid = X_grid.reshape((len(X_grid), 1))
     plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
     plt.plot(X_grid, sc_y.inverse_transform(regressor.predict(sc_X.transform(X_grid))), color = 'blue')
     plt.title('Truth or Bluff (SVR)')
     plt.xlabel('Position level')
     plt.ylabel('Salary')
     plt.show()
```

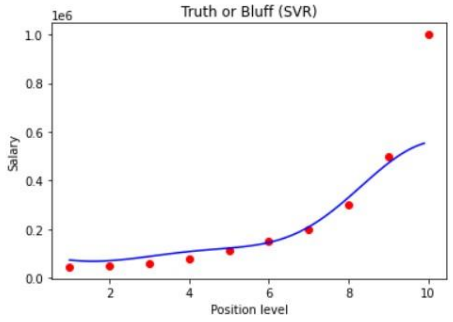

Figure 9 SVM Visual Results

5. Regression Analysis

Regression analysis is a statistical method used to measure the relationship between two or more quantitative variables. If the analysis is conducted using only one variable, it is called simple regression, and if multiple variables are used, it is called multiple regression analysis. Regression analysis can provide insights into the presence and strength of a relationship between variables, if such a relationship exists.

Regression analysis is a statistical method used to measure the relationship between two or more quantitative variables. If the analysis is done using a single variable, it is called simple regression; if multiple variables are used, it is called multiple regression analysis. Regression analysis can provide information about the presence and strength of the relationship between variables. For instance, an agricultural engineer may want to know the relationship between wheat yield and fertilizer amount, an engineer may want to know the relationship between pressure and temperature, an economist may want to know the relationship between income and consumption expenditures, or an educator may want to know the relationship between the number of days students were absent and their

achievement scores. Regression not only shows the functional form of the linear relationship between two (or more) variables, with one being dependent and the other being independent, but also enables estimation of one variable when the value of the other variable is known. Generally, all of these variables must be measured on a quantitative scale.

In regression analysis, one of the variables should be dependent while the others should be independent variables. The logic here is that the variable on the left side of the equation is influenced by the variables on the right side. The variables on the right side, however, are not affected by other variables. Here, not being affected means that when we put these variables into a linear equation, they do not have an impact. Multicollinearity and problems of sequential dependence are not referred to here.

6. Decision Tree Regression

A decision tree is a model that reaches a prediction by asking a series of questions to the data, narrowing down the possible values at each question until the model is confident enough to make a single prediction. The order and content of the questions are determined by the model. Also, all questions asked are in a True/False format.

The decision to make strategic divisions greatly affects the accuracy of a tree. The decision criteria are different for classification and regression trees. Regression decision trees typically use mean squared error (MSE) to decide to split a node into two or more child nodes.

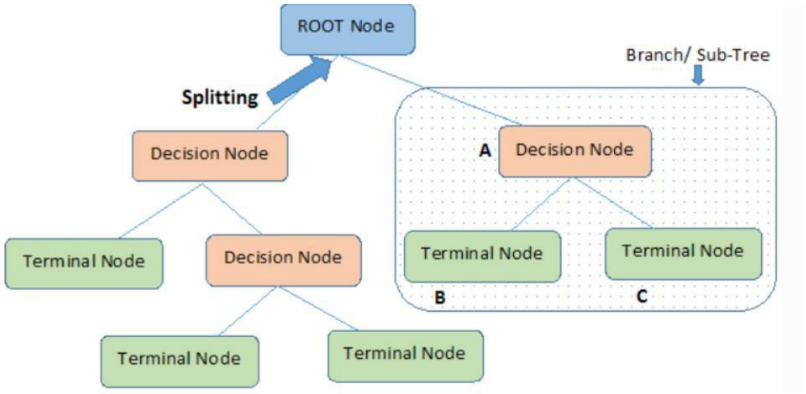

Figure 10 Decision Tree Diagram

- **Root Node:** Represents the entire population or sample, and is divided into two or more homogeneous clusters.
- **Splitting:** The process of dividing a node into two or more child nodes.
- **Decision Node:** A decision node is created when a parent node is split into two or more child nodes.
- **Leaf/Terminal Node:** A leaf or terminal node is a node that is not split further.
- **Pruning:** Removing the child nodes of a decision node is called pruning. It is the opposite of splitting.
- **Branch/Sub-Tree:** A sub-section of the entire tree is called a branch or sub-tree.
- **Parent and Child Node:** A node that has been split into child nodes is referred to as the parent node, while the child nodes are referred to as the child nodes of the parent node.

Application of Decision Tree

Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Figure 11 Decision Tree DataSet

▾ Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

▾ Importing the dataset

```
[2]  dataset = pd.read_csv('Position_Salaries.csv')
     X = dataset.iloc[:, 1:-1].values
     y = dataset.iloc[:, -1].values
```

Figure 12 Decision Tree Library and DataSet

▾ Training the Decision Tree Regression model on the whole dataset

```
[3]  from sklearn.tree import DecisionTreeRegressor
     regressor = DecisionTreeRegressor(random_state = 0)
     regressor.fit(X, y)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
```

▾ Predicting a new result

```
[4]  regressor.predict([[6.5]])
```

```
array([150000.])
```

Figure 13 Decision Tree Training and Prediction

▾ Visualising the Decision Tree Regression results (higher resolution)

```
[5]  X_grid = np.arange(min(X), max(X), 0.01)
     X_grid = X_grid.reshape((len(X_grid), 1))
     plt.scatter(X, y, color = 'red')
     plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
     plt.title('Truth or Bluff (Decision Tree Regression)')
     plt.xlabel('Position level')
     plt.ylabel('Salary')
     plt.show()
```

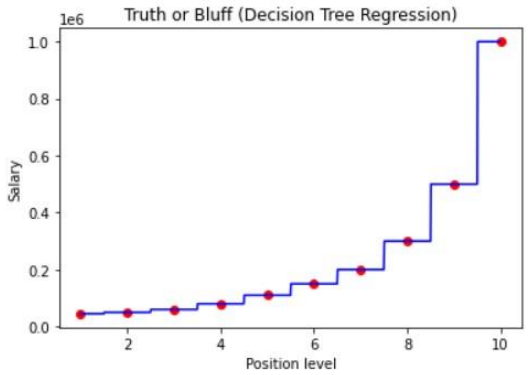

Figure 14 Decision Tree Visual Results

7. Multiple Linear Regression

Regression analysis is conducted to reveal the relationship between a single dependent variable and a set of independent variables associated with it.

Multiple linear regression examines the linear relationship between two or more independent variables and a dependent variable. In multiple regression, there is a correlation between the dependent and independent variables. Let's denote the independent variables as X and the dependent variable as Y.

$$Y=XB+E$$

- Y dependent variable observation vector
- X independent variables observation matrix
- B coefficients vector
- E random error vector

To apply multiple regression to the data, there should be no multicollinearity among the independent variables.

Figure 15 MLR Example

Application of Multiple Linear Regression

AT	V	AP	RH	PE
14.96	41.76	1024.07	73.17	463.26
25.18	62.96	1020.04	59.08	444.37
5.11	39.4	1012.16	92.14	488.56
20.86	57.32	1010.24	76.64	446.48
10.82	37.5	1009.23	96.62	473.9
26.27	59.44	1012.23	58.77	443.67
15.89	43.96	1014.02	75.24	467.35
9.48	44.71	1019.12	66.43	478.42
14.64	45	1021.78	41.25	475.98
11.74	43.56	1015.14	70.72	477.5

Figure 16 MLR DataSet

▾ Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

▾ Importing the dataset

```
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

▾ Splitting the dataset into the Training set and Test set

```
[3]  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_stat
```

Figure 17 MLR Library , Data Set and Data Set Splitting

▾ Training the Multiple Linear Regression model on the Training set

```
[4] from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

▾ Predicting the Test set results

```
[5] y_pred = regressor.predict(X_test)
    np.set_printoptions(precision=2)
    print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

```
[[431.43 431.23]
 [458.56 460.01]
 [462.75 461.14]
 ...
 [469.52 473.26]
 [442.42 438.  ]
 [461.88 463.28]]
```

Figure 18 Decision Tree Training and Prediction

▾ Evaluating the Model Performance

```
[6] from sklearn.metrics import r2_score
    r2_score(y_test, y_pred)
```

```
0.9325315554761302
```

Figure 19 Decision Tree Evaluation

8. Logistic Regression Classification

Logistic Regression is a regression method used for classification. It is used for classifying categorical or numerical data. It works when the dependent variable, or the outcome, can take only two different values (e.g., Yes/No, Male/Female, Obese/Thin).

The aim of logistic regression is to find the most appropriate (yet biologically plausible) model to describe the relationship between a set of independent (predictor or explanatory) variables and a two-way characteristic (dependent variable = response or outcome variable).

Tips for Logistic Regression

- Events are independent
- It does not assume a linear relationship between the dependent variable and the independent variables, but assumes a linear relationship between the logits of the explanatory variables and the response
- The independent variables can be the power terms of the original independent variables or some other nonlinear transformations.
- The dependent variable does not have to follow a normal distribution, but typically assumes a distribution from an exponential family (e.g., binomial, Poisson, multinomial, normal, …); binary logistic regression assumes a binomial distribution of the response.

- Variance homogeneity is not required to be satisfied.
- The errors must be independent, but need not be normally distributed.
- It uses maximum likelihood estimation (MLE) instead of ordinary least squares (OLS) to estimate the parameters, and thus relies on large sample approximations.
- Goodness-of-fit measures rely on sufficiently large samples where the intuitive rule is that less than 5 cells of the cell counts should be less than 20%.

Logistic Regression enables us to examine issues such as the probability of having lung cancer (yes or no) and how it changes based on weight and the number of cigarette packs smoked per day, or the impact of body weight, calorie intake, fat intake, and participant age on the likelihood of a heart attack (yes or no).

Application of Logistic Regression Classification

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Figure 20 LRC Data

Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Importing the dataset

```
[2]  dataset = pd.read_csv('Social_Network_Ads.csv')
     X = dataset.iloc[:, :-1].values
     y = dataset.iloc[:, -1].values
```

Figure 21 LRC Library and DataSet

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_sta
```

```
[4]  print(X_train)
```

```
[    21  72000]
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
```

Figure 22 LRC Splitting the dataset

Feature Scaling

```
[8] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[9] print(X_train)
```

```
[-1.6960924   0.07006676]
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
```

Figure 23 LRC Configuring the DataSet

Training the Logistic Regression model on the Training set

```
[11] from sklearn.linear_model import LogisticRegression
     classifier = LogisticRegression(random_state = 0)
     classifier.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Figure 24 LRC Training and Prediction

Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

```
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[0 0]
[1 1]
[0 1]
[0 0]
```

Figure 25 LRC Training Predictions

Making the Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 8 24]]
0.89
```

Figure 26 LRC Evaluation

Visualising the Training set results

```
[15] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_train), y_train
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Logistic Regression (Training set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
```

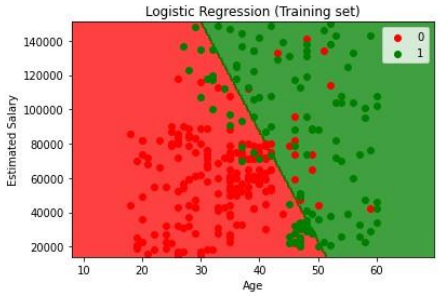

Figure 27 LRC Visualizing Training Results

▾ Visualising the Test set results

```
[16] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_test), y_test
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Logistic Regression (Test set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
```

Figure 28 LRC Visualizing Test Results

9. K-Nearest Neighborhood Classification

In statistics, k-nearest neighbors algorithm (k-NN) is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951 and later extended by Thomas Cover. It is used for both classification and regression.

The k-nearest neighbors (k-NN) algorithm is a non-parametric classification method that was first developed by Evelyn Fix and Joseph Hodges in 1951 and later expanded by Thomas Cover. It is used for both classification and regression. k-NN is the simplest and most commonly used classification algorithm. It is a lazy learning algorithm, meaning that unlike eager learning, there is no training phase. It does not learn from the training data, instead it "memorizes" the training dataset. When making a prediction, it searches for the nearest neighbors in the entire dataset.

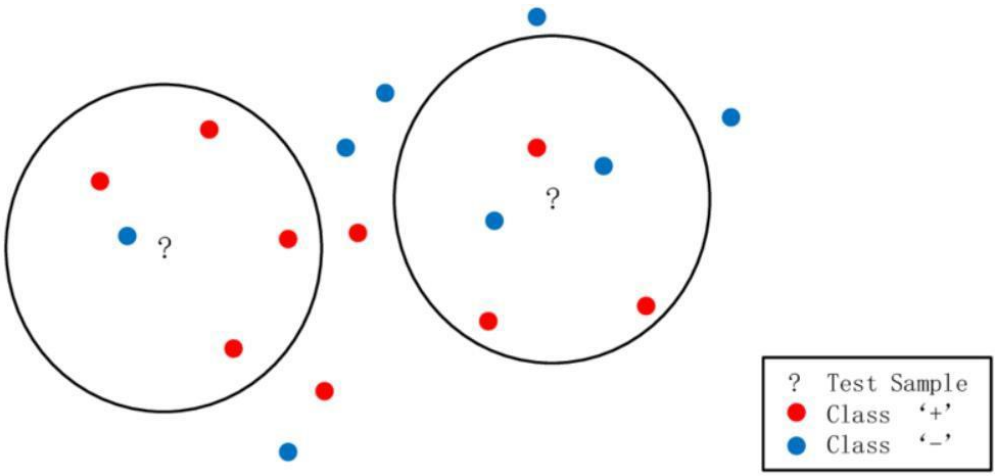

Figure 29 K-NN Classification Example

Application of K-Neares Neighborhood Classification

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Figure 30 K-NN Classification DataSet

Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Importing the dataset

```
[2]  dataset = pd.read_csv('Social_Network_Ads.csv')
     X = dataset.iloc[:, :-1].values
     y = dataset.iloc[:, -1].values
```

Figure 31 K-NN Classification Library ve DataSet

Splitting the dataset into the Training set and Test set

```
[3]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_sta
```

```
[4]  print(X_train)
```

```
[    21  72000]
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
[    29  43000]
[    33  43000]
[    35  38000]
[    41  45000]
```

Figure 32 K-NN Classification Dataset Splitting

Feature Scaling

```
[8] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[9] print(X_train)
```

```
[-1.6960924   0.07006676]
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
```

Figure 33 Scaling the K-NN Classification DataSet

Training the K-NN model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Figure 34 K-NN Classification Training and Prediction

Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [1 1]
```

Figure 35 K-NN Classification Training Predictions

Making the Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
0.93
```

Figure 36 K-NN Classification Evaluation

Visualising the Training set results

```
[15] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_train), y_train
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('K-NN (Training set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
```

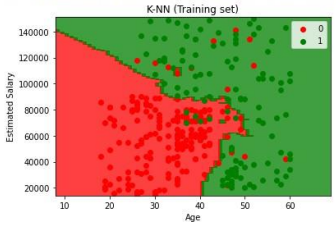

Figure 37 K-NN Classification Visualizing Training Results

Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
```

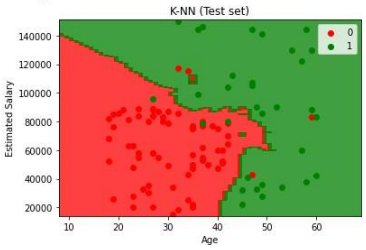

Figure 38 K-NN Classification Visualizing Test Results

10. Support Vector Machine Classification

A support vector machine (SVM) uses algorithms to train and classify data into polarity degrees, taking it beyond X/Y prediction.

For a simple visual explanation, we will use two labels with two data features, red and blue: X and Y. Then we will train our classifier to assign a red or blue label based on an X/Y coordinate.

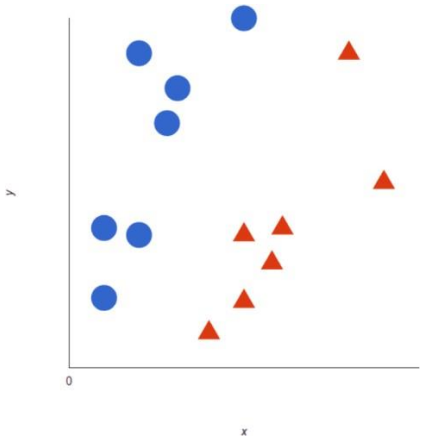

Figure 39 SVM Classification Example

SVM then assigns a hyperplane that separates the labels best. In two dimensions, this is just a line. Everything on one side of the line is red, everything on the other side is blue. In sensitivity analysis, for example, this would be positive and negative.

To maximize machine learning, the best hyperplane is the one with the largest distance between each label:

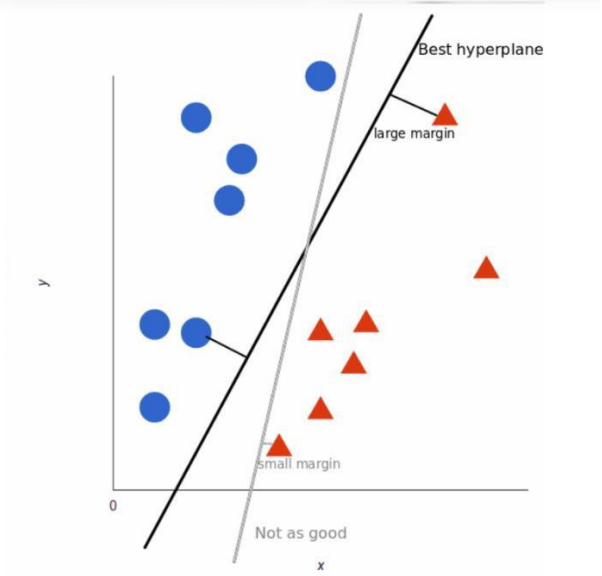

Figure 40 SVM Classification Example 2

However, as datasets become more complex, it may not be possible to draw a single line to separate the data into two classes:

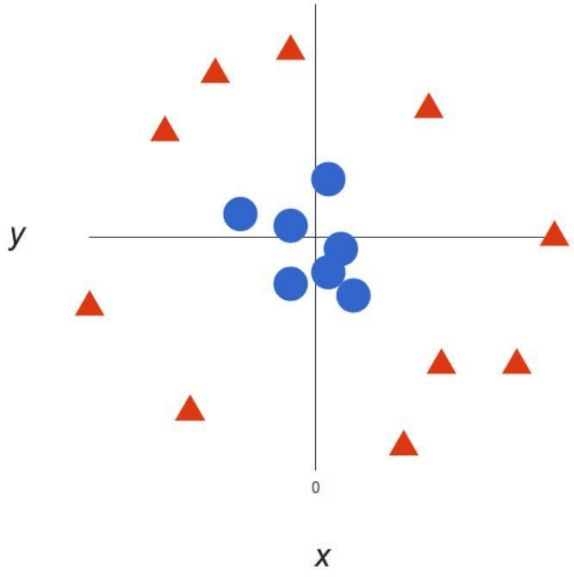

Figure 41 SVM Classification Example 3

By using SVM, the more complex the data is, the more accurate the predictor will be. Imagine the above with an added Z-axis as three-dimensional, so that there is a sphere.

In two dimensions, mapped back with the best hyperplane, it would appear as follows:

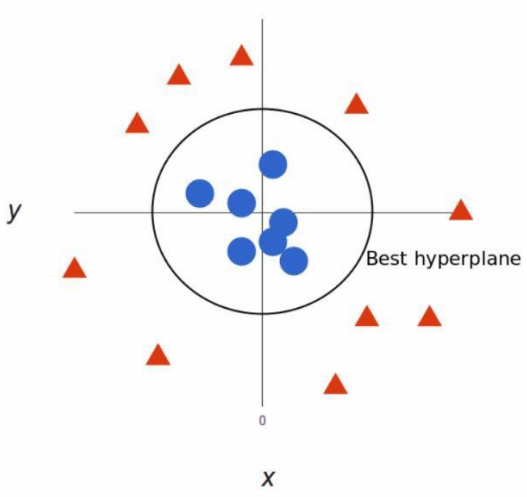

Figure 42 SVM Classification Example 4

Since SVM is multi-dimensional, it provides more accurate machine learning.

Application of SVM Classification

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Figure 43 SVM Classification DataSet

▾ Importing the libraries

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

▾ Importing the dataset

```
[2] dataset = pd.read_csv('Social_Network_Ads.csv')
    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values
```

▾ Splitting the dataset into the Training set and Test set

```
[3] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
[4] print(X_train)
```

```
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
[    29  43000]
```

Figure 44 SVM Classification Library , DataSet and Splitting

▾ Feature Scaling

```
[8] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[9] print(X_train)
```

```
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
[-1.6960924  -1.5535493 ]
[-1.20093113 -1.089659  ]
[-0.70576986 -0.1038921 ]
[ 0.08648817  0.09905991]
[ 0.28455268  0.27301877]
[ 0.8787462  -0.5677824 ]
[ 0.28455268 -1.14764529]
[-0.11157634  0.67892279]
[ 2.1661655  -0.68375498]
[-1.29996338 -1.37959044]
[-1.00286662 -0.94469328]
[-0.01254409 -0.42281668]
[-0.21060859 -0.45180983]
```

Figure 45 Scaling the K-NN Classification

▾ Training the SVM model on the Training set

```
[11] from sklearn.svm import SVC
     classifier = SVC(kernel = 'linear', random_state = 0)
     classifier.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```

▾ Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

▾ Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
```

Figure 46 SVM Classification Training and Predictions

Making the Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[66  2]
 [ 8 24]]
0.9
```

Figure 47 SVM Classification Evaluation

Visualising the Training set results

```
[15] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_train), y_train
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('SVM (Training set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

Figure 48 SVM Classification Visualizing Training Results

Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

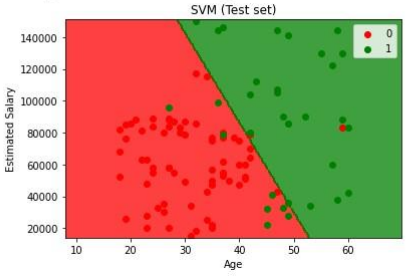

Figure 49 SVM Classification Visualizing Test Results

11. Kernel Trick/Method

In machine learning, kernel machines are a class of algorithms used for model analysis, with the best-known member being support vector machines (SVM). The general task of model analysis is to discover and examine general types of relationships in datasets, such as sets, rankings, principal components, correlations, and classifications. For many algorithms that solve these tasks, data in the raw representation must be explicitly transformed into feature vector representations using a user-specified feature map. In contrast, kernel methods only require a user-specified kernel, which is a similarity function over pairs of data points in the raw representation, instead of the number of data points.

Popular Kernel Methods

- Fisher kernel
- Graph kernels
- Kernel smoother
- Polynomial kernel
- Radial basis function kernel (RBF)
- String kernels
- Neural tangent kernel
- Neural network Gaussian process (NNGP) kernel

Application of Kernel SVM

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Figure 50 Kernel SVM DataSet

Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Importing the dataset

```
[2]  dataset = pd.read_csv('Social_Network_Ads.csv')
     X = dataset.iloc[:, :-1].values
     y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
[3]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_sta
```

```
[4]  print(X_train)
```

```
[    21  72000]
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
```

Figure 51 Kernel SVM Library, Dataset and Splitting

Feature Scaling

```
[8] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[9] print(X_train)
```

```
[-1.6960924   0.07006676]
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
[-1.6960924  -1.5535493 ]
[-1.20093113 -1.089659  ]
[-0.70576986 -0.1038921 ]
[ 0.08648817  0.09905991]
[ 0.28455268  0.27301877]
```

Figure 52 Scaling for Kernel SVM

Training the Kernel SVM model on the Training set

```
[11] from sklearn.svm import SVC
     classifier = SVC(kernel = 'rbf', random_state = 0)
     classifier.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```

Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

```
[0 0]
[0 0]
[0 0]
```

Figure 53 Kernel SVM Training and Predictions

Making the Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]
0.93
```

Figure 54 Kernel SVM Evaluation

Visualising the Training set results

```
[15] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_train), y_train
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Kernel SVM (Training set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches w
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches w
```

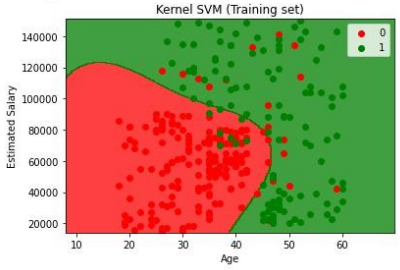

Figure 55 Kernel SVM Visualizing Training Results

▾ Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case i
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case i
```

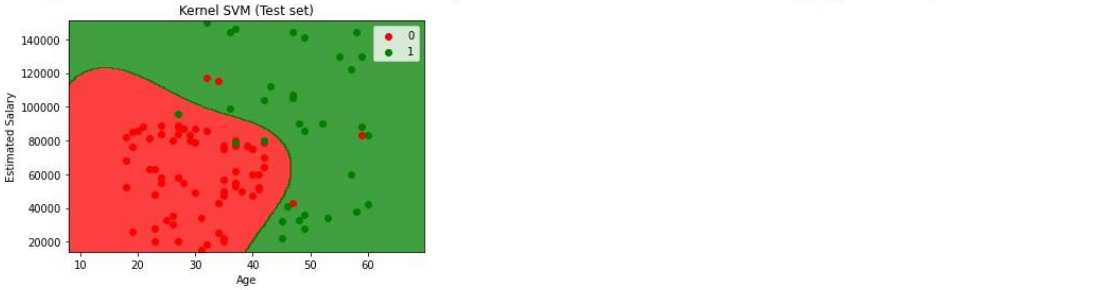

Figure 56 Kernel SVM Visualizing Test Results

12. Navie Bayes

Naive Bayes classification algorithm is a classification/categorization algorithm named after the mathematician Thomas Bayes. Naive Bayes classification aims to determine the class or category of the data presented to the system using a set of calculations defined by probability principles.

In Naive Bayes classification, a certain amount of trained data is presented to the system (e.g. 100 samples). The training data must have a certain class/category. Using probability calculations on the training data, the system operates on new test data presented to it based on previously obtained probability values to determine the category in which the test data belongs. Of course, the more trained data there is, the more accurate the system is in determining the true category of the test data.

Naive Bayes classification can be used in many areas, but what is important here is not what is being classified, but how it is classified. The data to be trained can be binary or text data, and what matters here is how we establish a proportional relationship between these data rather than what they are.

Application of Navie Bayes

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Her sayfada 10 satır göster

1 2 10 30 40

Figure 57 Navie Bayes Dataset

Importing the libraries

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

Importing the dataset

```
[2] dataset = pd.read_csv('Social_Network_Ads.csv')
    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
[3] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
[4] print(X_train)
```

```
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
```

Figure 58 Navie Bayes Library, Data Set and Splitting

▾ Feature Scaling

```
[8] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[9] print(X_train)
```

```
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
[-1.6960924  -1.5535493 ]
[-1.20093113 -1.089659  ]
[-0.70576986 -0.1038921 ]
[ 0.08648817  0.09905991]
[ 0.28455268  0.27301877]
[ 0.8787462  -0.5677824 ]
[ 0.28455268 -1.14764529]
[-0.11157634  0.67892279]
[ 2.1661655  -0.68375498]
[-1.29996338 -1.37959044]
[-1.00286662 -0.94469328]
```

Figure 59 Scaling for Navie Bayes

▾ Training the Naive Bayes model on the Training set

+ Kod + Metin

```
[11] from sklearn.naive_bayes import GaussianNB
     classifier = GaussianNB()
     classifier.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

▾ Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

▾ Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
```

Figure 60 Navie Bayes Training and Predictions

Making the Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 7 25]]
0.9
```

Figure 61 Navie Bayes Evaluation

Visualising the Training set results

```
[15] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_train), y_train
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Naive Bayes (Training set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

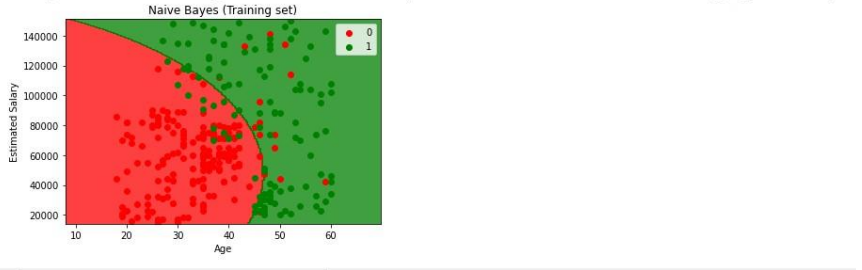

Figure 62 Navie Bayes Visualizing Training Results

▾ Visualising the Test set results

```
[16] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_test), y_test
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Naive Bayes (Test set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pr
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pr
```

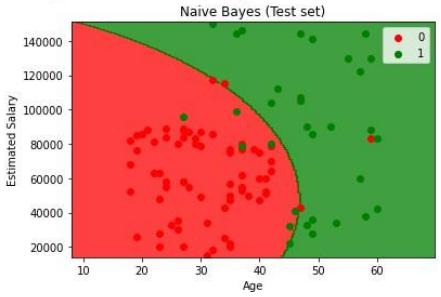

Figure 63 Navie Bayes Visualizing Test Results

13. Decision Tree Classification

Predictors				Target
Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

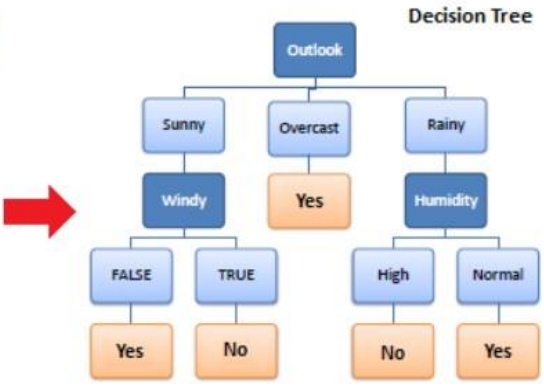

Figure 64 Decision Tree Classification Example

Application of Decision Tree Classification

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Her sayfada 10 satır göster

1 2 10 30 40

Figure 65 DT Classification DataSet

▾ Importing the libraries

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

▾ Importing the dataset

```
[2] dataset = pd.read_csv('Social_Network_Ads.csv')
    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values
```

▾ Splitting the dataset into the Training set and Test set

```
[3] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
[4] print(X_train)
```

```
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
[    29  43000]
```

Figure 66 DT Classification Library, DataSet and Splitting

▾ Feature Scaling

```
[8] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
[9] print(X_train)
```

```
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
[-1.6960924  -1.5535493 ]
[-1.20093113 -1.089659  ]
[-0.70576986 -0.1038921 ]
[ 0.08648817  0.09905991]
[ 0.28455268  0.27301877]
[ 0.8787462  -0.5677824 ]
[ 0.28455268 -1.14764529]
[-0.11157634  0.67892279]
[ 2.1661655  -0.68375498]
[-1.29996338 -1.37959044]
[-1.00286662 -0.94469328]
[-0.01254409 -0.42281668]
[-0.21060859 -0.45180983]
```

Figure 67 Scaling for DT Classification

▾ Training the Decision Tree Classification model on the Training set

```
[11] from sklearn.tree import DecisionTreeClassifier
     classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
     classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

▾ Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

▾ Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
```

Figure 68 DT Classification Training and Predictions

▾ Making the Confusion Matrix

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[62  6]
 [ 3 29]]
0.91
```

▾ Visualising the Training set results

Figure 69 DT Classification Evaluation

▾ Visualising the Training set results

```
[15] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_train), y_train
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Decision Tree Classification (Training set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

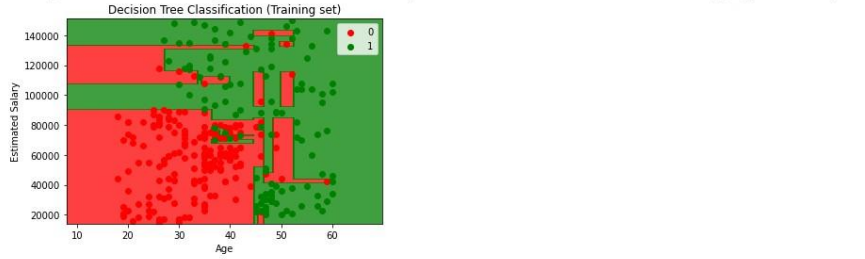

Figure 70 DT Classification Visualizing Training Results

▾ Visualising the Test set results

```
[16] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_test), y_test
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Decision Tree Classification (Test set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

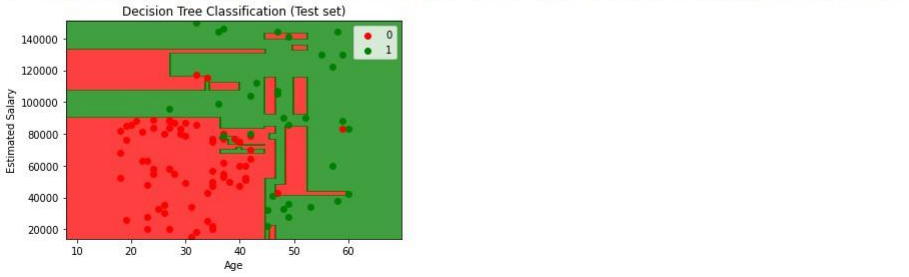

Figure 71 DT Classification Visualizing Test Results

14. Random Forest Classification

The Random Forests algorithm is a community learning method that aims to improve the classification value by generating multiple decision trees during the classification process. The decision trees individually created come together to form a decision forest. The decision trees here are randomly selected subsets from the data set to which they belong. It provides excellent validity. For many data sets, Adaboost and SVM provide more accurate results. It produces results using data sets that have thousands of variables, many class labels, missing data, or an imbalanced distribution. As trees are added to the community, it begins to give low-bias results for error prediction for the test set. It purifies noisy data.

Application areas:

- Astronomy
- Biomedical
- Control systems
- Financial analysis
- Health
- Molecular biology
- Physics
- Software development

Application of Random Forest Classification

Age	EstimatedSalary	Purchased
19	19000	0
35	20000	0
26	43000	0
27	57000	0
19	76000	0
27	58000	0
27	84000	0
32	150000	1
25	33000	0
35	65000	0

Her sayfada 10 satır göster

1 2 10 30 40

Figure 72 RF Classification DataSet

Importing the libraries

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

Importing the dataset

```
[2] dataset = pd.read_csv('Social_Network_Ads.csv')
    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
[3] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
[4] print(X_train)
```

```
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
[    29  43000]
```

Figure 73 RF Classification Library, DataSet and Splitting

Feature Scaling

```
[8]  from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

```
[9]  print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845  1.8676417 ]
 [-0.01254409  1.25878567]
 [-0.90383437  2.27354572]
 [-1.20093113 -1.58254245]
```

Figure 74 Scaling for RF Classification

Training the Random Forest Classification model on the Training set

```
[11] from sklearn.ensemble import RandomForestClassifier
     classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
     classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Predicting a new result

```
[12] print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
[13] y_pred = classifier.predict(X_test)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
```

Figure 75 RF Classification Training and Predictions

Making the Confusion Matrix

+ Kod + Metin

```
[14] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[63  5]
 [ 4 28]]
0.91
```

Figure 76 RF Classification Evaluation

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

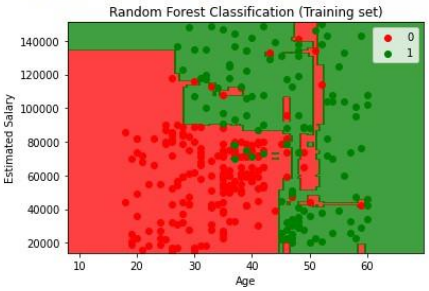

Figure 77 RF Classification Visualizing Training Results

Visualising the Test set results

```
[16] from matplotlib.colors import ListedColormap
     X_set, y_set = sc.inverse_transform(X_test), y_test
     X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                          np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
     plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
                  alpha = 0.75, cmap = ListedColormap(('red', 'green')))
     plt.xlim(X1.min(), X1.max())
     plt.ylim(X2.min(), X2.max())
     for i, j in enumerate(np.unique(y_set)):
         plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
     plt.title('Random Forest Classification (Test set)')
     plt.xlabel('Age')
     plt.ylabel('Estimated Salary')
     plt.legend()
     plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have prec
```

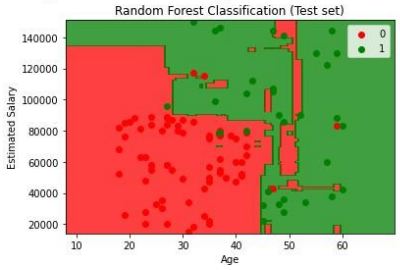

Figure 78 RF Classification Visualizing Test Results

15. K-Means Clustering

The K-means clustering method is a process of dividing a data set consisting of N data objects into K clusters, which are specified as input parameters. The goal is to maximize the similarity within each cluster and minimize the similarity between clusters.

K-means is one of the most commonly used clustering algorithms. It is easy to implement and can quickly and efficiently cluster large-scale data. The parameter "K" represents the required fixed number of clusters before starting the algorithm. The K-means algorithm with its iterative partitioning structure minimizes the sum of distances between each data object and its corresponding cluster. The K-means algorithm tries to determine the K number of clusters that will minimize the sum of squared

errors. A clustering can be considered accurate if the intra-cluster similarity is large and the inter-cluster similarity is small. Although the problem is NP-hard, the K-means algorithm generally provides a good solution with an iterative approach.

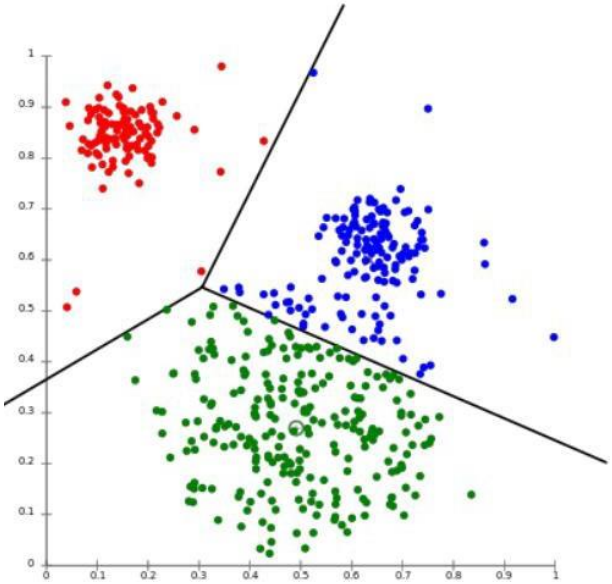

Figure 79 K-Means Ckustering Example

Application of K-Means Clustering

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0001	Male	19	15	39
0002	Male	21	15	81
0003	Female	20	16	6
0004	Female	23	16	77
0005	Female	31	17	40
0006	Female	22	17	76
0007	Female	35	18	6
0008	Female	23	18	94
0009	Male	64	19	3
0010	Female	30	19	72

Her sayfada 10 satır göster

1 2 10 20

Figure 80 K-Means Ckustering DataSet

Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Importing the dataset

```
[2]  dataset = pd.read_csv('Mall_Customers.csv')
     X = dataset.iloc[:, [3, 4]].values
```

Figure 81 K-Means Ckustering Library and Dataset

Using the elbow method to find the optimal number of clusters

```
[3]  from sklearn.cluster import KMeans
     wcss = []
     for i in range(1, 11):
         kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
         kmeans.fit(X)
         wcss.append(kmeans.inertia_)
     plt.plot(range(1, 11), wcss)
     plt.title('The Elbow Method')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
     plt.show()
```

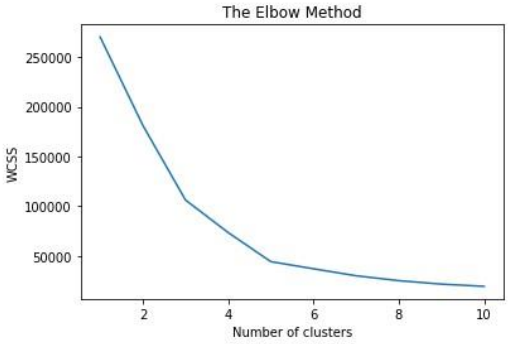

Figure 82 Optimum Clustering with K-Means Clustering

Training the K-Means model on the dataset

```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Figure 83 K-Means Ckustering Training

▾ Visualising the clusters

```
[5]  plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
     plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
     plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
     plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
     plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
     plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
     plt.title('Clusters of customers')
     plt.xlabel('Annual Income (k$)')
     plt.ylabel('Spending Score (1-100)')
     plt.legend()
     plt.show()
```

Figure 84 K-Means Ckustering Visualizing

16. Hierarchial Clustering

Hierarchical algorithms are divided into two categories: AGNES (Agglomerative Nesting), also known as Agglomerative Clustering, and DIANA (Divise Analysis), also known as Divisive Hierarchical Clustering. AGNES uses a bottom-up clustering approach. Each data point is initially considered as a cluster, and the most similar pairs are clustered together. This process continues until there are no more data points to cluster. The resulting tree is then shown in a dendrogram. The dendrogram in the figure illustrates the clustering results of AGNES and DIANA schematically.

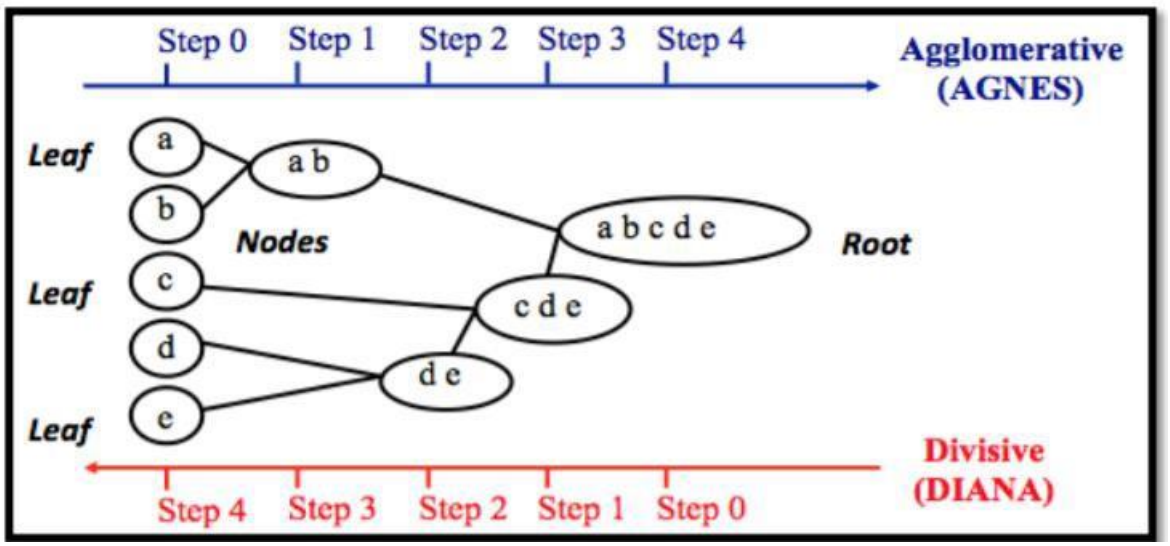

Figure 85 Hierarchial Clustering Example

Hierarchical algorithms are divided into two main categories: AGNES (Agglomerative Nesting), which is Agglomerative Clustering, and DIANA (Divise Analysis), which is Divisive Hierarchical Clustering. AGNES follows a bottom-up clustering logic. At the initial stage, each data point is considered as a cluster, and the most similar pairs are clustered. This process continues until there is no more data to cluster. The resulting tree is shown in a dendrogram. The dendrogram for AGNES and DIANA

clustering is presented schematically in the figure.

In contrast, DIANA follows the opposite logic to AGNES, dividing clusters from top to bottom until only one data point remains in each cluster. In hierarchical clustering, similarities and proximities between clusters can be determined by different methods. These include complete linkage clustering, single linkage clustering, average linkage clustering, and Ward's minimum variance method.

When using these methods, all data points are initially considered as similar and treated as if they are all in one cluster. Then, the two most dissimilar data points or subclusters are selected, and a relationship is established by measuring the distance between them and the other data points. In complete linkage clustering, at the first and second stages examined in the dendrogram, the algorithm calculates the differences between all pairs and uses these differences as the distance between two clusters. The single linkage clustering algorithm calculates the differences between pairs of clusters in the first and second stages and selects the smallest difference as the linkage criterion. The average linkage clustering algorithm calculates the average difference between pairs of clusters in the first and second stages and uses it as the distance between two clusters. In Ward's method, the within-cluster variance is minimized, and at each step, the two clusters with the smallest distance are merged.

Application of Hierarchial Clustering

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0001	Male	19	15	39
0002	Male	21	15	81
0003	Female	20	16	6
0004	Female	23	16	77
0005	Female	31	17	40
0006	Female	22	17	76
0007	Female	35	18	6
0008	Female	23	18	94
0009	Male	64	19	3
0010	Female	30	19	72

Her sayfada 10 satır göster

1 2 10 20

Figure 86 Hierarchial Clustering DataSet

▾ Importing the libraries

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

▾ Importing the dataset

```
[2]  dataset = pd.read_csv('Mall_Customers.csv')
     X = dataset.iloc[:, [3, 4]].values
```

Figure 87 Hierarchial Clustering Library and Dataset

Using the dendrogram to find the optimal number of clusters

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```

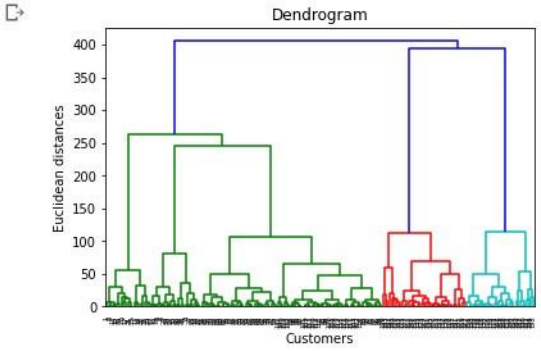

Figure 88 Hierarchical Clustering Optimum Clustering

Training the Hierarchical Clustering model on the dataset

```
[4] from sklearn.cluster import AgglomerativeClustering
    hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
    y_hc = hc.fit_predict(X)
```

Figure 89 Hierarchial Clustering Training

Visualising the clusters

```
[5]  plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
     plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
     plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
     plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
     plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
     plt.title('Clusters of customers')
     plt.xlabel('Annual Income (k$)')
     plt.ylabel('Spending Score (1-100)')
     plt.legend()
     plt.show()
```

Figure 90 Hierarchial Clustering Visualizing

17. Conclusion

As a result of our studies, we have learned about different methods for regression, classification, and clustering, the fields in which these methods are used, and their differences and the methods themselves, along with their applications.

18. References

https://medium.com/deep-learning-turkiye/nedir-bu-destek-vekt%C3%B6r-makineleri-makine-%C3%B6%C4%9Frenmesi-serisi-2-94e576e4223e

https://en.wikipedia.org/wiki/Support-vector_machine

https://tr.wikipedia.org/wiki/Regresyon_analizi

https://veribilimcisi.com/2017/07/18/coklu-dogrusal-regresyon-nedir-multivariate-linear-regression/

https://cdn.scribbr.com/wp-content/uploads//2020/02/multiple-regression-in-r-graph-1.png

https://medium.com/@ekrem.hatipoglu/machine-learning-classification-k-nn-k-en-yak%C4%B1n-kom%C5%9Fu-part-9-6f18cd6185d

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

https://veribilimcisi.com/2017/07/20/k-en-yakin-komsu-k-nearest-neighborsknn/

https://gdcoder.com/decision-tree-regressor-explained-in-depth/

https://medium.com/@ekrem.hatipoglu/machine-learning-classification-logistic-regression-part-8-b77d2a61aae1

https://veribilimcisi.com/2017/07/18/lojistik-regresyon/

https://monkeylearn.com/blog/classification-algorithms/

https://en.wikipedia.org/wiki/Kernel_method#:~:text=In%20machine%20learning%2C%20kernel%20machines,%2Dvector%20machine%20(SVM).&text=This%20approach%20is%20called%20the,images%2C%20as%20well%20as%20vectors.

https://makineogrenimi.wordpress.com/2017/06/08/destek-vektor-makineleri-support-vector-machines-svms-8/

https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78

https://tr.wikipedia.org/wiki/Naive_Bayes_s%C4%B1n%C4%B1fland%C4%B1r%C4%B1c%C4%B1s%C4%B1

https://kodedu.com/2014/05/naive-bayes-siniflandirma-algoritmasi/

https://www.analyticsvidhya.com/blog/2020/10/a-simple-explanation-of-k-means-clustering/

https://towardsdatascience.com/k-means-clustering-explained-4528df86a120

https://erdincuzun.com/makine_ogrenmesi/hiyerarsik-kumeleme-hierarchical-clustering-odev-benzerlikleri-uzerinden-kopya-gruplarini-bulma/

https://dergipark.org.tr/tr/download/article-file/751785

https://www.veribilimiokulu.com/hiyerarsik-kumeleme/