| MACHINE LEARNING LAB | | | |
|---|---|---|---|
| **Course Code** | **20IS607** | **CIE Marks** | **50** |
| **Teaching Hours/Week (L:T:P:S)** | **0:0:2:0** | **SEE Marks** | **50** |
| **Total Hours** | **2 hrs/week** | **Credits** | **1** |

**Course Learning Objectives:**

This course will enable students to:

1. Familiarize with machine learning tools and libraries to analyze the datasets.
2. Use data preprocessing techniques in machine learning.
3. Implement a model using supervised and unsupervised learning algorithms
4. Identify patterns, trends, and outliers in datasets using visualization libraries in Python.
5. Develop a system to perform various computational tasks faster than traditional systems.

**Lab Experiments:**

1. Demonstrate importing a dataset, identifying, and handling missing values, encoding categorical data, and feature scaling using machine learning libraries.
2. Implement the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a CSV file.
3. Demonstrate the working of the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
4. Construct a decision tree based on the ID3 algorithm. Use an appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.
5. Demonstrate the application of Linear regression to predict the stock market prices of any organization.
6. Demonstrate the use of the Support Vector Machine algorithm for a regression problem on any preferred dataset and evaluate the performance of the model.
7. Write a program to implement the *k*-Nearest Neighbor classification algorithm on the iris flower dataset and visualize the results.
8. Demonstrate the use of K-Means clustering algorithm on any preferred dataset. Use the elbow method to find the optimal number of clusters and visualize the clusters.
9. Apply Hierarchical clustering on the customer segmentation dataset and visualize the clusters and plot the dendrograms.
10. Perform Random Forest classification on the Pima Indians diabetes dataset.
11. Write a program to implement the naïve Bayesian classifier for a sample training dataset. Compute the accuracy of the classifier and visualize the results.

12. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate datasets.

**Course Outcomes:**

Upon completion of the course, the students will be able to

| **Sl. No.** | **Course Outcome (CO)** | **Bloom's Taxonomy Level (BTL)** |
|---|---|---|
| C607.1 | Make use of machine learning tools, and libraries to explore the dataset in building a model. | **L2** |
| C607.2 | Perform data preprocessing techniques to clean and transform the dataset. | **L3** |
| C607.3 | Experiment with various supervised and unsupervised learning algorithms on appropriate datasets. | **L3** |
| C607.4 | Analyze the patterns in data using data visualization techniques. | **L4** |
| C607.5 | Build a neural network model to solve computational tasks and provide better outcomes. | **L3** |

1. **Demonstrate importing a dataset, identifying, and handling missing values, encoding categorical data, and feature scaling using machine learning libraries.**

```
# Data Preprocessing Tools
# Importing the libraries
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
print(X)
print(y)
# Taking care of missing data
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan,
strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
print(X)
# Encoding categorical data
# Encoding the Independent Variable
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder',
OneHotEncoder(), [0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
# Encoding the Dependent Variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
print(y)
# 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 = 0.2, random_state = 1)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])
print(X_train)
print(X_test)
```

2. **Implement the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a CSV file.**

```python
import pandas as pd
import numpy as np

data = pd.read_csv('lab2.csv')
concepts = np.array(data)[:,:-1]
target = np.array(data)[:,-1]

def train(con,tar):
    for i,val in enumerate(tar):
        if val=='yes':
            specific_h = con[i].copy()
            break

    for i,val in enumerate(con):
        if tar[i]=='yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else:
                    pass
    return specific_h

print(train(concepts,target))
```

**3. Demonstrate the working of the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

```
import numpy as np
import pandas as pd

data = pd.read_csv("enjoy_sport.csv")
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)


def initialize(concepts):
    print("\nInitialization of specific_h and genearal_h")
    specific_h = ['0']*len(concepts[0])
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i 
in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    return specific_h, general_h


def learn(concepts, target):
    specific_h,general_h=initialize(concepts)


    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "Yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x] and i==0:
                    specific_h = concepts[0].copy()
                elif h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "No":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", 
specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", 
general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == 
['?', '?', '?', '?', '?', '?']]
```

```
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    indices = [i for i, val in enumerate(general_h) if val ==
['?']*len(concepts[0])]
    #for i in indices:
     #   general_h.remove(['?']*len(concepts[0]))
    #return specific_h, general_h
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

4. **Construct a decision tree based on the ID3 algorithm. Use an appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.**

```
# Decision Tree Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
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
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)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
# Training the Decision Tree Classification model on the
Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy',
random_state = 0)
classifier.fit(X_train, y_train)
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

5. **Demonstrate the application of Linear regression to predict the stock market prices of any organization.**

```
# Simple Linear Regression
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
# 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 = 1/3, random_state = 0)
# Training the Simple Linear Regression model on the Training
set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predicting the Test set results
y_pred = regressor.predict(X_test)
# Visualising the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
# Visualising the Test set results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

# Multiple Linear Regression
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
print(X)
# Encoding categorical data
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder',
OneHotEncoder(), [3])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
print(X)
# 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 = 0.2, random_state = 0)
# Training the Multiple Linear Regression model on the
Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predicting the Test set results
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))
```

**# Polynomial Regression**

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
# Training the Linear Regression model on the whole dataset
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
# Training the Polynomial Regression model on the whole
dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
# Visualising the Linear Regression results
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
# Visualising the Polynomial Regression results
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)),
color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
# Visualising the Polynomial Regression results (for higher
resolution and smoother curve)
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid,
lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color =
'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
# Predicting a new result with Linear Regression
lin_reg.predict([[6.5]])
# Predicting a new result with Polynomial Regression
lin_reg_2.predict(poly_reg.fit_transform([[6.5]]))
```

**6. Demonstrate the use of the Support Vector Machine algorithm for a regression problem on any preferred dataset and evaluate the performance of the model.**

```
# Support Vector Regression (SVR)
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
print(X)
print(y)
y = y.reshape(len(y),1)
print(y)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
print(X)
print(y)
# Training the SVR model on the whole dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
# Predicting a new result
sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]
])))
# Visualising the SVR results
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()
# Visualising the SVR results (for higher resolution and
smoother curve)
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()
```

**7. Write a program to implement the *k*-Nearest Neighbor classification algorithm on the iris flower dataset and visualize the results.**

```
# K-Nearest Neighbors (K-NN)
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
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
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)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
# 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)
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
# 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 = 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()
# 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()
```

**8. Demonstrate the use of K-Means clustering algorithm on any preferred dataset. Use the elbow method to find the optimal number of clusters and visualize the clusters.**

```
# K-Means Clustering
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
# Using the elbow method to find the optimal number of
clusters
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()
# 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)
# Visualising the clusters
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()
```

**9. Apply Hierarchical clustering on the customer segmentation dataset and visualize the clusters and plot the dendrograms.**

```
# Hierarchical Clustering
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
# 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()
# Training the Hierarchical Clustering model on the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity =
'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
# Visualising the clusters
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()
```

**10. Perform Random Forest classification on the Pima Indians diabetes dataset.**

```
# Random Forest Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
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
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)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
# Training the Random Forest Classification model on the
Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10,
criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
# 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()
```

**11. Write a program to implement the naïve Bayesian classifier for a sample training dataset. Compute the accuracy of the classifier and visualize the results.**

```
# Naive Bayes
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
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
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)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
# 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('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
# 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('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

## 12. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate datasets.

```
# Artificial Neural Network
# Importing the libraries
import numpy as np
import pandas as pd
import tensorflow as tf
tf.__version__
# Part 1 - Data Preprocessing
# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
print(X)
print(y)
# Encoding categorical data
# Label Encoding the "Gender" column
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
print(X)
# One Hot Encoding the "Geography" column
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder',
OneHotEncoder(), [1])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
# 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 = 0.2, random_state = 0)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Part 2 - Building the ANN
# Initializing the ANN
ann = tf.keras.models.Sequential()
# Adding the input layer and the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
# Adding the second hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
# Adding the output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# Part 3 - Training the ANN
# Compiling the ANN
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])
# Training the ANN on the Training set
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
# Part 4 - Making the predictions and evaluating the model
# Predicting the result of a single observation
```

```
"""
Homework:
Use our ANN model to predict if the customer with the
following informations will leave the bank:
Geography: France
Credit Score: 600
Gender: Male
Age: 40 years old
Tenure: 3 years
Balance: $ 60000
Number of Products: 2
Does this customer have a credit card? Yes
Is this customer an Active Member: Yes
Estimated Salary: $ 50000
So, should we say goodbye to that customer?
Solution:
"""
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3,
60000, 2, 1, 1, 50000]])) > 0.5)
"""
Therefore, our ANN model predicts that this customer stays in
the bank!
Important note 1: Notice that the values of the features were
all input in a double pair of square brackets. That's because
the "predict" method always expects a 2D array as the format
of its inputs. And putting our values into a double pair of
square brackets makes the input exactly a 2D array.
Important note 2: Notice also that the "France" country was
not input as a string in the last column but as "1, 0, 0" in
the first three columns. That's because of course the predict
method expects the one-hot-encoded values of the state, and as
we see in the first row of the matrix of features X, "France"
was encoded as "1, 0, 0". And be careful to include these
values in the first three columns, because the dummy variables
are always created in the first columns.
"""
# Predicting the Test set results
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```