import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

Importing the Dataset

data = pd.read\_csv('drive/My Drive/super/Artificial Neural Networks/Churn\_Modelling.csv')

data.head(4)

data.info()

data.describe()

data.tail()

# Checking if our dataset contains any NULL values

data.isnull().sum()

Data Analysis

data['Gender'].value\_counts()

# Plotting the features of the dataset to see the correlation between them

plt.hist(x = data.Gender, bins = 3, color = 'pink')

plt.title('comparison of male and female')

plt.xlabel('Gender')

plt.ylabel('population')

plt.show()

data['Age'].value\_counts()

# comparison of age in the dataset

plt.hist(x = data.Age, bins = 10, color = 'orange')

plt.title('comparison of Age')

plt.xlabel('Age')

plt.ylabel('population')

plt.show()

data['Geography'].value\_counts()

# comparison of geography

plt.hist(x = data.Geography, bins = 5, color = 'green')

plt.title('comparison of Geography')

plt.xlabel('Geography')

plt.ylabel('population')

plt.show()

data['HasCrCard'].value\_counts()

# comparision of how many customers hold the credit card

plt.hist(x = data.HasCrCard, bins = 3, color = 'red')

plt.title('how many people have or not have the credit card')

plt.xlabel('customers holding credit card')

plt.ylabel('population')

plt.show()

data['IsActiveMember'].value\_counts()

# How many active member does the bank have ?

plt.hist(x = data.IsActiveMember, bins = 3, color = 'brown')

plt.title('Active Members')

plt.xlabel('Customers')

plt.ylabel('population')

plt.show()

# comparison between Geography and Gender

Gender = pd.crosstab(data['Gender'],data['Geography'])

Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(6, 6))

# comparison between geography and card holders

HasCrCard = pd.crosstab(data['HasCrCard'], data['Geography'])

HasCrCard.div(HasCrCard.sum(1).astype(float), axis = 0).plot(kind = 'bar',

stacked = True,figsize = (6, 6))

# comparison of active member in differnt geographies

IsActiveMember = pd.crosstab(data['IsActiveMember'], data['Geography'])

IsActiveMember.div(IsActiveMember.sum(1).astype(float), axis = 0).plot(kind = 'bar',

stacked = True, figsize= (6, 6))

# comparing ages in different geographies

Age = pd.crosstab(data['Age'], data['Geography'])

Age.div(Age.sum(1).astype(float), axis = 0).plot(kind = 'bar',

stacked = True, figsize = (15,15))

# calculating total balance in france, germany and spain

total\_france = data.Balance[data.Geography == 'France'].sum()

total\_germany = data.Balance[data.Geography == 'Germany'].sum()

total\_spain = data.Balance[data.Geography == 'Spain'].sum()

print("Total Balance in France :",total\_france)

print("Total Balance in Germany :",total\_germany)

print("Total Balance in Spain :",total\_spain)

# plotting a pie chart

labels = 'France', 'Germany', 'Spain'

colors = ['cyan', 'magenta', 'orange']

sizes = [311, 300, 153]

explode = [ 0.01, 0.01, 0.01]

plt.pie(sizes, colors = colors, labels = labels, explode = explode, shadow = True)

plt.axis('equal')

plt.show()

Data Preprocessing

# Removing the unnecassary features from the dataset

data = data.drop(['CustomerId', 'Surname', 'RowNumber'], axis = 1)

print(data.columns)

data.shape

# splitting the dataset into x(independent variables) and y(dependent variables)

x = data.iloc[:,0:10]

y = data.iloc[:,10]

print(x.shape)

print(y.shape)

print(x.columns)

#print(y)

# Encoding Categorical variables into numerical variables

# One Hot Encoding

x = pd.get\_dummies(x)

x.head()

x.shape

# splitting the data into training and testing set

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

# Feature Scaling

# Only on Independent Variable to convert them into values ranging from -1 to +1

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.fit\_transform(x\_test)

x\_train = pd.DataFrame(x\_train)

x\_train.head()

Modelling

Decision Tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion\_matrix

model = DecisionTreeClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuaracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

Random Forest

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

# k fold cross validatio

from sklearn.model\_selection import cross\_val\_score

cvs = cross\_val\_score(estimator = model, X = x\_train, y = y\_train, cv = 10)

print(cvs)

print("Mean Accuracy :", cvs.mean())

print("Variance :", cvs.std())

Logistic Regression

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

Support Vector Machine

from sklearn.svm import SVC

model = SVC()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

# k fold cross validatio

from sklearn.model\_selection import cross\_val\_score

cvs = cross\_val\_score(estimator = model, X = x\_train, y = y\_train, cv = 10)

print(cvs)

print("Mean Accuracy :", cvs.mean())

print("Variance :", cvs.std())

Multi Layer Perceptron

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier(hidden\_layer\_sizes = (100, 100), activation ='relu',

solver = 'adam', max\_iter = 50)

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

Aritificial Neural Networks

import keras

from keras.models import Sequential

from keras.layers import Dense

# creating the model

model = Sequential()

# first hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu', input\_dim = 14))

# second hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# third hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# fourth hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# fifth hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# output layer

model.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN

# binary\_crossentropy loss function used when a binary output is expected

model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

model.fit(x\_train, y\_train, batch\_size = 10, nb\_epoch = 50)

# creating the model

model = Sequential()

from keras.layers import Dropout

# first hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu', input\_dim = 14))

model.add(Dropout(0.5))

# second hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

model.add(Dropout(0.5))

# output layer

model.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN

# binary\_crossentropy loss function used when a binary output is expected

model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

model.fit(x\_train, y\_train, batch\_size = 10, nb\_epoch = 50)

# creating the model

model = Sequential()

# first hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu', input\_dim = 14))

model.add(Dropout(0.1))

# second hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

model.add(Dropout(0.1))

# output layer

model.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN

# binary\_crossentropy loss function used when a binary output is expected

model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

model.fit(x\_train, y\_train, batch\_size = 10, nb\_epoch = 50)

# creating the model

model = Sequential()

# first hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu', input\_dim = 13))

# second hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# output layer

model.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN

# binary\_crossentropy loss function used when a binary output is expected

model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

model.fit(x\_train, y\_train, batch\_size = 10, nb\_epoch = 49)

data.columns

'''

predicting if the costumer having following information will leave the bank or not ?

Geography : france

Age = 50

Credit score = 850

Tenure = 4

Balance = 150000

Number of Products = 5

Gender = Female

Has Credit Card = yes

Is Active Member = yes

Estimated Salary = 85000

'''

new\_prediction = model.predict(sc.transform(np.array([[850, 50, 4, 150000, 5, 1, 1, 85000, 1, 0, 0, 1, 0]])))

new\_prediction = (new\_prediction > 0.5 )

print(new\_prediction)

from keras.wrappers.scikit\_learn import KerasClassifier

from sklearn.model\_selection import cross\_val\_score

from keras.layers import Dense

from keras.models import Sequential

def build\_classifier():

# creating the model

model = Sequential()

# first hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu', input\_dim = 13))

# second hidden layer

model.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# output layer

model.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN

# binary\_crossentropy loss function used when a binary output is expected

model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

return model

model = KerasClassifier(build\_fn = build\_classifier, batch\_size = 10, nb\_epoch = 50)

accuracies = cross\_val\_score(estimator = model, X = x\_train, y = y\_train, cv = 10, n\_jobs = -1)

print("Accuracies :", accuracies)

print("Mean Accuracy :", accuracies.mean())

print("Variance :", accuracies.std())