```
# -*- coding: utf-8 -*-
"""appl_ml_project2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SkH4FNF85oc3T6F3h67y50DcVnBcbGgP
"""

import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
import numpy as np
import matplotlib.pyplot as plt
from itertools import combinations
from tabulate import tabulate
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from matplotlib import pyplot as plt

"""# MLP classifier"""

def read_dataset(path):
  df=pd.read_excel(path)
  lab_encoder = preprocessing.LabelEncoder()
  df['Date'] = lab_encoder.fit_transform(df['Date'])
  X=df.copy()
  Y=X['Profit']
  X.drop(['Profit','Unnamed: 0','targetPrice','Unnamed: 0.1'],axis=1,inplace=True)

  scalerX = MinMaxScaler()
  scalerX.fit(X)
  X = scalerX.transform(X)
  return X,Y
def mlp(hl,lr,lri,activ,X,Y):
  trainX, testX, trainY, testY =train_test_split(X, Y, test_size=0.3,
random_state=24061)
  # Solve the problem using an artificial neural network
  regpenalty = 0.001
  clf = MLPClassifier(hidden_layer_sizes=hl, activation=activ, solver="sgd",
                      alpha=regpenalty, early_stopping=True, learning_rate=lr,
learning_rate_init=lri, validation_fraction=0.42)
  clf.fit(trainX,trainY)
  annPredY = clf.predict(testX)
  tn, fp, fn, tp =confusion_matrix(testY,annPredY).ravel()
  miscl=(fp+fn)/(tn+fp+fn+tp)
  accuracy=accuracy_score(testY, annPredY)
  auroc= metrics.roc_auc_score(testY, annPredY)
  return miscl,auroc,accuracy



"""# dataset 1

columns :['Date', 'Volume', 'unemp_rate', 'T5YIE', 'Close',
```

```
       'Open', 'High', 'Low', 'GDP', 'prev_close']

"""

path="/content/finalDataset.xlsx"
lrs=["adaptive"]
lris=[0.001]
activ=["identity", "logistic", "tanh", "relu"]
tolerance=1e-4
hls=[(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10),
    (1,1,1),(2,2,2),(3,3,3),(4,4,4),(5,5,5),(6,6,6),(7,7,7),(8,8,8),(9,9,9),
(10,10,10)]
results_2=[]
X,Y=read_dataset(path)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)

# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_2.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_2.append([hl,lr,lri,tolerance,activation,acc,miscl,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

results_2.sort(key=lambda x: (x[2],x[3]),reverse=True)
headers3=["hidden_layer_size","actiavtion","accuracy","AUROC"]

print(tabulate(results_2[:10], headers=headers3))

print(tabulate(results_2[:100], headers=headers3))

"""# dataset 1 polynomial features

degree= 2
"""

poly_results_2=[]
X,Y=read_dataset(path)
trans = PolynomialFeatures(degree=2)
data = trans.fit_transform(X)

# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
```

```
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_2.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_2.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

print(tabulate(poly_results_2, headers=headers3))

poly_results_2.sort(key=lambda x: (x[2],x[3]),reverse=True)



"""# dataset 2
columns: ['Date', 'Volume', 'unemp_rate', 'T5YIE', 'Close',
       'Open', 'High', 'Low', 'GDP']

"""

path="/content/finalDataset (1).xlsx"

results_1=[]
X,Y=read_dataset(path)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
```

```
#results_acc_1=sorted(results_acc_1, key = lambda x: int(x[5]))
results_1.sort(key=lambda x: (x[2],x[3]),reverse=True)
#sort_AUROC=sorted(results_auroc, key = lambda x: int(x[5]))

print(tabulate(results_1[:100], headers=headers3))

"""# dataset 2 polynomial features
degree = 2
"""

#path="/content/finalDataset.xlsx"
poly_results_1=[]
X,Y=read_dataset(path)
trans = PolynomialFeatures(degree=2)
data = trans.fit_transform(X)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)


poly_results_1.sort(key=lambda x: (x[2],x[3]),reverse=True)


print(tabulate(poly_results_1[:10],headers=headers3))

print(tabulate(poly_results_1[:100],headers=headers3))

"""# dataset 3

columns: ['Date', 'Volume', 'unemp_rate', 'T5YIE', 'Close',
       'Open', 'High', 'Low', 'GDP', 'prev_close', 'prev_open',
'prev_high','prev_low']

"""

path="/content/finalDataset 2.xlsx"
results_3=[]
X,Y=read_dataset(path)
for i in range(1,5):
```

```
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

print(tabulate(results_3, headers=headers3))

results_3.sort(key=lambda x: (x[2],x[3]),reverse=True)

print(tabulate(results_3[:10],headers=headers3))

print(tabulate(results_3[:100],headers=headers3))

"""# dataset 3 polynomial features

degree = 2
"""

poly_results_3=[]
X,Y=read_dataset(path)
trans = PolynomialFeatures(degree=2)
data = trans.fit_transform(X)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc,=mlp(hl,lr,lri,activation,X,Y)
        poly_results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_3.append([hl,activation,acc,auroc])
```

```
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

print(tabulate(poly_results_3, headers=headers3))

poly_results_3.sort(key=lambda x: (x[2],x[3]),reverse=True)

print(tabulate(poly_results_3[:10], headers=headers3))

print(tabulate(poly_results_3[:100], headers=headers3))

"""#results"""

# result to excel
r=pd.DataFrame.from_records(results_1)
overall=pd.DataFrame.from_records(results_1,columns=['hidden_layer','activation','d
s_1_accu,','ds_1_auroc'])
r=pd.DataFrame.from_records(poly_results_1)
overall["poly_1_accu"]=list(r[2])
overall ["poly_1_auroc"]=list(r[3])
r=pd.DataFrame.from_records(results_2)
overall["ds_2_accu"]=list(r[2])
overall["ds_2_auroc"]=list(r[3])
r=pd.DataFrame.from_records(poly_results_2)
overall["poly_2_accu"]=list(r[2])
overall["poly_2_auroc"]=list(r[3])
r=pd.DataFrame.from_records(results_3)
overall["ds_3_accu"]=list(r[2])
overall["ds_3_auroc"]=list(r[3])
r=pd.DataFrame.from_records(poly_results_3)
overall["poly_3_accu"]=list(r[2])
overall["poly_3_auroc"]=list(r[3])

overall

overall.to_excel('/content/overall.xlsx')

"""#MLP regressor"""

# target price as target and performing regression on best performing dataset
def mlp(path,hl):
  df=pd.read_excel(path)
  lab_encoder = preprocessing.LabelEncoder()
  df['Date'] = lab_encoder.fit_transform(df['Date'])
  X=df.copy()
  Y=X['targetPrice']
  X.drop(['Profit','Unnamed: 0','targetPrice'],axis=1,inplace=True)

  scalerX = MinMaxScaler()
  scalerX.fit(X)
  X = scalerX.transform(X)
  trainX, testX, trainY, testY =train_test_split(X, Y, test_size=0.3,
random_state=24061)

  # Solve the problem using an artificial neural network
  regpenalty = 0.001
  clf = MLPRegressor(hidden_layer_sizes=hl, activation='relu', solver="adam",
```

```
                      alpha=regpenalty, early_stopping=True,
validation_fraction=0.35)
  clf.fit(trainX,trainY)
  sco=clf.score(testX,testY)
  pred=clf.predict(testX)
  return sco,testY,pred,testX,clf

hidden_layer=[(100,),(200,),(300,),(100,200),(200,100),(300,100),
              (100,200,300),(200,300,400)]
path1="/content/finalDataset.xlsx"
path2="/content/finalDataset 2.xlsx"
result=[]
for hl in hidden_layer:
  score,actual,pred,textX,clf=mlp(path1,hl)
  score1,actual1,pred1,textX1,clf1=mlp(path2,hl)
  result.append([hl,score,score1])
  print('hiddenlayer',hl,'activation','relu ---',score)
  print('hiddenlayer',hl,'activation','relu ---',score1)

# five year graph actual vs predicted

df1=pd.read_excel("/content/finalDataset.xlsx")
df=df1.copy()
lab_encoder = preprocessing.LabelEncoder()
df1['Date'] = lab_encoder.fit_transform(df1['Date'])
X=df1.copy()
Y=X['targetPrice']
X.drop(['Profit','Unnamed: 0','targetPrice'],axis=1,inplace=True)
scalerX = MinMaxScaler()
scalerX.fit(X)
X = scalerX.transform(X)
pred=clf.predict(X)
fig, ax = plt.subplots(figsize=(12, 6))
Y=df['targetPrice']
df['Date']  = pd.to_datetime(df['Date'])
print("dataset2")
plt.title('actual vs prediction')
plt.plot(df['Date'], Y, label = 'Actual')
plt.plot(df['Date'], pred, label = 'Prediction')
plt.legend()
plt.show()

df1=pd.read_excel("/content/finalDataset 2.xlsx")
df=df1.copy()
lab_encoder = preprocessing.LabelEncoder()
df1['Date'] = lab_encoder.fit_transform(df1['Date'])
X=df1.copy()
Y=X['targetPrice']
X.drop(['Profit','Unnamed: 0','targetPrice'],axis=1,inplace=True)
scalerX = MinMaxScaler()
scalerX.fit(X)
X = scalerX.transform(X)
pred=clf1.predict(X)

fig, ax = plt.subplots(figsize=(12, 6))
Y=df['targetPrice']
df['Date']  = pd.to_datetime(df['Date'])
print("dataset3")
plt.title('actual vs prediction')
```

```
plt.plot(df['Date'], Y, label = 'Actual')
plt.plot(df['Date'], pred, label = 'Prediction')
plt.legend()
plt.show()

# 3 month graph actual vs predicted

fig, ax = plt.subplots(figsize=(12, 6))
plt.title('actual vs prediction')
plt.plot(df['Date'][1138:], Y[1138:], label = 'Actual')
plt.plot(df['Date'][1138:], pred[1138:], label = 'Prediction')
plt.legend()
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
plt.title('actual vs prediction')
plt.plot(df['Date'][1138:], Y[1138:], label = 'Actual')
plt.plot(df['Date'][1138:], pred[1138:], label = 'Prediction')
plt.legend()
plt.show()

# write regression score to excel
overall=pd.DataFrame.from_records(result,columns=['hidden_layer','dataset1_acc','da
taset2_acc'])

overall.to_excel("regression_accuracy.xlsx")
```