HW3

2021321148
Insik Cho

1.

o gradient descent

```
 #theta=회귀계수, alpha = 학습률, iteration = 반복수
def Gradient_descent( X ,  Y,  theta, alpha,iteration):
  def cost(X,Y,theta): #cost function(mse/2)
    m = len(Y)
    y = X @ theta
    j = sum(np.sqrt((y-Y)**2))/(2*m)
    return j

  j=1000
  for c in range(iteration): # 반복
    y = X@theta
    for i in np.arange(X.shape[1]): #for문을 이용해 각각의 회귀계수에 대해 계산
      theta[i] = theta[i] - alpha * ((sum((y-Y)*X.iloc[:,i]))/(X.shape[0]))
    j = cost(X,Y,theta)
    if c %10000  ==0:
      print("진행상황: ",c)
      print("cost: ", j)
      print("계수: ", theta)
  return theta
```

Gradient_descent라는 함수를 정의해 각 회귀계수에 대해 gradient를 계산, 기존의 회귀계수에서 빼는 방식으로 코드를 작성했습니다.

```
theta = np.zeros(shape = (X.shape[1]))
result = Gradient_descent(X,Y,theta,0.0001, 4000000)
```

θ_0 =(0,0,0,0,0)에서 시작해 학습률 0.0001로 400000회 반복하는 경사하강법을 실행한 결과,

```
진행상황:  3960000
cost:  192.37566889773444
계수:  [3.52627877e+03 9.00303813e+01 1.26905639e+00 2.34068669e+01
 7.22450469e+02]
진행상황:  3970000
cost:  192.3756688262445
계수:  [3.52628238e+03 9.00301274e+01 1.26905471e+00 2.34068510e+01
 7.22450726e+02]
진행상황:  3980000
cost:  192.37566875655708
계수:  [3.52628591e+03 9.00298799e+01 1.26905308e+00 2.34068355e+01
 7.22450977e+02]
진행상황:  3990000
cost:  192.37566868862692
계수:  [3.52628934e+03 9.00296386e+01 1.26905149e+00 2.34068204e+01
 7.22451221e+02]
```

cost 및 회귀계수가 수렴하는 모습을 확인할 수 있었습니다.

o statsmodels

```
statols = sm.OLS(Y,X)
stat_fitted = statols.fit()
stats_results = stat_fitted.params
```

문제에 요구된 바와 같이 statsmodels 패키지를 이용해 선형회귀를 했고,

```
import sys
sys.stdout = open(out_name,'w')

print("              ")
print("result check by gradient descent method")
print("---------------")
print("constant =   ", result[0])
print("beta1 =  ", result[1])
print("beta2 =  ", result[2])
print("beta3 =  ", result[3])
print("beta4 =  ", result[4])

print("              ")
print("result check by statsmodels")
print("---------------")
print("constant =   ", stats_results.iloc[0])
print("beta1 =  ", stats_results.iloc[1])
print("beta2 =  ", stats_results.iloc[2])
print("beta3 =  ", stats_results.iloc[3])
print("beta4 =  ", stats_results.iloc[4])
```

result.txt 파일로 결과물이 출력되도록 했습니다.

결과를 비교한 바는 다음과 같습니다.

```
result check by gradient descent method
---------------
constant =    3526.29269
beta1 =   90.0294034
beta2 =   1.26904994
beta3 =   23.4068056
beta4 =   722.451459

result check by statsmodels
---------------
constant =    3526.4221106890086
beta1 =   90.02031094294053
beta2 =   1.2689900146697224
beta3 =   23.40623576800519
beta4 =   722.4606713810764
```

Gradient descent 결과 정답(OLS)에 수렴한 것을 확인할 수 있었습니다.

2.

o regression / classfication 선택

```
def homework3():
  train_data_name=input("Enter the name of train data file [(ex) harris.dat] : ") # data name
  test_data_name=input("Enter the name of test data file [(ex) harris.dat] : ")
  coding_fm=int(input("Select the data coding format(1 = 'a b c' or 2 = 'a,b,c'): ")) # data separator
  separator_fm={coding_fm ==1 : " "}.get(True, ",")
  res_pos=int(input("Enter the column position of the response variable : [from 1 to p] : "))
  header=input("Does the data have column header? (y/n) : ")
  if(header=="y") : trdata=pd.read_csv(train_data_name, sep=separator_fm) # loading data
  else : trdata=pd.read_csv(train_data_name, sep=separator_fm, header=None) # loading data
  if(header=="y") : tstdata=pd.read_csv(test_data_name, sep=separator_fm) # loading data
  else : tstdata=pd.read_csv(test_data_name, sep=separator_fm, header=None) # loading data
  out_name=input("Enter the output file name to export [(ex) result.txt] : ")

  choice = input("Do you want classification or regression? (c/r) :")
  if(choice =="c") : confusion(trdata,tstdata,res_pos,out_name)
  if(choice =="r") : LinearRegression(trdata,tstdata,res_pos,out_name)
```

hw2까지 사용하던 코드를 변형해, 함수 입력시 train/tst data 요구/ 반응변수 위치 요구/ heaaer 요구 등에 더해 1.classification /2. regression까지 요구하는 함수를 만들었습니다. 위 함수에서 c (classification)입력 시 문제에서 요구한 confusion matrix가 출력되고, r (regression)입력 시 hw2에서 요구했던 결과치가 출력되는 방식입니다.

o Linear Discriminant Analysis

문제에서 요구한 LDA는

```
def LDApredict(trdata, tstdata,res_pos):
  mu = []
  cov= []
  prior = []
  for i in np.arange(len(np.unique(trdata[res_pos-1]))): #평균, 공분산, prior에 관한 추정치를 계산
    mu.append(np.mean(trdata[trdata[res_pos-1]==(i+1)].drop([trdata.columns[res_pos-1]], axis=1 ), axis=0))
    cov.append(((trdata[trdata[res_pos-1]==(i+1)].drop([trdata.columns[res_pos-1]], axis=1 )-
                np.mean(trdata[trdata[res_pos-1]==(i+1)].drop([trdata.columns[res_pos-1]], axis=1 ), axis=0))).T
              @ ((trdata[trdata[res_pos-1]==(i+1)].drop([trdata.columns[res_pos-1]], axis=1 )-
                  np.mean(trdata[trdata[res_pos-1]==(i+1)].drop([trdata.columns[res_pos-1]], axis=1 ), axis=0))))
    prior.append(trdata[trdata[res_pos-1]==(i+1)].shape[0]/trdata.shape[0])
  pooled_cov = sum(cov) / (trdata.shape[0] - len(np.unique(trdata[res_pos-1]))) #공분산 계산

  pred=[]
  tstX= tstdata.drop([tstdata.columns[res_pos-1]], axis=1) #예측 때 반응변수 미리 제거
  for i in np.arange(tstX.shape[0]):#for문을 이용해 각 data에 관해 classification 예측
    C=[]
    for j in np.arange(len(np.unique(trdata[res_pos-1]))): # 각각의 class에 대해 discriminant를 계산
      C.append((mu[j].T @ np.linalg.inv(pooled_cov)@ tstX.iloc[i]) - (1/2)*(mu[j].T @ np.linalg.inv(pooled_cov)
      @ mu[j]) +np.log(prior[j]) )
    pred.append((np.argmax(C)+1)) # discriminant가 가장 높게 나타난 class를 예측치로 배정
  pred = pd.DataFrame(pred)
  return pred
```

위와 같은 방식으로 수업 시간에 배운 내용을 그대로 코드로 작성하였습니다.

o confusion matrix

```
def confusion(trdata,tstdata,res_pos,out_name):
  predtr = LDApredict(trdata,trdata,res_pos)
  predtst = LDApredict(trdata,tstdata,res_pos)
  confusion_tr = confusion_matrix(trdata[res_pos-1], predtr)
  confusion_tst = confusion_matrix(tstdata[res_pos-1], predtst)

  accu_tr = 0
  for i in range(len(np.unique(trdata[res_pos-1]))):
    accu_tr = accu_tr + confusion_tr[i][i]
  accuracy_tr = accu_tr / trdata.shape[0]

  accu_tst = 0
  for i in range(len(np.unique(trdata[res_pos-1]))):
    accu_tst = accu_tst + confusion_tst[i][i]
  accuracy_tst = accu_tst / tstdata.shape[0]

  sys.stdout = open(out_name,'w')
  print('confusion matrix (training)')
  print('-----------------------------------')
  print('           predicted class ₩n Actual 1 ' ,confusion_tr[0], '₩n class  2 ', confusion_tr[1])
  for i in range(2,len(np.unique(trdata[res_pos-1])) ):
    print(f'        {i} ', confusion_tr[i])
  print('model summary')
  print('-----------------------------------')
  print('Overall accuracy = ' ,accuracy_tr)

  print('₩n₩nconfusion matrix (test)')
  print('-----------------------------------')

  print('           predicted class ₩n Actual 1 ' ,confusion_tst[0], '₩n class  2 ', confusion_tst[1])
  for i in range(2,len(np.unique(tstdata[res_pos-1])) ):
    print(f'        {i} ', confusion_tst[i])
  print('model summary')
  print('-----------------------------------')
  print('Overall accuracy = ' ,accuracy_tst)
```

문제에서 요구한 형태로 confusion matrix가 출력되도록 만든 함수입니다.
이때 confusion()함수를 입력시 LDA까지 구동돼 결과물이 출력되도록 했습니다.

o result

다음과 같이 homework3() 함수 및 요구사항을 입력 시,

```
homework3()
```

```
Enter the name of train data file [(ex) harris.dat] : veh.dat
Enter the name of test data file [(ex) harris.dat] : vehtest.dat
Select the data coding format(1 = 'a b c' or 2 = 'a,b,c'): 2
Enter the column position of the response variable : [from 1 to p] : 19
Does the data have column header? (y/n) : n
Enter the output file name to export [(ex) result.txt] : result.txt
Do you want classification or regression? (c/r) :c
```

```
confusion matrix (training)
-----------------------------------
                 predicted class
 Actual 1   [76 23  2  4]
 class  2   [30 76  1  3]
        2   [  2   1 106   1]
        3   [ 1  0  0 99]
model summary
-----------------------------------
Overall accuracy =  0.84


confusion matrix (test)
-----------------------------------
                 predicted class
 Actual 1   [48 30  4  4]
 class  2   [23 44 10  8]
        2   [ 0  0 85  1]
        3   [ 0  0  2 77]
model summary
-----------------------------------
Overall accuracy =  0.7559523809523809
```

문제의 예시와 같은 형태로 결과물이 출력됩니다.