HW9

2021321148 INSIK CHO

1. bootstrap

```
def bootstrap(dataset, B=101):
  boot = list()
  oob = list()
  while len(boot)<B:
    sample = list()
    while len(sample) < len(dataset):
      index = random.randrange(len(dataset))
      sample.append(dataset.iloc[index])
    Z = pd.DataFrame(sample)
    oob.append(data[data.index.isin(Z[0].index) == False])
    boot.append(Z)
  return boot, oob

def most_frequent(data):
    return max(data, key=data.count)
```

처음에는 부트스트랩 코드를 작성했습니다.

2. RandomFeature $ bagging

```
def rfLDA(data,  n_bootstrapping=101):
  clf = LinearDiscriminantAnalysis()
  bag = {}
  b= bootstrap(data, B = n_bootstrapping)
  ooberr = []
  lab = []
  p =pd.DataFrame(index = range(n_bootstrapping), columns = range(len(data.columns)-1))
  m = np.round((len(data.columns)-1)/2)
  for i in range(n_bootstrapping):
    boot = b[0]
    oob = b[1]
    label = random.sample(list(boot[i].drop([data.columns[res_pos-1]], axis=1 ).columns.values),int(m))
    lab.append(label)
    newX = boot[i].drop([data.columns[res_pos-1]], axis=1 ).loc[:,label]
    bag[f'LDA{i}'] = clf.fit(newX,boot[i].iloc[:, res_pos-1 ].values)
    ooby = oob[i].iloc[:, res_pos-1 ].values
    oobpredy = clf.predict(newX )
    ooberr.append(len(ooby[(ooby != oobpredy) ==True ]) / len(ooby))

    for j in label:
      poob = oob[i].drop([data.columns[res_pos-1]], axis=1 ).iloc[:,label]
      poob.reset_index(drop = True, inplace = True)
      H = pd.Series(data = poob.loc[:,j].sample(n=len(oob[i]),replace = False))
      H.reset_index(drop = True,inplace = True)
      poob[j] = H
      poobpredy = clf.predict(poob)
      p[j][i] = (len(ooby[(ooby != poobpredy) ==True ]) / len(ooby))
```

```
  #i번째 변수의 중요도 계산
  f=[]
  for j in list(data.drop([data.columns[res_pos-1]], axis=1 ).columns.values):
    ppp = []
    icollect = []
    for i in range(n_bootstrapping):
      if j in lab[i]:
        icollect.append(i)
        ppp.append(p[j][i])
    eee=[]
    for k in icollect:
      eee.append(ooberr[k])
    eee= np.array(eee)
    ppp = np.array(ppp)
    d = ppp - eee
    f.append((np.sum(d)/n_bootstrapping)/(np.std(d)*np.sqrt(len(d)-1)/np.sqrt(n_bootstrapping-1)))
  return bag, f ,lab
```

Random Feautre LDA 코드를 작성했습니다. 위에서 m 값만 m = 변수수로 고치면 bagging이 되도록 코드를 만들었습니다.

```
def result(tstdata, method,n_bootstrapping=101):
  print('Variable Importance: ')
  for i in range(len(data.columns)-1):
    print(f'   X{i+1}:', Z[1][i])

  predy = []
  for i in range(n_bootstrapping):
    X = tstdata.loc[:,Z[2][i]]
    pred = Z[0][f'LDA{i}'].predict(X)
    predy.append(pred)

  newy=[]
  for j in range(len(tstdata)):
    r = []
    for i in range(n_bootstrapping):
      r.append(predy[i][j])
    newy.append(most_frequent(r))


  print('    ')
  print( 'Confusion Matrix( LDA - ', method, ')')
  print('-----------------------------------------------------')
  confusion_tst = confusion_matrix(tstdata.iloc[:,res_pos-1], newy)

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

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

마지막으로 결과를 출력하는 코드를 작성한 후,결과 출력시

(bagging)

```
Variable Importance:
    X1: 9.191956635420206
    X2: 9.94456788620993
    X3: 8.447757080681692
    X4: 16.79238304322505
    X5: 12.201714957444091
    X6: 6.733062190261799
    X7: 8.436361136530873
    X8: 9.05875609119743
    X9: 4.970734080666604
    X10: 10.741157882257033
    X11: 5.740033851342791
    X12: 8.60058865296781
    X13: 7.620249622735597
    X14: 6.141212416265852
    X15: 6.647054100577157
    X16: 7.609276888952221
    X17: 9.929617374221504
    X18: 10.178064837030117

Confusion Matrix( LDA -  Bagging )
------------------------------------------------
                predicted class
 Actual  1   [28  0 58  0]
 class   2   [28  0 57  0]
         3   [54  0 32  0]
         4   [73  0  6  0]
model summary
-----------------------------------
Overall accuracy =  0.17857142857142858
```

(Random Feature)

```
Variable Importance:
    X1: 5.583533016458313
    X2: 3.263435415166492
    X3: 5.819877947695917
    X4: 2.8902154452581983
    X5: 3.4008266158783558
    X6: 4.803576219608648
    X7: 4.966556393551183
    X8: 9.403179809690439
    X9: 4.748087201060037
    X10: 5.551290864964803
    X11: 4.841907731420092
    X12: 5.847589999580107
    X13: 4.711824166107802
    X14: 4.788792743169575
    X15: 5.255200235527969
    X16: 4.709133504751177
    X17: 5.216397434025591
    X18: 3.580719183060817

Confusion Matrix( LDA -  Random Feature )
-----------------------------------------------
           predicted class
 Actual 1  [ 0 48  0 38]
 class  2  [ 0 51  0 34]
        3  [ 0 66  0 20]
        4  [ 0 78  0  1]
model summary
----------------------------------
Overall accuracy =  0.15476190476190477
```