**Lab5**

201635825 오성원

**Lab5-1**

|  |
| --- |
| Source Code |
| import pandas as pd  import numpy as np  from sklearn import tree  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import accuracy\_score  import seaborn as sn  import warnings  warnings.filterwarnings(action='ignore')  #Read the test datafile and save it in iris.  iris=pd.read\_csv('C:/python\_file/Iris\_test\_dataset.csv', encoding='utf-8')  target=iris['Species']  #Read 10 training datasets and save them in the samples list  samples=[]  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (1).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (2).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (3).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (4).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (5).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (6).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (7).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (8).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (9).csv', encoding='utf-8')  samples.append(sample)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (10).csv', encoding='utf-8')  samples.append(sample)  """  The bagging function is a function  that returns the predicted values by receiving the training data and test data.  """  def bagging(train\_data, test\_data):  #Create test and train data.  predict=[]  train=train\_data.copy()  test=test\_data.copy()  test\_x=test\_data  test\_x.drop(['Species'], axis=1, inplace=True)    for i in range(10):  train\_y=train[i]['Species']  train\_x=train[i]  train\_x.drop(['Species'], axis=1, inplace=True)  #Create model by inserting train data into the DecisionTreeCalssifier.  tree\_model=tree.DecisionTreeClassifier().fit(train\_x, train\_y)  #Predict results using model  result=tree\_model.predict(test\_x)  #Insert the generated result into the predict list.  predict.append(result)  return predict  #Obtain the prediction through the bagging function.  prediction=bagging(samples,iris)  predicted=[]  for i in range(len(iris)):  #variables for counting for each flower  se=0  ve=0  vi=0  max\_vote=-1  max\_index=''  #Conduct a test data length and voting each data.  for j in range(10):  if(prediction[j][i]=='Iris-setosa'):  se+=1  elif(prediction[j][i]=='Iris-versicolor'):  ve+=1  elif(prediction[j][i]=='Iris-virginica'):  vi+=1  if(max\_vote < se):  max\_vote=se  max\_index='Iris-setosa'  if(max\_vote < ve):  max\_vote=ve  max\_index='Iris-versicolor'  if(max\_vote < vi):  max\_vote=ve  max\_index='Iris-virginica'  predicted.append(max\_index)  #Construct a confusion matrix using actual data and predicted result data.  actual\_data=np.array(target)  data={'y\_Predicted':predicted, 'y\_Actual': actual\_data}  df2=pd.DataFrame(data, columns=['y\_Actual', 'y\_Predicted'])  confusion\_matrix=pd.crosstab(df2['y\_Actual'], df2['y\_Predicted'], rownames=['Actual'], colnames=['Predicted'],margins=True)  print('---------------------------Confusion Matrix---------------------------')  print(confusion\_matrix)  print()  print('Accuracy\_score')  print(accuracy\_score(predicted, actual\_data)) |

|  |
| --- |
| Screen shot |
|  |

**Lab5-2**

|  |
| --- |
| Source Code |
| import pandas as pd  import numpy as np  import random  from sklearn import tree  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import accuracy\_score  import seaborn as sn  import warnings  warnings.filterwarnings(action='ignore')  #Read the test datafile and save it in iris.  iris=pd.read\_csv('C:/python\_file/Iris\_test\_dataset.csv', encoding='utf-8')  target=iris['Species']  #Read the 10 sample data and put them all together in df.  sample1=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (1).csv', encoding='utf-8')  sample2=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (2).csv', encoding='utf-8')  df=pd.concat([sample1, sample2],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (3).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (4).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (5).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (6).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (7).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (8).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (9).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  sample=pd.read\_csv('C:/python\_file/Iris\_train\_datasets/Iris\_bagging\_dataset (10).csv', encoding='utf-8')  df=pd.concat([df,sample],axis=0)  #Override index of data in df  a=list(range(len(df)))  df.index=a  """  The bagging function is a function  that returns the predicted values by receiving the training data and test data.  """  def bagging(train\_data, test\_data, k):  #Create k training datasets containing 30 data randomly using bootstrap method.  sample=[]  for j in range(k):  df2 = pd.DataFrame(index=range(0), columns=['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species'])  for i in range(30):  ran=random.randrange(300)  temp=df.loc[ran]  df2.loc[i]=temp  sample.append(df2)  #Create test and train data.  predict=[]  train=sample.copy()  test=test\_data.copy()  test\_x=test  test\_x.drop(['Species'], axis=1, inplace=True)    for i in range(k):  train\_y=train[i]['Species']  train\_x=train[i]  train\_x.drop(['Species'], axis=1, inplace=True)  #Create model by inserting train data into the DecisionTreeCalssifier.  tree\_model=tree.DecisionTreeClassifier(max\_depth=1).fit(train\_x, train\_y)    #Predict results using model  result=tree\_model.predict(test\_x)  #Insert the generated result into the predict list.  predict.append(result)    return predict  """  Function to output the confusion matrix  by receiving the K value and releasing the enemble reading.  """  def ensemble(k):  #Recall bagging function using k value  prediction=bagging(df, iris, k)  predicted=[]  for i in range(len(iris)):  #variables for counting for each flower  se=0  ve=0  vi=0  max\_vote=-1  max\_index=''  #Conduct a test data length and voting each data.  for j in range(k):  if(prediction[j][i]=='Iris-setosa'):  se+=1  elif(prediction[j][i]=='Iris-versicolor'):  ve+=1  elif(prediction[j][i]=='Iris-virginica'):  vi+=1  if(max\_vote < se):  max\_vote=se  max\_index='Iris-setosa'  if(max\_vote < ve):  max\_vote=ve  max\_index='Iris-versicolor'  if(max\_vote < vi):  max\_vote=ve  max\_index='Iris-virginica'  predicted.append(max\_index)  #Construct a confusion matrix using actual data and predicted result data.  actual\_data=np.array(target)  data={'y\_Predicted':predicted, 'y\_Actual': actual\_data}  df2=pd.DataFrame(data, columns=['y\_Actual', 'y\_Predicted'])  confusion\_matrix=pd.crosstab(df2['y\_Actual'], df2['y\_Predicted'], rownames=['Actual'], colnames=['Predicted'],margins=True)  print('=================== K={0} ==================='.format(k))  print('---------------------------Confusion Matrix---------------------------')  print(confusion\_matrix)  print()  print('Accuracy\_score')  print(accuracy\_score(predicted, actual\_data),'\n')  #Implemented when K is 5,10,20,50,100, respectively.  ensemble(5)  ensemble(10)  ensemble(20)  ensemble(50)  ensemble(100) |

|  |
| --- |
| Screen shot |
|  |