**Machine Learning PHW #1 - group 4**

|  |
| --- |
| Source Code |
| import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  import warnings  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.preprocessing import StandardScaler  from sklearn.preprocessing import MinMaxScaler  from sklearn.tree import DecisionTreeClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn import svm  from sklearn.model\_selection import GridSearchCV  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  warnings.filterwarnings(action='ignore')  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.max\_rows', None)  pd.set\_option('display.max\_colwidth', None)  # from google.colab import drive  # drive.mount('/content/drive')  **###Data load###**  df = pd.read\_csv("data/breast-cancer-wisconsin.data", header=None)  # df = pd.read\_csv("/content/drive/Shareddrives/머신러닝/breast-cancer-wisconsin.data",header=None)  df.columns = ['Sample code number', 'Clump Thickness ', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',  'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli',  'Mitoses', 'Class']  **###Data preprocessing###**  **#1) Remove missing value**  df = df.replace('?', np.NaN)  print(df.isnull().sum())  df.fillna(0, inplace=True)  print(df.isnull().sum())  df.drop('Sample code number', axis=1, inplace=True)  X, y = df.drop(['Class'], axis=1), df['Class']  **###Set model parameters variables###**  **#1) Parameters by each model**  # model, scaler, K parameter  models = ['DecisionTreeClassifier', 'LogisticRegression', 'svm.SVC']  scalers = [StandardScaler(), MinMaxScaler()]  Ks = [5, 10, 15]  # decision\_tree\_parameter  criterions = ['gini', 'entropy']  splitters = ['best', 'random']  max\_depths = [1, 10, 100]  # logi\_tree\_parameter  solvers = ['lbfgs', 'sag']  max\_iters = [50, 100, 200]  # svm\_parameter  Cs = [0.1, 1]  gammas = [0.1, 0.3, 0.5, 1, 5]  kernels = ['rbf', 'sigmoid']  max\_iters = [50, 100, 200]  # Dictionary parameter  dt\_parameter = {  'criterion': ['gini', 'entropy'],  'splitter': ['best', 'random'],  'max\_depth': [1, 10, 100]  }  lg\_parameter = {  'solver': ['lbfgs', 'sag'],  'max\_iter': [50, 100, 200]  }  svm\_parameter = {  'C': [0.1, 1],  'gamma': [0.1, 0.3, 0.5, 1, 5],  'kernel': ['rbf', 'sigmoid'],  'max\_iter': [50, 100, 200]  }  # make parameter list  total\_parameter = [scalers, Ks, criterions, splitters, max\_depths, solvers, max\_iters, Cs, gammas, kernels, max\_iters]  **###Search the best model using hard-coded parameters###**  def create\_model(X, y, models, params):  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, random\_state=42)  accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  for model in models:  if model == 'DecisionTreeClassifier':  DT\_accuracy = DT\_train\_model(X\_train, X\_test, y\_train, y\_test, params[0], params[1], params[2], params[3],  params[4])  accuracy = pd.concat([accuracy, DT\_accuracy])  print("\n")  elif model == 'LogisticRegression':  LG\_accuracy = LG\_train\_model(X\_train, X\_test, y\_train, y\_test, params[0], params[1], params[5], params[6])  accuracy = pd.concat([accuracy, LG\_accuracy])  print("\n")  elif model == 'svm.SVC':  SVM\_accuracy = SVM\_train\_model(X\_train, X\_test, y\_train, y\_test, params[0], params[1], params[7], params[8],  params[9], params[10])  accuracy = pd.concat([accuracy, SVM\_accuracy])  print("\n")  else:  print("Input model Error")  return accuracy  **#1) Train function for Decision Tree model**  **#Input**  #-scaler(scaler)  #-DT\_parameter(criterions,splitters)  #-kfold(ks)  **#Output**  #best 5 model dataframe  def DT\_train\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, criterions, splitters, max\_depths):  DT\_accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  print("========================")  print("[DecisionTreeClassifier]")  print("========================")  for scaler in scalers:  for criterion in criterions:  for splitter in splitters:  for max\_depth in max\_depths:  for K in Ks:  # do use Scaler  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  # build DecisionTreeClassifier model and fit data  DT = DecisionTreeClassifier(criterion=criterion, splitter=splitter, max\_depth=max\_depth,  random\_state=42)  # do k-fold validation (cv=k)  score = cross\_val\_score(DT, X\_train, y\_train, cv=K)  score = np.mean(score)  print(  "DecisionTreeClassifier Average of scores : %f (scaler = %s, k = %s, criterion = %s, splitter = %s, max\_depth = %s)" % (  score, scaler, K, criterion, splitter, max\_depth))  data\_to\_insert = {'model': 'DecisionTreeClassifier', 'scaler': scaler, 'K': K,  'parm': '{\'criterion\' : %s, \'splitter\' : %s, \'max\_depth\' : %s}' % (  criterion, splitter, max\_depth), 'score': score}  DT\_accuracy = DT\_accuracy.append(data\_to\_insert, ignore\_index=True)  DT\_accuracy = DT\_accuracy.nlargest(5, 'score')  return DT\_accuracy  **#2) Train function for Linear regression model**  **#Input**  #-scaler(scaler)  #-Linear regression\_parameter(solvers,max\_iters)  #-kfold(ks)  **#Output**  #best 5 model dataframe  def LG\_train\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, solvers, max\_iters):  LG\_accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  print("========================")  print("[LogisticRegression]")  print("========================")  for scaler in scalers:  for solver in solvers:  for max\_iter in max\_iters:  for K in Ks:  # do use Scaler  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  # build LogisticRegression model and fit data  LG = LogisticRegression(solver=solver, max\_iter=max\_iter, random\_state=42)  # do k-fold validation (cv=k)  score = cross\_val\_score(LG, X\_train, y\_train, cv=K)  score = np.mean(score)  print(  "LogisticRegression Average of scores : %f (scaler = %s, k = %s, solver = %s, max\_iter = %s)" % (  score, scaler, K, solver, max\_iter))  data\_to\_insert = {'model': 'LogisticRegression', 'scaler': scaler, 'K': K,  'parm': '{\'solver\' : %s, \'max\_iter\' : %s}' % (solver, max\_iter),  'score': score}  LG\_accuracy = LG\_accuracy.append(data\_to\_insert, ignore\_index=True)  LG\_accuracy = LG\_accuracy.nlargest(5, 'score')  return LG\_accuracy  **#3) Train function for SVM model**  **#Input**  #-scaler(scaler)  #-SVM parameter(Cs,max\_iters)  #-kfold(ks)  **#Output**  #best 5 model dataframe  def SVM\_train\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, Cs, gammas, kernels, max\_iters):  SVM\_accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  print("========================")  print("[SVM]")  print("========================")  for scaler in scalers:  for C in Cs:  for gamma in gammas:  for kernel in kernels:  for max\_iter in max\_iters:  for K in Ks:  # do use Scaler  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  # build SVM model and fit data  SVM = svm.SVC(C=C, gamma=gamma, kernel=kernel, max\_iter=max\_iter, random\_state=42)  # do k-fold validation (cv=k)  score = cross\_val\_score(SVM, X\_train, y\_train, cv=K)  score = np.mean(score)  print(  "SVM Average of scores : %f (scaler = %s, k = %s, C = %s, gamma = %s, kernel = %s, max\_iter = %s)" % (  score, scaler, K, C, gamma, kernel, max\_iter))  data\_to\_insert = {'model': 'SVM', 'scaler': scaler, 'K': K,  'parm': '{\'C\' : %s, \'gamma\' : %s, \'kernel\' : %s, \'max\_iter\' : %s}' % (  C, gamma, kernel, max\_iter), 'score': score}  SVM\_accuracy = SVM\_accuracy.append(data\_to\_insert, ignore\_index=True)  SVM\_accuracy = SVM\_accuracy.nlargest(5, 'score')  return SVM\_accuracy  **###Use single major function ‘create\_model’###**  accuracy = create\_model(X, y, models, total\_parameter)  # sort value by score by descending order  accuracy = accuracy.sort\_values(by=['score'], ascending=False)  # reset index number and restore  accuracy = accuracy.reset\_index(drop=True)  print(accuracy)  # checking model parameter  def check\_model(X, y, models, scalers, Ks, dt\_parameter, lg\_parameter, svm\_parameter):  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, random\_state=42)  accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  for model in models:  if model == 'DecisionTreeClassifier':  DT\_accuracy = DT\_grid\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, dt\_parameter)  accuracy = pd.concat([accuracy, DT\_accuracy])  print("\n")  elif model == 'LogisticRegression':  LG\_accuracy = LG\_grid\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, lg\_parameter)  accuracy = pd.concat([accuracy, LG\_accuracy])  print("\n")  elif model == 'svm.SVC':  SVM\_accuracy = SVM\_grid\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, svm\_parameter)  accuracy = pd.concat([accuracy, SVM\_accuracy])  print("\n")  else:  print("Input model Error")  return accuracy  **###GridsearchCV to automatically discover the best models###**  def DT\_grid\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, dt\_parameter):  DT\_accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  print("==========================================")  print("[DecisionTreeClassifier With GridSearchCV]")  print("==========================================")  for scaler in scalers:  print("------------------------------------------")  print("[%s]" % scaler)  print("------------------------------------------")  for K in Ks:  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  DT = DecisionTreeClassifier(random\_state=42);  grid\_DT = GridSearchCV(DT, param\_grid=dt\_parameter, cv=K, scoring="accuracy")  grid\_DT.fit(X\_train, y\_train)  print('GridSearchCV Best parameters (using k : %s) : ' % K, grid\_DT.best\_params\_)  print('GridSearchCV Best accuracy : %0.6f' % grid\_DT.best\_score\_)  data\_to\_insert = {'model': 'DecisionTreeClassifier', 'scaler': scaler, 'K': K,  'parm': grid\_DT.best\_params\_, 'score': grid\_DT.best\_score\_}  DT\_accuracy = DT\_accuracy.append(data\_to\_insert, ignore\_index=True)  DT\_accuracy = DT\_accuracy.nlargest(1, 'score')  return DT\_accuracy  def LG\_grid\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, lg\_parameter):  LG\_accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  print("==========================================")  print("[LogisticRegression With GridSearchCV]")  print("==========================================")  for scaler in scalers:  print("------------------------------------------")  print("[%s]" % scaler)  print("------------------------------------------")  for K in Ks:  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  LG = LogisticRegression(random\_state=42);  grid\_LG = GridSearchCV(LG, param\_grid=lg\_parameter, cv=K, scoring="accuracy")  grid\_LG.fit(X\_train, y\_train)  print('GridSearchCV Best parameters (using k : %s) : ' % K, grid\_LG.best\_params\_)  print('GridSearchCV Best accuracy : %0.6f' % grid\_LG.best\_score\_)  data\_to\_insert = {'model': 'LogisticRegression', 'scaler': scaler, 'K': K,  'parm': grid\_LG.best\_params\_, 'score': grid\_LG.best\_score\_}  LG\_accuracy = LG\_accuracy.append(data\_to\_insert, ignore\_index=True)  LG\_accuracy = LG\_accuracy.nlargest(1, 'score')  return LG\_accuracy  def SVM\_grid\_model(X\_train, X\_test, y\_train, y\_test, scalers, Ks, svm\_parameter):  SVM\_accuracy = pd.DataFrame(columns=['model', 'scaler', 'K', 'parm', 'score'])  print("==========================================")  print("[SVM With GridSearchCV]")  print("==========================================")  for scaler in scalers:  print("------------------------------------------")  print("[%s]" % scaler)  print("------------------------------------------")  for K in Ks:  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  SVM = svm.SVC(random\_state=42);  grid\_SVM = GridSearchCV(SVM, param\_grid=svm\_parameter, cv=K, scoring="accuracy")  grid\_SVM.fit(X\_train, y\_train)  print('GridSearchCV Best parameters (using k : %s) : ' % K, grid\_SVM.best\_params\_)  print('GridSearchCV Best accuracy : %0.6f' % grid\_SVM.best\_score\_)  data\_to\_insert = {'model': 'SVM', 'scaler': scaler, 'K': K,  'parm': grid\_SVM.best\_params\_, 'score': grid\_SVM.best\_score\_}  SVM\_accuracy = SVM\_accuracy.append(data\_to\_insert, ignore\_index=True)  SVM\_accuracy = SVM\_accuracy.nlargest(1, 'score')  return SVM\_accuracy  **#Use single major function ‘check\_model’**  grid\_accuracy = check\_model(X, y, models, scalers, Ks, dt\_parameter, lg\_parameter, svm\_parameter)  # sort value by score by descending order  grid\_accuracy = grid\_accuracy.sort\_values(by=['score'], ascending=False)  # reset index number and restore  grid\_accuracy = grid\_accuracy.reset\_index(drop=True)  print(grid\_accuracy)  **###Analysis Part###**  # now we know the best parameters with GridSearchCV  # Analysis the model  print("========================")  print("[DecisionTreeClassifier]")  print("========================")  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, random\_state=42)  scaler = grid\_accuracy[grid\_accuracy['model'] == 'DecisionTreeClassifier'].scaler  scaler = scaler.array[0]  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  parm = grid\_accuracy[grid\_accuracy['model'] == 'DecisionTreeClassifier'].parm  parm\_dic = parm.array[0]  DT = DecisionTreeClassifier(criterion=parm\_dic['criterion'], max\_depth=parm\_dic['max\_depth'],  splitter=parm\_dic['splitter'], random\_state=42);  DT.fit(X\_train, y\_train)  y\_pred = DT.predict(X\_test)  print("------------------------")  print("parameters")  print("------------------------")  print(DT.get\_params())  print()  print("------------------------")  print("Accuracy")  print("------------------------")  print("Accuracy score (training) : %0.6f" % DT.score(X\_train, y\_train))  print("Accuracy score (testing) : %0.6f" % DT.score(X\_test, y\_test)) # same score -> accuracy\_score(y\_test, y\_pred)  dt\_cf = confusion\_matrix(y\_test, y\_pred)  dt\_mat = pd.DataFrame(dt\_cf)  plt.figure(figsize=(5, 3))  plt.title('DecisionTreeClassifier Confusion Matrix')  sns.heatmap(dt\_mat, annot=True, fmt='.1f')  plt.show()  print("---------------------")  print("Classification Report")  print("---------------------")  print(classification\_report(y\_test, y\_pred))  print("========================")  print("[LogisticRegression]")  print("========================")  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, random\_state=42)  scaler = grid\_accuracy[grid\_accuracy['model'] == 'LogisticRegression'].scaler  scaler = scaler.array[0]  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  parm = grid\_accuracy[grid\_accuracy['model'] == 'LogisticRegression'].parm  parm\_dic = parm.array[0]  LG = LogisticRegression(max\_iter=parm\_dic['max\_iter'], solver=parm\_dic['solver'], random\_state=42);  LG.fit(X\_train, y\_train)  y\_pred = LG.predict(X\_test)  print("------------------------")  print("parameters")  print("------------------------")  print(LG.get\_params())  print()  print("------------------------")  print("Accuracy")  print("------------------------")  print("Accuracy score (training) : %0.6f" % LG.score(X\_train, y\_train))  print("Accuracy score (testing) : %0.6f" % LG.score(X\_test, y\_test)) # same score -> accuracy\_score(y\_test, y\_pred)  lg\_cf = confusion\_matrix(y\_test, y\_pred)  lg\_cf\_mat = pd.DataFrame(lg\_cf)  plt.figure(figsize=(5, 3))  plt.title('LogisticRegression Confusion Matrix')  sns.heatmap(lg\_cf\_mat, annot=True, fmt='.1f')  plt.show()  print("---------------------")  print("Classification Report")  print("---------------------")  print(classification\_report(y\_test, y\_pred))  print("========================")  print("[SVM]")  print("========================")  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, random\_state=42)  scaler = grid\_accuracy[grid\_accuracy['model'] == 'SVM'].scaler  scaler = scaler.array[0]  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.fit\_transform(X\_test)  parm = grid\_accuracy[grid\_accuracy['model'] == 'SVM'].parm  parm\_dic = parm.array[0]  SVM = svm.SVC(C=parm\_dic['C'], gamma=parm\_dic['gamma'], kernel=parm\_dic['kernel'],  max\_iter=parm\_dic['max\_iter'], random\_state=42);  SVM.fit(X\_train, y\_train)  y\_pred = SVM.predict(X\_test)  print("------------------------")  print("parameters")  print("------------------------")  print(SVM.get\_params())  print()  print("------------------------")  print("Accuracy")  print("------------------------")  print("Accuracy score (training) : %0.6f" % SVM.score(X\_train, y\_train))  print("Accuracy score (testing) : %0.6f" % SVM.score(X\_test, y\_test)) # same score -> accuracy\_score(y\_test, y\_pred)  svm\_cf = confusion\_matrix(y\_test, y\_pred)  svm\_cf\_mat = pd.DataFrame(lg\_cf)  plt.figure(figsize=(5, 3))  plt.title('SVM Confusion Matrix')  sns.heatmap(svm\_cf\_mat, annot=True, fmt='.1f')  plt.show()  print("---------------------")  print("Classification Report")  print("---------------------")  print(classification\_report(y\_test, y\_pred)) |
| Missing value before preprocessing    Missing value after preprocessing    Consider all parameters of the decision tree      Consider all parameters of the Logistic Regression    Consider all parameters of the SVM                  5 high accuracy for each model    Find parameter with GridSearchCV        Best parameters for each model    Checking Accuracy Decision Tree model with Using best parameters    Checking Accuracy Logistic Regression model with Using best parameters    Checking Accuracy SVM with Using best parameters |

Details of task allocations

|  |  |  |  |
| --- | --- | --- | --- |
| 곽민선 | 김세빈 | 이예준 | 표지성 |
| lab meeting | lab meetings  idea discussion  construct the structure  write the code  write the result document | lab meeting  idea discussion  construct the structure  write the code  write the result document | lab meeting  idea discussion  construct the structure  write the code write the result document |

Contribution

|  |  |  |  |
| --- | --- | --- | --- |
| 곽민선 | 김세빈 | 이예준 | 표지성 |
| 0 | 33.33 | 33.33 | 33.34 |