**Machine Learning Term Project Output**

# Before preprocessing

텍스트이(가) 표시된 사진

자동 생성된 설명

# After preprocessing

텍스트이(가) 표시된 사진

자동 생성된 설명

# Grid Search

DecisionTreeClassifier(criterion='entropy')



DecisionTreeClassifier(criterion='gini')



LogisticRegression()

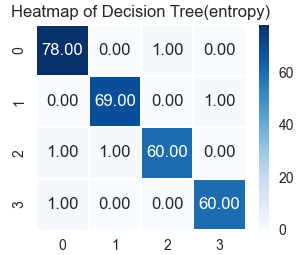
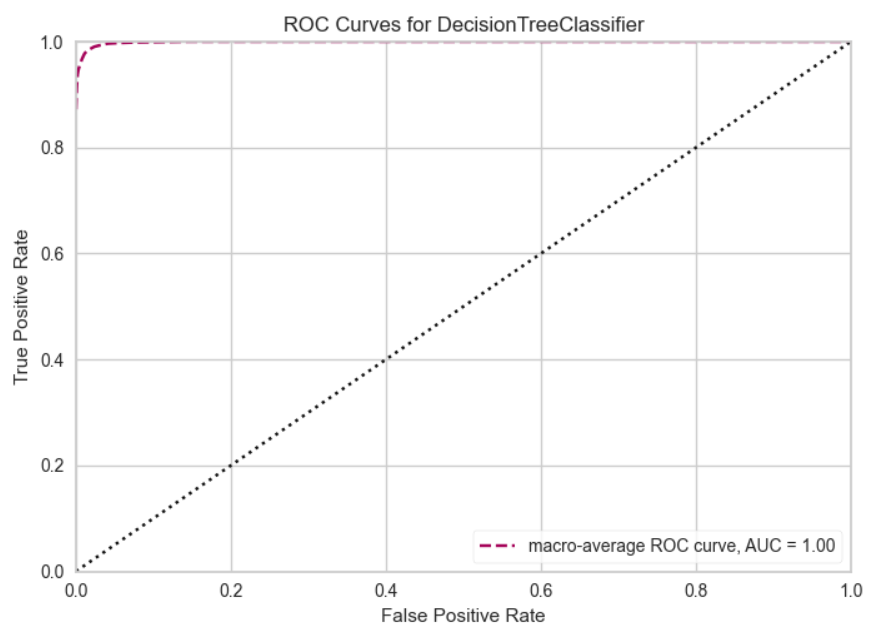


SVM()

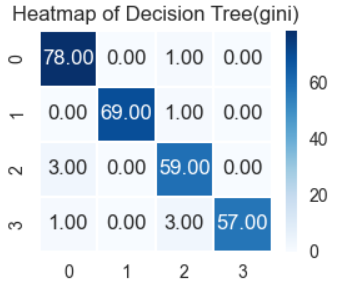
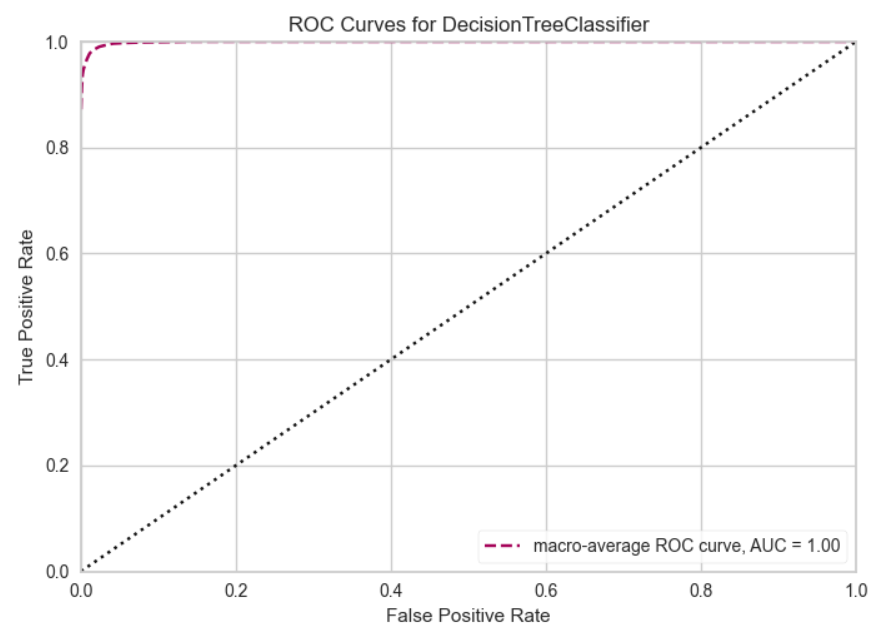


# Visualization

DecisionTreeClassifier(criterion='entropy')

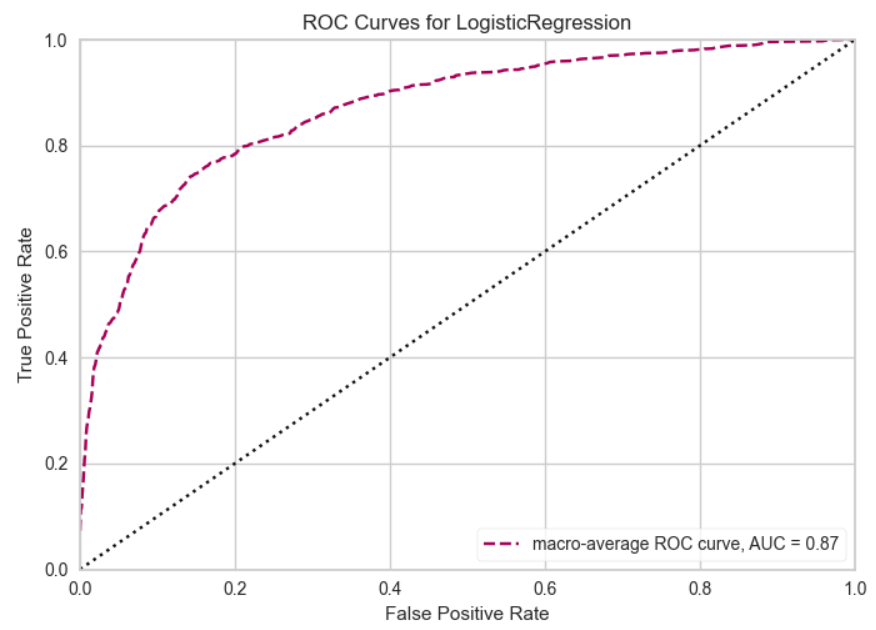
 

DecisionTreeClassifier(criterion='gini')

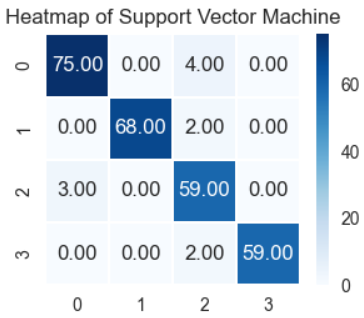
 

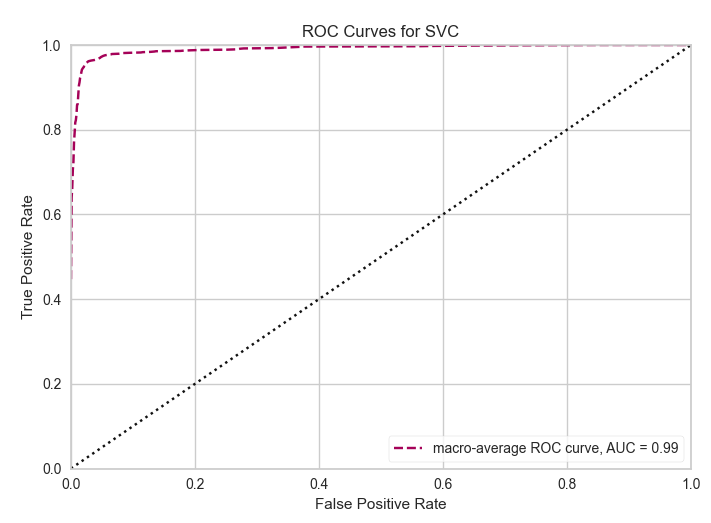
LogisticRegression()

테이블이(가) 표시된 사진

자동 생성된 설명 

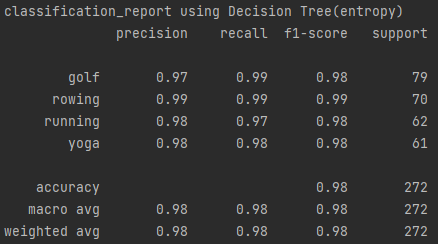
SVM()



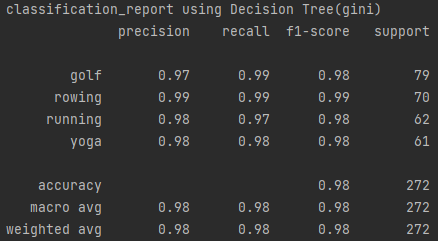


# Confusion Matrix

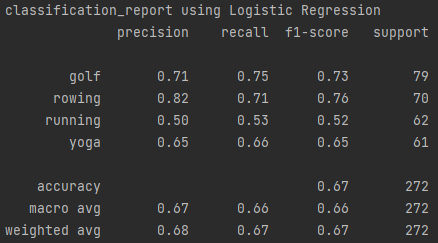
Decision Tree(entropy)



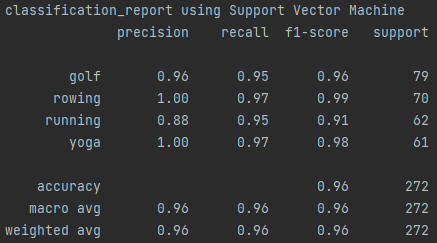
Decision Tree(gini)



Logistic Regression



SVM



# Code

import warnings  
  
from imblearn.over\_sampling import SMOTE  
from sklearn.preprocessing import StandardScaler  
  
warnings.filterwarnings(action='ignore')  
  
import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold, cross\_val\_score  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
import matplotlib.pyplot as plt  
from yellowbrick.classifier import ROCAUC  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix, classification\_report  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
import seaborn as sns  
  
dataset = pd.read\_csv('mpii\_human\_pose.csv')  
  
category\_df = dataset  
category\_df['new'] = np.nan  
  
cate\_list = dataset['Activity'].astype('category').values.categories  
  
# yoga  
category\_df.loc[category\_df['Activity'] == 'yoga, Power', 'new'] = 'yoga'  
category\_df.loc[category\_df['Activity'] == 'yoga, Nadisodhana', 'new'] = 'yoga'  
category\_df.loc[category\_df['Activity'] == 'stretching', 'new'] = 'yoga'  
# category\_df.loc[category\_df['Activity'] == 'pilates, general', 'new'] = 'yoga'  
print('yoga: ' + str(len(category\_df.loc[category\_df['new'] == 'yoga', :]))) # 340  
  
# manmom  
# category\_df.loc[category\_df['Activity'] == 'video exercise workouts, TV conditioning programs', 'new'] = 'manmom'  
# category\_df.loc[category\_df['Activity'] == 'resistance training', 'new'] = 'resistance'  
# category\_df.loc[category\_df['Activity'] == 'circuit training', 'new'] = 'resistance'  
# category\_df.loc[category\_df['Activity'] == 'aerobic, step', 'new'] = 'aerobic'  
  
# category\_df.loc[category\_df['Activity'] == 'calisthenics', 'new'] = 'yoga'  
# category\_df.loc[category\_df['Activity'] == 'home exercise, general', 'new'] = 'home'  
# category\_df.loc[category\_df['Activity'] == 'slide board exercise, general', 'new'] = 'slide board'  
# category\_df.loc[category\_df['Activity'] == 'stretching', 'new'] = 'manmom'  
# category\_df.loc[category\_df['Activity'] == 'rope skipping, general', 'new'] = 'rope'  
# print('manmom: ' + str(len(category\_df.loc[category\_df['new'] == 'manmom', :]))) # 1024  
  
# rowing  
category\_df.loc[category\_df['Activity'] == 'rowing, stationary', 'new'] = 'rowing'  
print('rowing: ' + str(len(category\_df.loc[category\_df['new'] == 'rowing', :]))) # 150  
  
# skiing  
# ski\_list = getContainList(cate\_list, 'skiing')  
# category\_df.loc[((category\_df['Activity'].isin(ski\_list)) &  
# (category\_df['Category'] == 'winter activities')), 'new'] = 'skiing'  
# print('ski: ' + str(len(category\_df.loc[category\_df['new'] == 'skiing', :]))) # 355  
  
# running  
category\_df.loc[category\_df['Category'] == 'running', 'new'] = 'running'  
print('running: ' + str(len(category\_df.loc[category\_df['new'] == 'running', :]))) # 291  
  
# skateboarding  
# category\_df.loc[category\_df['Activity'] == 'skateboarding', 'new'] = 'skateboarding'  
# print('skateboarding: ' + str(len(category\_df.loc[category\_df['new'] == 'skateboarding', :]))) # 184  
  
# baseball  
# category\_df.loc[category\_df['Activity'] == 'softball, general', 'new'] = 'baseball'  
# print('baseball: ' + str(len(category\_df.loc[category\_df['new'] == 'baseball', :]))) # 173  
  
# soccer  
# category\_df.loc[category\_df['Activity'] == 'soccer', 'new'] = 'soccer'  
# print('soccer: ' + str(len(category\_df.loc[category\_df['new'] == 'soccer', :]))) # 137  
  
# golf  
category\_df.loc[category\_df['Activity'] == 'golf', 'new'] = 'golf'  
print('golf: ' + str(len(category\_df.loc[category\_df['new'] == 'golf', :]))) # 138  
  
# basketball  
# category\_df.loc[category\_df['Activity'] == 'basketball', 'new'] = 'basketball'  
# category\_df.loc[category\_df['Activity'] == 'basketball, game (Taylor Code 490)', 'new'] = 'basketball'  
# print('basketball: ' + str(len(category\_df.loc[category\_df['new'] == 'basketball', :]))) # 170  
  
category\_df.dropna(axis=0, inplace=True)  
X = category\_df.drop(columns=['ID', 'NAME', 'Scale', 'Activity', 'Category', 'new'])  
  
# category\_df.dropna(axis=0, inplace=True)  
# category\_df['nose\_X'] = (category\_df['upper neck\_X'] + category\_df['head top\_X'])/2  
# category\_df['nose\_Y'] = (category\_df['upper neck\_Y'] + category\_df['head top\_Y'])/2  
# X = category\_df.drop(columns=['ID', 'NAME', 'Scale', 'Activity', 'Category', 'new',  
# 'pelvis\_X','pelvis\_Y', 'thorax\_X', 'thorax\_Y',  
# 'upper neck\_X', 'upper neck\_Y', 'head top\_X', 'head top\_Y'])  
  
y = category\_df['new']  
  
scaledX = pd.DataFrame(StandardScaler().fit\_transform(X.transpose()))  
# encode\_y = LabelEncoder().fit\_transform(y)  
  
sm = SMOTE()  
x\_resample, y\_resample = sm.fit\_resample(scaledX.transpose(), y)  
  
print(dataset.head())  
print(dataset.describe())  
print(dataset.info())  
  
  
  
  
# Decision Tree Entropy  
def dteClassifier(X\_train, Y\_train, X\_test, Y\_test):  
 dte = DecisionTreeClassifier(criterion="entropy")  
 dte.fit(X\_train, Y\_train)  
 print(dte.score(X\_test, Y\_test))  
  
  
# Decision Tree Gini  
def dtgClassifier(X\_train, Y\_train, X\_test, Y\_test):  
 dtg = DecisionTreeClassifier(criterion="gini")  
 dtg.fit(X\_train, Y\_train)  
 print(dtg.score(X\_test, Y\_test))  
  
  
# Logistic Regression  
def logisticRegr(X\_train, y\_train, X\_test, y\_test):  
 logisticRegr = LogisticRegression(solver='lbfgs')  
 logisticRegr.fit(X\_train, y\_train)  
 logisticRegr.predict(X\_train[0].reshape(1, -1))  
 logisticRegr.predict(X\_train[0:10])  
 predictions = logisticRegr.predict(y\_test)  
 score = logisticRegr.score(X\_test, y\_test)  
 print(score)  
  
  
# SVC  
def svc(X\_train, y\_train, X\_test, y\_test):  
 # create an SVC classifier model  
 svclassifier = SVC(kernel='linear')  
 # fit the model to train dataset  
 svclassifier.fit(X\_train, y\_train)  
 # make predictions using the trained model  
 y\_pred = svclassifier.predict(X\_test)  
 print(svclassifier.score(X\_test, y\_test))  
  
  
  
# evaluation each model  
def evaluation(x, y, classifier):  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x, y, test\_size=0.2, shuffle=True, random\_state=1)  
 skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=1)  
 score = cross\_val\_score(classifier, X\_train, Y\_train, cv=skf)  
 print(classifier, '\nCross validation score :', score)  
 classifier.fit(X\_train, Y\_train)  
 print('Accuracy on test set :', classifier.score(X\_test, Y\_test))  
 print('')  
  
  
listClassifier = [DecisionTreeClassifier(criterion="entropy"), DecisionTreeClassifier(criterion="gini"),  
 LogisticRegression(), SVC()]  
  
# listBestDf index 0=DecisionTree(entropy), 1=DecisionTree(gini) 2=LogisticRegression, 3=SVC  
  
for i in range(len(listClassifier)):  
 evaluation(x\_resample, y\_resample, listClassifier[i])  
  
# grid Decision Tree Entropy  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
param\_grid = [{'max\_features': np.arange(1, len(X\_test.columns)), 'max\_depth': np.arange(1, 20)}]  
dt\_entropy\_gscv = GridSearchCV(listClassifier[0], param\_grid, cv=2)  
dt\_entropy\_gscv.fit(X\_train, y\_train)  
print(dt\_entropy\_gscv.best\_params\_)  
print('Best score :', dt\_entropy\_gscv.best\_score\_)  
  
# grid Decision Tree Gini  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
param\_grid = [{'max\_features': np.arange(1, len(X\_test.columns)), 'max\_depth': np.arange(1, 10)}]  
dt\_gini\_gscv = GridSearchCV(listClassifier[1], param\_grid, cv=2, n\_jobs=2)  
dt\_gini\_gscv.fit(X\_train, y\_train)  
print(dt\_gini\_gscv.best\_params\_)  
print('Best score :', dt\_gini\_gscv.best\_score\_)  
  
# grid Logistic Regression  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
param\_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],  
 'penalty': ['l1', 'l2']}  
lr\_gscv = GridSearchCV(listClassifier[2], param\_grid, cv=2, n\_jobs=2)  
lr\_gscv.fit(X\_train, y\_train)  
print(lr\_gscv.best\_params\_)  
print('Best score :', lr\_gscv.best\_score\_)  
  
# grid SVC  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
param\_grid = {'C': [0.1, 1, 10, 100, 1000],  
 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],  
 'kernel': ['rbf', 'poly', 'sigmoid']}  
svc\_gscv = GridSearchCV(listClassifier[3], param\_grid, cv=2, n\_jobs=2)  
svc\_gscv.fit(X\_train, y\_train)  
print(svc\_gscv.best\_params\_)  
print('Best score :', svc\_gscv.best\_score\_)  
  
# print('\n---------After GridSearchCV---------\n')  
# dt\_e = DecisionTreeClassifier(max\_depth=10, max\_features=29, criterion="entropy")  
# dt\_g = DecisionTreeClassifier(max\_depth=9, max\_features=24, criterion="gini")  
# lr = LogisticRegression(C=100, penalty="l2")  
# svc = SVC(C=10, gamma=0.1, kernel='rbf')  
  
# evaluation(listBestDf[0], y, dt\_e)  
# evaluation(listBestDf[1], y, dt\_g)  
# evaluation(listBestDf[2], y, lr)  
# evaluation(listBestDf[3], y, svc)  
  
# visulization  
  
# Decision Tree(entropy)  
dte\_best = DecisionTreeClassifier(max\_depth=10, max\_features=29, criterion="entropy")  
dte\_best.fit(x\_resample, y\_resample)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
print(confusion\_matrix(Y\_test, dte\_best.predict(X\_test)))  
plt.figure(figsize=(2, 2))  
ax = sns.heatmap(metrics.confusion\_matrix(Y\_test, dte\_best.predict(X\_test)), annot=True, fmt='.2f', linewidths=.1, cmap='Blues')  
plt.title("Heatmap of Decision Tree(entropy)")  
plt.show()  
  
visualizer = ROCAUC(dte\_best, classes=['golf', 'rowing', 'running', 'yoga'], micro=False, macro=True, per\_class=False)  
visualizer.fit(x\_resample, y\_resample)  
visualizer.score(x\_resample, y\_resample)  
visualizer.show()  
  
print("classification\_report using Decision Tree(entropy)")  
print(classification\_report(Y\_test, dte\_best.predict(X\_test)))  
  
# Decision Tree(gini)  
dtg\_best = DecisionTreeClassifier(max\_depth=9, max\_features=24, criterion="gini")  
dtg\_best.fit(x\_resample, y\_resample)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
print(confusion\_matrix(Y\_test, dtg\_best.predict(X\_test)))  
plt.figure(figsize=(2, 2))  
ax = sns.heatmap(metrics.confusion\_matrix(Y\_test, dtg\_best.predict(X\_test)), annot=True, fmt='.2f', linewidths=.1, cmap='Blues')  
plt.title("Heatmap of Decision Tree(gini)")  
plt.show()  
  
visualizer = ROCAUC(dte\_best, classes=['golf', 'rowing', 'running', 'yoga'], micro=False, macro=True, per\_class=False)  
visualizer.fit(x\_resample, y\_resample)  
visualizer.score(x\_resample, y\_resample)  
visualizer.show()  
  
print("classification\_report using Decision Tree(gini)")  
print(classification\_report(Y\_test, dte\_best.predict(X\_test)))  
  
# LogisticRegression  
lr\_best = LogisticRegression(C=100, penalty="l2")  
lr\_best.fit(x\_resample, y\_resample)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
print(confusion\_matrix(Y\_test, lr\_best.predict(X\_test)))  
plt.figure(figsize=(2, 2))  
ax = sns.heatmap(metrics.confusion\_matrix(Y\_test, lr\_best.predict(X\_test)), annot=True, fmt='.2f', linewidths=.1, cmap='Blues')  
plt.title("Heatmap of Logistic Regression")  
plt.show()  
  
visualizer = ROCAUC(lr\_best, classes=['golf', 'rowing', 'running', 'yoga'], micro=False, macro=True, per\_class=False)  
visualizer.fit(x\_resample, y\_resample)  
visualizer.score(x\_resample, y\_resample)  
visualizer.show()  
  
print("classification\_report using Logistic Regression")  
print(classification\_report(Y\_test, lr\_best.predict(X\_test)))  
  
# SVM  
svc\_best = SVC(C=10, gamma=0.1, kernel='rbf')  
svc\_best.fit(x\_resample, y\_resample)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x\_resample, y\_resample, test\_size=0.2, shuffle=True, random\_state=1)  
print(confusion\_matrix(Y\_test, svc\_best.predict(X\_test)))  
plt.figure(figsize=(2, 2))  
ax = sns.heatmap(metrics.confusion\_matrix(Y\_test, svc\_best.predict(X\_test)), annot=True, fmt='.2f', linewidths=.1, cmap='Blues')  
plt.title("Heatmap of Support Vector Machine")  
plt.show()  
  
visualizer = ROCAUC(svc\_best, classes=['golf', 'rowing', 'running', 'yoga'], micro=False, macro=True, per\_class=False)  
visualizer.fit(x\_resample, y\_resample)  
visualizer.score(x\_resample, y\_resample)  
visualizer.show()  
  
print("classification\_report using Support Vector Machine")  
print(classification\_report(Y\_test, svc\_best.predict(X\_test)))