# SVM

import pandas as pd  
from sklearn import svm  
from sklearn.metrics import confusion\_matrix  
  
  
def getData(year):  
 df = pd.read\_csv('TMO\_weekly\_label.csv')  
 df = df[df['Year'] == year]  
 X = df[['Mean Return', 'Volatility']].values  
 Y = df['Label'].values  
 return X, Y  
  
  
def getTable(cm, i):  
 TP = cm[i][0][0]  
 FP = cm[i][0][1]  
 FN = cm[i][1][0]  
 TN = cm[i][1][1]  
 TPR = TP / (TP + FN)  
 TNR = TN / (TN + FP)  
 ACC = (TP + TN) / (TP + TN + FP + FN)  
 d = {'Accuracy': [ACC], 'True positive rate': [TPR], 'True negative rate': [TNR]}  
 dfx = pd.DataFrame(data=d)  
 return dfx  
  
  
def SVM():  
 x\_train, y\_train = getData(2021)  
 x\_test, y\_test = getData(2022)  
 cm, prediction = [], []  
  
 *# Task 1-5* for i in range(3):  
 if i == 0:  
 *# 1-3. implement a linear SVM.* svm\_classifier = svm.SVC(kernel='linear')  
 kern = 'linear'  
 elif i == 1:  
 *# 4. implement a Gaussian SVM* svm\_classifier = svm.SVC(kernel='rbf')  
 kern = 'Gaussian'  
 else:  
 *# 5. implement polynomial SVM degree 2* svm\_classifier = svm.SVC(kernel='poly', degree=2)  
 kern = 'polynomial'  
  
 svm\_classifier.fit(x\_train, y\_train)  
 predicted = svm\_classifier.predict(x\_test)  
 accuracy = svm\_classifier.score(x\_test, y\_test)  
 prediction.append(predicted)  
 cm.append(confusion\_matrix(y\_test, predicted))  
 dfx = getTable(cm, i)  
  
 print('\nTask', i+1)  
 print('Implement a', kern, 'SVM:')  
 print('The accuracy is', accuracy)  
 print('The confusion matrix is:')  
 print(cm[i])  
 print(dfx)  
  
 *# 6. implement a trading strategy based on your labels (from linear SVM) for year 2  
 # and compare the performance with the ”buy-and-hold” strategy.  
 # Which strategy results in a larger amount at the end of the year?* df2 = pd.read\_csv("TMO\_weekly\_label.csv")  
 df2 = df2[df2['Year'] == 2022]  
 *meanReturn* = df2['Mean Return']  
 print("\nTask 4:")  
 print('Money earned based on buy-and-hold strategy for Year2:')  
 print("-2.2672499999999984")  
  
 for i in range(3):  
 if i == 0:  
 kern = 'linear'  
 elif i == 1:  
 kern = 'Gaussian'  
 else:  
 kern = 'polynomial'  
  
 meanReturn = list(df2['Mean Return'])  
 moneyEarned = 0  
 for j in range(52):  
 if prediction[i][j] == 'g':  
 moneyEarned = moneyEarned + meanReturn[j]  
 print('\nNew strategy: only buy when the predicted label is green.')  
 print('Money earned based on', kern, 'SVM strategy for Year2:')  
 print(moneyEarned)  
  
 print('\nStrategy based on linear SVM has the largest amount at the end of the year.')  
  
  
SVM()

# NB, Trees & RF

import pandas as pd  
import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB  
from sklearn import tree  
from sklearn import svm  
import warnings  
warnings.filterwarnings('ignore')  
  
  
def getData():  
 *# Q1 data preparation* df = pd.read\_excel('cardiotocography\_data\_set.xls', sheet\_name="Raw Data")  
 df["NSP"] = np.where(df["NSP"] < 2, 1, 0)  
 X = df[["ASTV", "MLTV", "Max", "Median"]].values  
 Y = df[["NSP"]].values  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, train\_size=0.5)  
 return x\_train, x\_test, y\_train, y\_test  
  
  
def getTable(cm, i, all=False):  
 TP = cm[i][0][0]  
 FP = cm[i][0][1]  
 FN = cm[i][1][0]  
 TN = cm[i][1][1]  
 TPR = TP / (TP + FN)  
 TNR = TN / (TN + FP)  
 ACC = (TP + TN) / (TP + TN + FP + FN)  
 d = {'Accuracy': [ACC], 'True positive rate': [TPR], 'True negative rate': [TNR]}  
 dfx = pd.DataFrame(data=d)  
 if all:  
 return TP, FP, FN, TN, TPR, TNR, ACC  
 return dfx  
  
  
def Q2\_Q7():  
 x\_train, x\_test, y\_train, y\_test = getData()  
 result\_table = pd.DataFrame(columns=['Method', 'TP', 'FP', 'FN', 'TN', 'Accuracy', 'TPR', 'TNR'])  
 method = ['Naive Bayesian', 'Logistic Regression', 'Decision Tree', 'Random Forest',  
 'linear SVM', 'degree 2 SVM', 'Gaussian SVM']  
 cm = []  
  
 *# 2. Use Naive Bayesian NB classifier* NB\_classifier = GaussianNB().fit(x\_train, y\_train)  
 accuracy = accuracy\_score(y\_test, NB\_classifier.predict(x\_test))  
 cm.append(confusion\_matrix(y\_test, NB\_classifier.predict(x\_test)))  
 dfx = getTable(cm, 0)  
  
 print("\nQ2:")  
 print('Implement a Naive Bayesian classifier:')  
 print('The accuracy is', accuracy)  
 print(cm[0])  
 print(dfx)  
  
 *# 3. Use Logistic regression classifier* log\_reg\_classifier = LogisticRegression()  
 log\_reg\_classifier.fit(x\_train, y\_train)  
 prediction = log\_reg\_classifier.predict(x\_test)  
 accuracy = log\_reg\_classifier.score(x\_train, y\_train)  
 cm.append(confusion\_matrix(y\_test, prediction))  
 dfx = getTable(cm, 1)  
  
 print("\nQ3:")  
 print('Implement a Logistic regression classifier:')  
 print('The accuracy is', accuracy)  
 print(cm[1])  
 print(dfx)  
  
 *# 4. Use Decision Tree* clf = tree.DecisionTreeClassifier(criterion='entropy')  
 clf = clf.fit(x\_train, y\_train)  
 prediction = clf.predict(x\_test)  
 accuracy = accuracy\_score(y\_test, prediction)  
 cm.append(confusion\_matrix(y\_test, prediction))  
 dfx = getTable(cm, 2)  
  
 print("\nQ4:")  
 print('Implement a Decision Tree:')  
 print('The accuracy is', accuracy)  
 print('Confusion matrix:')  
 print(cm[2])  
 print(dfx)  
  
 *# 5. Use Random Forest classifier* error\_rate = []  
 random\_forest\_table = pd.DataFrame(columns=['n\_estimators', 'max\_depth', 'accuracy'])  
 for i in range(1, 11):  
 for j in range(1, 6):  
 rf = RandomForestClassifier(n\_estimators=i, max\_depth=j)  
 rf.fit(x\_train, y\_train)  
 error\_rate.append(1 - accuracy\_score(y\_test, rf.predict(x\_test)))  
 ACC = accuracy\_score(y\_test, rf.predict(x\_test))  
 random\_forest\_table.loc[len(random\_forest\_table.index)] = [i, j, ACC]  
  
 *# plot the error rate* plt.plot(range(1, 11), error\_rate[:10], label="max\_depth=1")  
 plt.plot(range(1, 11), error\_rate[10:20], label="max\_depth=2")  
 plt.plot(range(1, 11), error\_rate[20:30], label="max\_depth=3")  
 plt.plot(range(1, 11), error\_rate[30:40], label="max\_depth=4")  
 plt.plot(range(1, 11), error\_rate[40:50], label="max\_depth=5")  
 plt.legend()  
 plt.xlabel("n\_estimators")  
 plt.ylabel("error rate")  
 *#plt.show()* best\_n = error\_rate.index(min(error\_rate)) % 10 + 1  
 best\_max = error\_rate.index(min(error\_rate)) % 5 + 1  
  
 print("\nQ5:")  
 print('Implement a Random Forest classifier :')  
 print("The best n\_estimators and max\_depth are", best\_n, "and", best\_max)  
  
 rf = RandomForestClassifier(n\_estimators=best\_n, max\_depth=best\_max)  
 rf.fit(x\_train, y\_train)  
 cm.append(confusion\_matrix(y\_test, rf.predict(x\_test)))  
 dfx = getTable(cm, 3)  
 accuracy = accuracy\_score(y\_test, rf.predict(x\_test))  
  
 print('The accuracy is', accuracy)  
 print('Confusion matrix:')  
 print(cm[3])  
 print(dfx)  
  
 *# 6. Use SVM classifier (linear, poly degree 2 and Gaussian)* for i in range(4, 7):  
 if i == 4:  
 *# linear SVM.* svm\_classifier = svm.SVC(kernel='linear')  
 kern = 'linear'  
 elif i == 5:  
 *# Gaussian SVM* svm\_classifier = svm.SVC(kernel='rbf')  
 kern = 'Gaussian'  
 else:  
 *# SVM degree 2* svm\_classifier = svm.SVC(kernel='poly', degree=2)  
 kern = 'polynomial'  
  
 svm\_classifier.fit(x\_train, y\_train)  
 predicted = svm\_classifier.predict(x\_test)  
 accuracy = svm\_classifier.score(x\_test, y\_test)  
 cm.append(confusion\_matrix(y\_test, predicted))  
 dfx = getTable(cm, i)  
  
 print('\nQ6 - Task', i-3, ':')  
 print('Implement a', kern, 'SVM:')  
 print('The accuracy is', accuracy)  
 print('Confusion matrix is:')  
 print(cm[i])  
 print(dfx)  
  
 *# 7. Summarize your results for Naive Bayesian, decision tree and random forest in a  
 # table below and discuss your findings.* for i in range(7):  
 TP, FP, FN, TN, ACC, TPR, TNR = getTable(cm, i, True)  
 new = [method[i], TP, FP, FN, TN, ACC, TPR, TNR]  
 result\_table = result\_table.append(pd.Series(new, index=result\_table.columns[:len(new)]), ignore\_index=True)  
  
 print('\nQ7:')  
 print(result\_table)  
  
  
Q2\_Q7()