ML2021S_HW5

March 6, 2022

1 CE-40717: Machine Learning

1.1 HW5-Support Vector Machine

1.1.1 Please fill this part

1. Full Name: AmirPourmand
2. Student Number: 99210259

You are just allowded to change those parts that start with "TO DO". Please do not change other parts.

It is highly recommended to read each codeline carefully and try to understand what it exactly does. Best of luck and have fun!

[2]:
```
# You are not allowed to import other packages.
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.svm import SVC

import cvxopt
```

About the Data: Heart diseases, also known as Cardiovascular diseases (CVDs), are the first cause of death worldwide, taking an estimated 17.9 million lives each year which is about 32% of all deaths all over the world.

In the present HomeWork, we are going to implement Support Vector Machines (SVM) algorithm that determines which patient is in danger and which is not.

For this perpose, `Heart_Disease_Dataset.csv` file can be used that is attached to the HomeWork folder. Use `Dataset_Description.pdf` for more detail.

[3]:
```
df = pd.read_csv("./Heart_Disease_Dataset.csv")
df
```

```
[3]:       age  sex  chest pain type  resting bp s  cholesterol  \
     0      40    1                2           140          289
     1      49    0                3           160          180
     2      37    1                2           130          283
     3      48    0                4           138          214
     4      54    1                3           150          195
     …    …   …              …            …            …
     1185   45    1                1           110          264
     1186   68    1                4           144          193
     1187   57    1                4           130          131
     1188   57    0                2           130          236
     1189   38    1                3           138          175

           fasting blood sugar  resting ecg  max heart rate  exercise angina  \
     0                       0            0             172                0
     1                       0            0             156                0
     2                       0            1              98                0
     3                       0            0             108                1
     4                       0            0             122                0
     …                     …            …               …                …
     1185                    0            0             132                0
     1186                    1            0             141                0
     1187                    0            0             115                1
     1188                    0            2             174                0
     1189                    0            0             173                0

           oldpeak  ST slope  target
     0         0.0         1       0
     1         1.0         2       1
     2         0.0         1       0
     3         1.5         2       1
     4         0.0         1       0
     …       …       …       …
     1185      1.2         2       1
     1186      3.4         2       1
     1187      1.2         2       1
     1188      0.0         2       1
     1189      0.0         1       0

     [1190 rows x 12 columns]
```

1.1.2 Pre-Processing - (15pts)

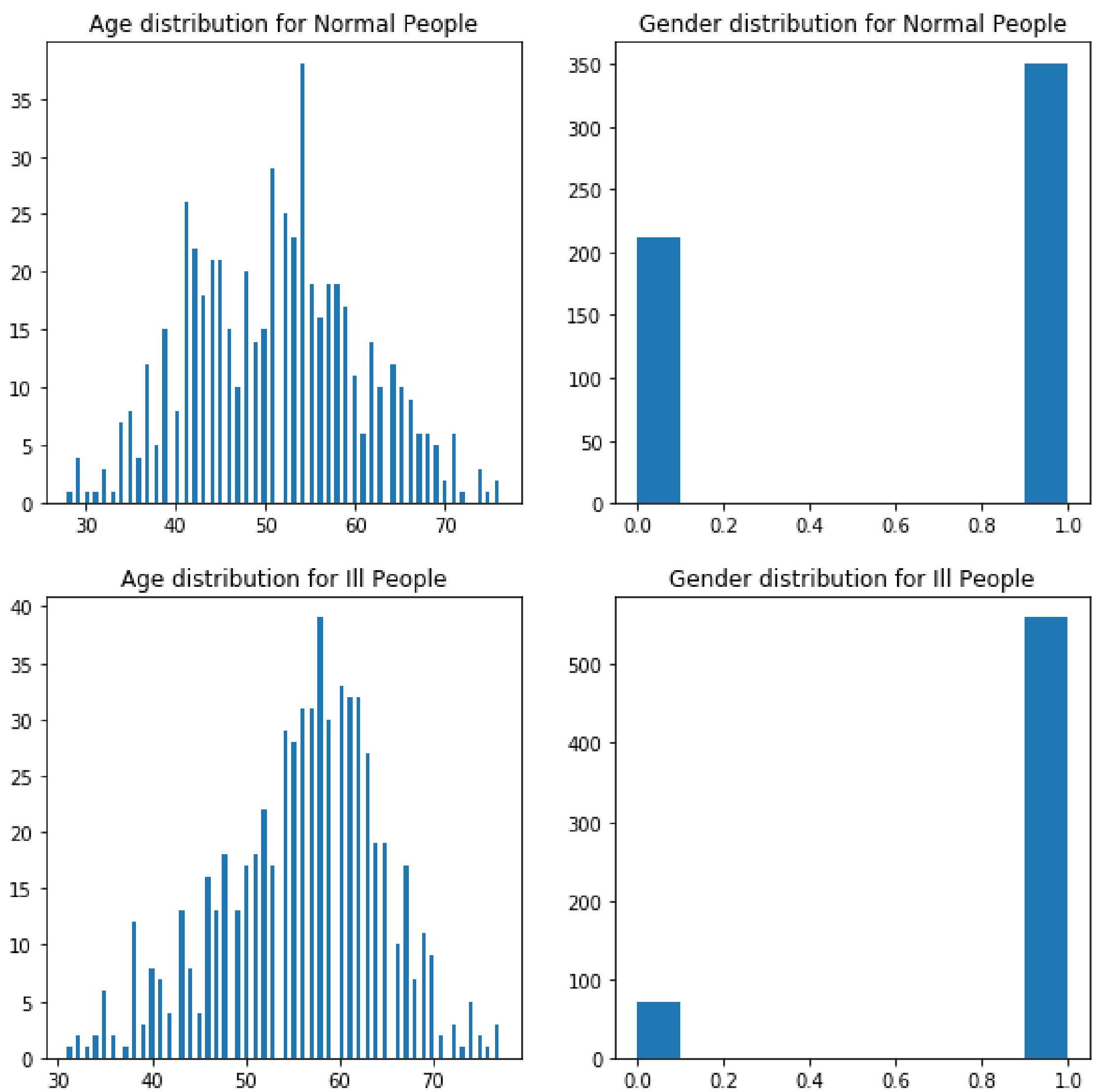
Exploratory Data Analysis (EDA): In statistics, exploratory data analysis is an approach to analyze datasets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

This is a general approach that should be applied when you encounter a dataset.

[4]:
```
###############################################################################
## TODO: Find the shape of the dataset.                                      ##
###############################################################################
shape = df.shape
print("shape of dataset is: " , shape)

###############################################################################
## TODO: Check if there is missing entries in the dataset columnwise.        ##
###############################################################################
missings = df.info()
print("this dataset doesn't have any missing value as shown above.")

###############################################################################
## TODO: Check whether the dataset is balanced or not.                       ##
## If the difference between 2 classes was less than 100 for our dataset,    ##
## it is called "ballanced".                                                 ##
###############################################################################
values=df.target.value_counts()
print("ballanced: ",np.abs(values[0]-values[1])<100)

###############################################################################
## TODO: plot the age distirbution and gender distrbution for both normal    ##
## and heart diseses patients.(4 plots)                                      ##
###############################################################################

print("--------------------- Plots -------------------------")

plt.figure(figsize=(10,10))

plt.subplot(2,2,1)
plt.hist(df[df.target==0].age,bins=100)
plt.title('Age distribution for Normal People')

plt.subplot(2,2,2)
plt.hist(df[df.target==0].sex)
plt.title('Gender distribution for Normal People')


plt.subplot(2,2,3)
plt.hist(df[df.target==1].age,bins=100)
plt.title('Age distribution for Ill People')

plt.subplot(2,2,4)
plt.hist(df[df.target==1].sex)
plt.title('Gender distribution for Ill People')
```

```
shape of dataset is:  (1190, 12)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
age                    1190 non-null int64
sex                    1190 non-null int64
chest pain type        1190 non-null int64
resting bp s           1190 non-null int64
cholesterol            1190 non-null int64
fasting blood sugar    1190 non-null int64
resting ecg            1190 non-null int64
max heart rate         1190 non-null int64
exercise angina        1190 non-null int64
oldpeak                1190 non-null float64
ST slope               1190 non-null int64
target                 1190 non-null int64
dtypes: float64(1), int64(11)
memory usage: 111.7 KB
this dataset doesn't have any missing value as shown above.
ballanced:  True
--------------------- Plots --------------------------
```

```
[4]: Text(0.5, 1.0, 'Gender distribution for Ill People')
```



Question 1: What do you conclude from the plots?

Answer:

I simply get the idea that ill people's age follows (roughly) from normal distribution. Also, the percentage of men who have heart attack is far higher than woman which means men should be more careful :)

Outlier Detection & Removal: We will filter ouliers using Z-test.

Z-test formula:

$$Z = \left|\frac{x - mu}{std}\right|$$

[5]:
```
###############################################################################
## TODO: Suppose that, based on our prior knowledge, we know some columns have##
## outliers. Calculate z-score for each featuer and determine the outliers    ##
## with threshold=3, then eliminate them. Target dataframe has(1173,12)shape. ##
###############################################################################
columns = ["age","resting bp s","cholesterol","max heart rate"]
threshold = 3

count = df.shape[0]
for col in columns:
    z_test_age = (df[col]-df[col].mean())/df[col].std()
    df=df[np.abs(z_test_age)<threshold]
df
###############################################################################
#                              END OF YOUR CODE                               #
###############################################################################
```

[5]:

	age	sex	chest pain type	resting bp s	cholesterol	\
0	40	1	2	140	289	
1	49	0	3	160	180	
2	37	1	2	130	283	
3	48	0	4	138	214	
4	54	1	3	150	195	
…	…	…	…	…	…	
1185	45	1	1	110	264	
1186	68	1	4	144	193	
1187	57	1	4	130	131	
1188	57	0	2	130	236	
1189	38	1	3	138	175	

	fasting blood sugar	resting ecg	max heart rate	exercise angina	\
0	0	0	172	0	
1	0	0	156	0	
2	0	1	98	0	
3	0	0	108	1	
4	0	0	122	0	
…	…	…	…	…	
1185	0	0	132	0	
1186	1	0	141	0	
1187	0	0	115	1	
1188	0	2	174	0	
1189	0	0	173	0	

```
      oldpeak  ST slope  target
0         0.0         1       0
1         1.0         2       1
2         0.0         1       0
3         1.5         2       1
4         0.0         1       0
…       …       …       …
1185      1.2         2       1
1186      3.4         2       1
1187      1.2         2       1
1188      0.0         2       1
1189      0.0         1       0

[1173 rows x 12 columns]
```

Feature Engineering: Sometimes the collected data are raw; they are either incompatible with your model or hinders its performance. That's when feature engineering comes to rescue. It encompasses preprocessing techniques to compile a dataset by extracting features from raw data.

[6]:
```
################################################################################
## TODO: Normalize numerical features to be between 0 and 1                  ##
## Note that just numerical fetures should be normalized. type of features is ##
## determined in dataset description file.                                    ##
################################################################################
def min_max_scaler(feature):
    return (feature-feature.min())/(feature.max()-feature.min())

numberical_columns = ['age','resting bp s','cholesterol','max heart␣
 ↪rate','oldpeak']
for col in numberical_columns:
    df[col] = min_max_scaler(df[col])

df
################################################################################
#                              END OF YOUR CODE                                #
################################################################################
```

[6]:
```
           age  sex  chest pain type  resting bp s  cholesterol  \
0     0.244898    1                2      0.571429     0.588595
1     0.428571    0                3      0.761905     0.366599
2     0.183673    1                2      0.476190     0.576375
3     0.408163    0                4      0.552381     0.435845
4     0.530612    1                3      0.666667     0.397149
…          …  …              …             …            …
1185  0.346939    1                1      0.285714     0.537678
1186  0.816327    1                4      0.609524     0.393075
```

```
1187  0.591837    1              4      0.476190      0.266802
1188  0.591837    0              2      0.476190      0.480652
1189  0.204082    1              3      0.552381      0.356415

      fasting blood sugar  resting ecg  max heart rate  exercise angina  \
0                       0            0        0.777778                0
1                       0            0        0.659259                0
2                       0            1        0.229630                0
3                       0            0        0.303704                1
4                       0            0        0.407407                0
…                     …           …             …               …
1185                    0            0        0.481481                0
1186                    1            0        0.548148                0
1187                    0            0        0.355556                1
1188                    0            2        0.792593                0
1189                    0            0        0.785185                0

       oldpeak  ST slope  target
0     0.295455         1       0
1     0.409091         2       1
2     0.295455         1       0
3     0.465909         2       1
4     0.295455         1       0
…         …       …      …
1185  0.431818         2       1
1186  0.681818         2       1
1187  0.431818         2       1
1188  0.295455         2       1
1189  0.295455         1       0

[1173 rows x 12 columns]
```

1.1.3 SVM - (25pts)

spliting data

[7]:
```python
# The original dataset labels is 0 and 1 and in the following code we change it
 ↪to -1 and 1.
df.target.replace(0 , -1 , inplace = True)

# Turn pandas dataframe to numpy array type
df = df.to_numpy()

# Splitting data into train and test part. 70% for train and 30% for test
train = df[:int(len(df) * 0.7)]
test = df[int(len(df) * 0.7):]
```

```
# Getting features
X_train = train[: , :-1]
y_train = train[: , -1]

# Getting labels
X_test = test[: , :-1]
y_test = test[: , -1]

# shapes should be:
# Train: (821, 11) (821,)
# Test: (352, 11) (352,)
print("Train: ", X_train.shape ,y_train.shape)
print("Test: " ,X_test.shape ,y_test.shape)
```

```
Train:  (821, 11) (821,)
Test:  (352, 11) (352,)
```

SVM Using sklearn: Use the standard libarary SVM classifier (SVC) on the training data, and then test the classifier on the test data. You will need to call SVM with 3 kernels: (1) Linear, (2) Polynomial and (3) RBF. You can change C to achive better results. For "RBF" find "gamma" witch takes 90% accuracy, at least. For polynomial kernel you are allowed to change "degree" to find best results.

For each kernel, reportting the followings is required: Accuracy, Precision, Recall, F1score.

[8]:
```
def classification_report(y_true, y_pred):
    ␣
 ↪##################################################################################
    ## TODO: Define a function that returns the followings:                        ␣
 ↪  ##
    ## Accuracy, Precision, Recall, F1score.                                       ␣
 ↪  ##
    ␣
 ↪##################################################################################
    True_Positive = np.sum((y_true==1)&(y_pred==1))
    False_Positive = np.sum((y_true==-1)&(y_pred==1))
    False_Negative = np.sum((y_true==1)&(y_pred==-1))
    True_Negative = np.sum((y_true==-1)&(y_pred==-1))


    Accuracy = np.mean(np.equal(y_true,y_pred))
    Precision = True_Positive/(True_Positive+False_Positive)
    Recall = True_Positive/(True_Positive+False_Negative)
    F1score = 2*Precision*Recall/(Precision+Recall)
    ␣
 ↪##################################################################################
```

```
    #                                     END OF YOUR CODE                    ␣
 ↪    #
  ␣
 ↪###########################################################################
    return f'{Accuracy:.3f}', f'{Precision:.3f}', f'{Recall:.3f}', f'{F1score:.
 ↪3f}'
```

[9]:

```
##########################################################################################
## TODO: Use svm of sklearn package (imported above) with 3 kernels.                     ␣
 ↪        ##
## You should define model, fit using X_train, predict using X_test.                    ␣
 ↪        ##
## your predictions known as y_pred.                                                     ␣
 ↪        ##
## then use classification_report function to evaluate model.                          ␣
 ↪        ##
##########################################################################################

linearClf=svm.SVC(kernel='linear',C=10)

linearClf.fit(X_train,y_train)
y_pred=linearClf.predict(X_test)
# linear kernel

print("results of sklearn svm linear kernel:", classification_report(y_test,␣
 ↪y_pred))


polyClf=svm.SVC(kernel='poly',C=1,degree=2)

polyClf.fit(X_train,y_train)
y_pred=polyClf.predict(X_test)
# polynomial kernel

print("results of sklearn svm polynomial kernel:",␣
 ↪classification_report(y_test, y_pred))


RBFClf=svm.SVC(kernel='rbf',C=50,gamma=20)

RBFClf.fit(X_train,y_train)
y_pred=RBFClf.predict(X_test)
# rbf kernel

print("results of sklearn svm RBF kernel:", classification_report(y_test,␣
 ↪y_pred))
```

```
###############################################################################
#                                        END OF YOUR CODE                     ␣
↪         #
###############################################################################
```

```
<IPython.core.display.Javascript object>

results of sklearn svm linear kernel: ('0.787', '0.779', '0.745', '0.762')

<IPython.core.display.Javascript object>

results of sklearn svm polynomial kernel: ('0.790', '0.796', '0.727', '0.760')

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

<IPython.core.display.Javascript object>

results of sklearn svm RBF kernel: ('0.932', '0.910', '0.944', '0.927')
```

SVM: Now that you know how the standard library SVM works on the dataset, attempt to implement your own version of SVM. Implement SVM using Quadratic Programming(QP) approach. Remember that SVM objective fuction with QP is:

$$min_\alpha \quad \frac{1}{2}\alpha^T\,Q\,\alpha - 1^T\,\alpha s.t. \qquad y^T\,\alpha = 0,\, \alpha \geq 0$$

where:

$$Q_{i,j} = y_i\,y_j\,\langle x_i\,,\,x_j\rangle$$

and:

$$\text{if } (\alpha_n > 0) \text{ then } x_n \text{ is a support vector}$$

For this perpose, complete the following code. You are allowed to use "cvxopt" package. It's an optimization package for Quadratic Programming. Below is the user's guide for the QP from CVXOPT:

Quadratic Programming

[17]:
```
# Hide cvxopt output
cvxopt.solvers.options["show_progress"] = False

###############################################################################
## TODO: Use the information from the lecture slides to formulate the SVM     ␣
↪   ##
```

```
## kernels. These kernel functions will be called in the SVM class.                      ␣
↪   ##
####################################################################################

def linear_kernel(x,z,gamma=None,poly=None):
    return np.dot(x,z)

def polynomial_kernel(x,z,gamma=None,polynomial=1):
    return (1 + np.dot(x,z)) ** polynomial

def rbf_kernel(x,z,gamma,poly=None):
    return np.exp(-gamma*np.linalg.norm(x-z)**2)

####################################################################################
#                                  END OF YOUR CODE                                ␣
↪    #
####################################################################################


class MySVM(object):
    def __init__(self, kernel=linear_kernel, C=None,gamma=None,degree=None):
        self.kernel = kernel
        self.C = C
        if self.C is not None: self.C = float(self.C)
        self.gamma = gamma
        self.degree = degree

    def fit(self, X, y):

        n_samples, n_features = X.shape

        ␣
↪####################################################################################
        ## TODO: Compute Gram matrix "K" for the given kernel.                        ␣
↪            ##
        ␣
↪####################################################################################␣
↪
        K = np.zeros(shape=(n_samples,n_samples))
        for i in np.arange(n_samples):
            for j in np.arange(n_samples):
                K[i,j] = self.kernel(X[i],X[j],self.gamma,self.degree)

        ␣
↪####################################################################################
```

```
        #                                                   END OF YOUR CODE                                                         ␣
↪               #
       ␣
↪#####################################################################################


       ␣
↪#####################################################################################
        ## TODO: Setup SVM objective function in QP form (Notation from␣
↪attached link).    ##
        ## Guidance: G and h have defferent definition if C is used or not.        ␣
↪             ##
       ␣
↪#####################################################################################

        P = cvxopt.matrix(y[:,None]*y[None,:]* K)
        q = cvxopt.matrix(-1*np.ones(n_samples))
        A = cvxopt.matrix(y, (1, n_samples))
        b = cvxopt.matrix(0.0)

        if self.C is None:
            G = cvxopt.matrix(-1*np.identity(n_samples) )
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            G = cvxopt.matrix(np.vstack((-1*np.identity(n_samples), np.
↪identity(n_samples))))
            h = cvxopt.matrix(np.hstack((np.zeros(n_samples), np.
↪ones(n_samples) * self.C)))


       ␣
↪#####################################################################################
        #                                                   END OF YOUR CODE                                                         ␣
↪               #
       ␣
↪#####################################################################################

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers
        alpha = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv = alpha > 1e-5
```

```
        #this will actually give the indices of the support vectors
        ind = np.arange(len(alpha))[sv]

        # get alphas of support vector , Xs and ys too.
        self.alpha = alpha[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]

        ␣
↪########################################################################################
        ## TODO: Compute the Intercept b and Weight vector w.                            ␣
↪           ##
        ␣
↪########################################################################################
        # Intercept
        self.b = 0
        diff = 0

        for n in range(len(self.alpha)):
            if self.C is not None and self.alphan[n] > self.C - 1e-5:
                continue
            diff =diff+ 1
            self.b =self.b+ self.sv_y[n]
            self.b =self.b- np.sum(self.alpha * self.sv_y * K[ind[n],sv])

        if diff>0:
            self.b=self.b/diff
        else:
            self.b = 0


        # Weight vector
        if self.kernel == linear_kernel:
            self.w = np.zeros(n_features)
            for n in range(len(self.alpha)):
                self.w = self.w + self.alpha[n] * self.sv_y[n] * self.sv[n]
        else:
            self.w = None #Guidance: for non-linear case this should be None.␣
↪(do not change)
        ␣
↪########################################################################################
        #                                   END OF YOUR CODE                             ␣
↪            #
        ␣
↪########################################################################################
```

```
    def predict(self, X):
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        else:
       ␣
 ↪#####################################################################################
        ## TODO: For non-linear case, implement the kernel trick to predict the␣
 ↪label.     ##
       ␣
 ↪#####################################################################################
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for alpha, sv_y, sv in zip(self.alpha, self.sv_y, self.sv):
                    s += alpha * sv_y * self.kernel(X[i], sv,self.gamma,self.
 ↪degree)
                y_predict[i] = s
            return y_predict + self.b


       ␣
 ↪#####################################################################################
        #                                   END OF YOUR CODE                    ␣
 ↪             #
       ␣
 ↪#####################################################################################
```

[13]:
```
#####################################################################################
## TODO: define 3 model same as previous part (SVM Using sklearn) and evaluate ␣
 ↪ ##
## them. Note that for comaparing your result with that part for each kernel␣
 ↪use ##
## same parameters in both parts.                                              ␣
 ↪ ##
#####################################################################################

# linear kernel
linearSVM = MySVM(linear_kernel,C=10)
linearSVM.fit(X_train,y_train)
y_pred=np.sign(linearSVM.predict(X_test))
print("results of MySVM linear kernel:", classification_report(y_test , y_pred))


# polynomial kernel
#best degree for polynomial is 1!
polySVM=MySVM(polynomial_kernel,C=1,degree=2)
```

```
polySVM.fit(X_train,y_train)
y_pred=np.sign(polySVM.predict(X_test))

print("results of sklearn svm polynomial kernel:",␣
↪classification_report(y_test, y_pred))

Rbf_SVM=MySVM(rbf_kernel,C=50,gamma=20)

Rbf_SVM.fit(X_train,y_train)
y_pred=np.sign(Rbf_SVM.predict(X_test))
# rbf kernel

print("results of sklearn svm RBF kernel:", classification_report(y_test,␣
↪y_pred))
```

```
results of MySVM linear kernel: ('0.787', '0.779', '0.745', '0.762')
results of sklearn svm polynomial kernel: ('0.793', '0.789', '0.745', '0.767')
results of sklearn svm RBF kernel: ('0.932', '0.910', '0.944', '0.927')
```

Question 2: Report best results.

1. Best kernel:
2. Best Accuracy:

Question 2: Report best results.

1. Best kernel: RBF Kernel
2. Best Accuracy: 93.18%

1.1.4 Bonus Score - (5pts)

In this step you can check other kernel functions or change parameters or any idea to get better result in compare with last section's results.

[14]:
```
df = pd.read_csv("./Heart_Disease_Dataset.csv")
columns = ["age","resting bp s","cholesterol","max heart rate"]
threshold = 3

count = df.shape[0]
for col in columns:
    z_test_age = (df[col]-df[col].mean())/df[col].std()
    df=df[np.abs(z_test_age)<threshold]

##here is my idea###
### we should scale all features. Not just numerical ones.
for col in df.columns:
```

```
    df[col] = min_max_scaler(df[col])

df.target.replace(0 , -1 , inplace = True)
df = df.to_numpy()
train = df[:int(len(df) * 0.7)]
test = df[int(len(df) * 0.7):]
X_train = train[: , :-1]
y_train = train[: , -1]
X_test = test[: , :-1]
y_test = test[: , -1]
Rbf_SVM=MySVM(rbf_kernel,C=50,gamma=20)
Rbf_SVM.fit(X_train,y_train)
y_pred=np.sign(Rbf_SVM.predict(X_test))
print("results of sklearn svm RBF kernel:", classification_report(y_test,␣
 ↪y_pred))
```

```
results of sklearn svm RBF kernel: ('0.938', '0.921', '0.944', '0.933')
```

My idea is simple: we should scale all features not just numerical ones.

you can see the overall performance of F1_score which is the best one we have is increased 1% from 0.92 to 0.93 percent and accuracy went from 93.2 to 93.8

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: