Programming Assignment 6: Support Vector Machines

Member: Luyao Wang, Di Jin, Yingqi Lin

Individual contributions:

(1) Luyao Wang:
Implementation of Support Vector Machines
The first part of the report

(2) Di Jin:
Implementation of Support Vector Machines
The second part of the report

(3) Yingqi Lin:
Implementation of Support Vector Machines
The third part of the report

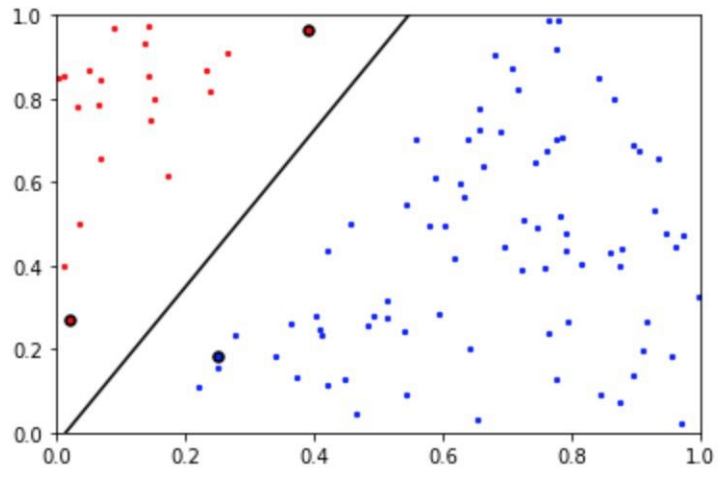
Part 1: Implementation

(1) Implementation of Support Vector Machines

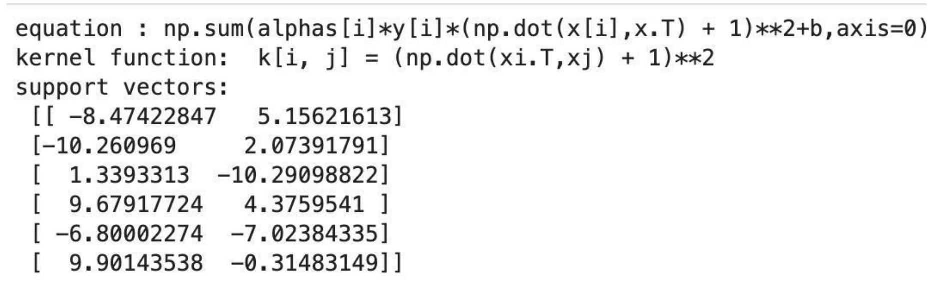
Output:

For the linear case, we get three support vectors, [0.24979414 0.18230306], [0.3917889 0.96675591], [0.02066458 0.27003158]. The three points are closest to the equation, y= 1.8773340822683058 x + -0.027703367686168996. The equation is used to separate the two groups of points.

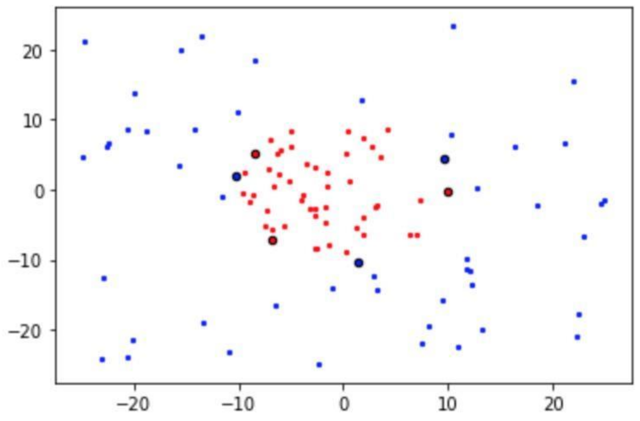
```
support vectors:
 [[0.24979414 0.18230306]
 [0.3917889  0.96675591]
 [0.02066458 0.27003158]]
equation is: y= 1.8773340822683058 x + -0.027703367686168996
```



For the nonlinear case, we get six support vectors, [-8.47422847   5.15621613], [-10.260969 2.07391791], [1.3393313 -10.29098822], [9.67917724   4.3759541], [-6.80002274 -7.02384335], [9.90143538 -0.31483149]. We use the polynomial kernel function. The equation of the curve is as follows.



```
equation : np.sum(alphas[i]*y[i]*(np.dot(x[i],x.T) + 1)**2+b,axis=0)
kernel function:  k[i, j] = (np.dot(xi.T,xj) + 1)**2
support vectors:
 [[ -8.47422847   5.15621613]
 [-10.260969     2.07391791]
 [  1.3393313  -10.29098822]
 [  9.67917724   4.3759541 ]
 [ -6.80002274  -7.02384335]
 [  9.90143538  -0.31483149]]
```

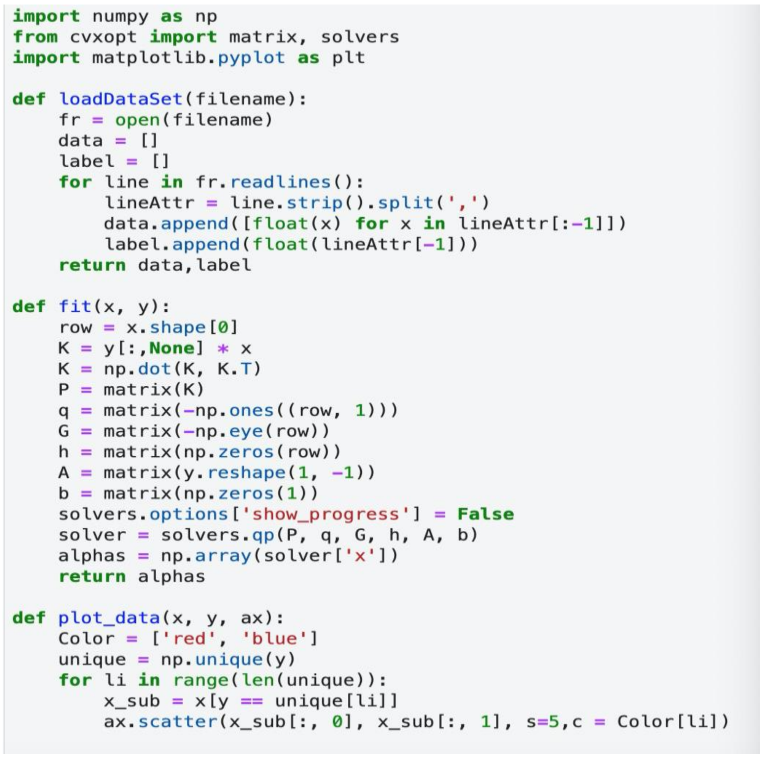


Data Structure:

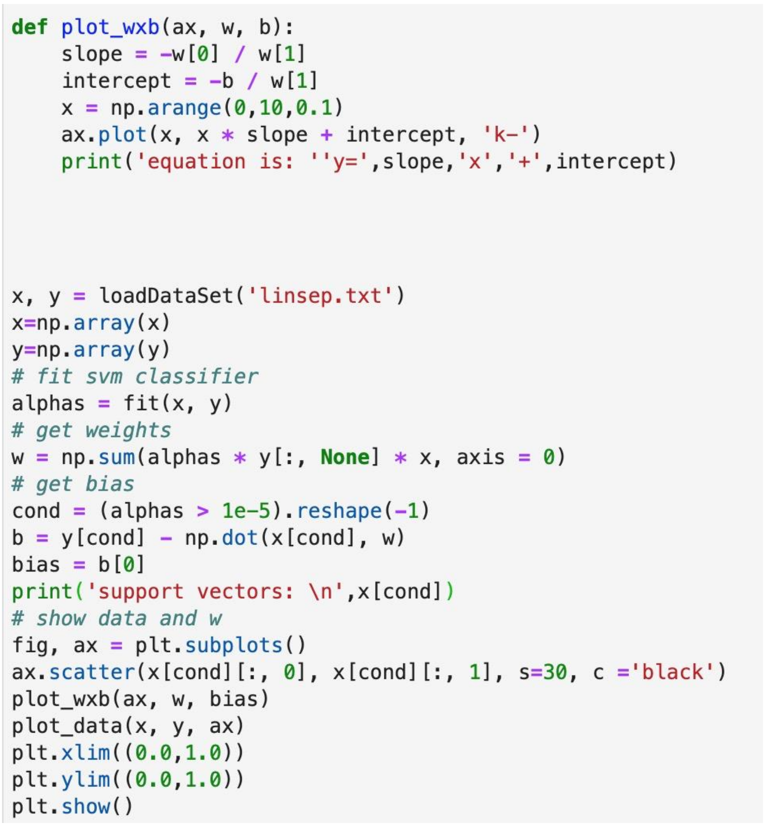
We use the two lists ‘data’ and ‘label’ to store the values of x and y because it is very convenient to store data from txt. Besides, we need to use the quadratic programming solver, so the list formal is also very convenient to be transferred to np.array formal. The np.array can be used in the quadratic programming solver.

Code-level Optimizations:

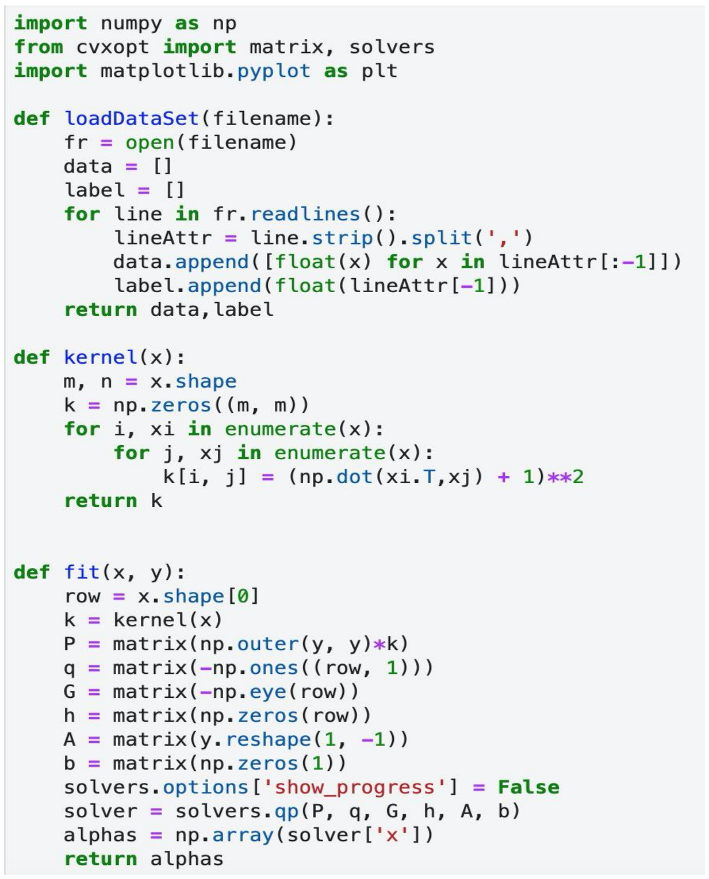
(1) We import cvxopt module’s fit function to calculate the alphas. It improves our efficiency because we will spend a lot of time and write a lot of codes to calculate the minimum value of the Lagrange formula.
(2) We import the matplotlib module to visualize points and support vectors. If there are some mistakes, we can directly observe through the pictures. At the same time, we also can observe our picture to judge whether we choose a proper kernel function.



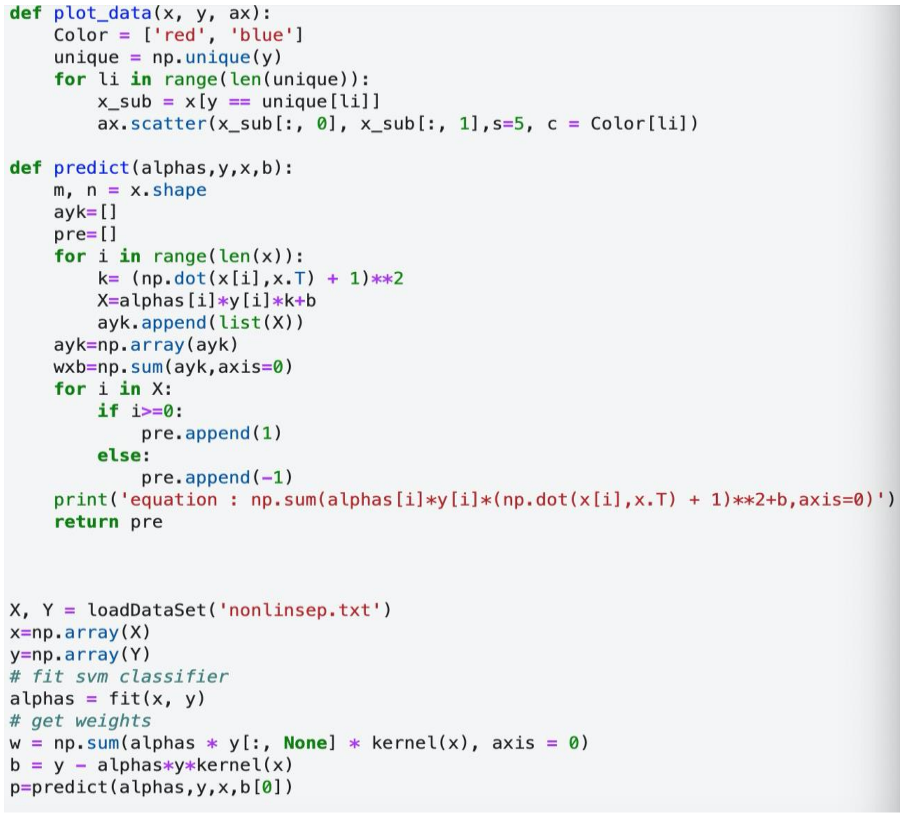
```python
import numpy as np
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt

def loadDataSet(filename):
    fr = open(filename)
    data = []
    label = []
    for line in fr.readlines():
        lineAttr = line.strip().split(',')
        data.append([float(x) for x in lineAttr[:-1]])
        label.append(float(lineAttr[-1]))
    return data,label

def fit(x, y):
    row = x.shape[0]
    K = y[:,None] * x
    K = np.dot(K, K.T)
    P = matrix(K)
    q = matrix(-np.ones((row, 1)))
    G = matrix(-np.eye(row))
    h = matrix(np.zeros(row))
    A = matrix(y.reshape(1, -1))
    b = matrix(np.zeros(1))
    solvers.options['show_progress'] = False
    solver = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solver['x'])
    return alphas

def plot_data(x, y, ax):
    Color = ['red', 'blue']
    unique = np.unique(y)
    for li in range(len(unique)):
        x_sub = x[y == unique[li]]
        ax.scatter(x_sub[:, 0], x_sub[:, 1], s=5,c = Color[li])
```



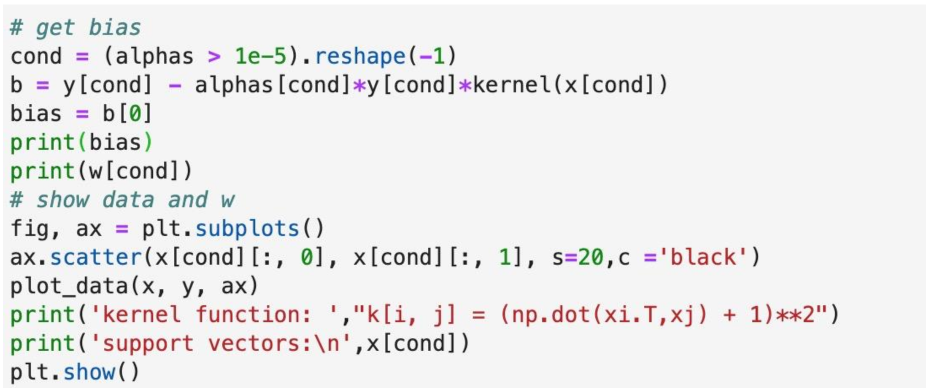
```python
def plot_wxb(ax, w, b):
    slope = -w[0] / w[1]
    intercept = -b / w[1]
    x = np.arange(0,10,0.1)
    ax.plot(x, x * slope + intercept, 'k-')
    print('equation is: ''y=',slope,'x','+',intercept)




x, y = loadDataSet('linsep.txt')
x=np.array(x)
y=np.array(y)
# fit svm classifier
alphas = fit(x, y)
# get weights
w = np.sum(alphas * y[:, None] * x, axis = 0)
# get bias
cond = (alphas > 1e-5).reshape(-1)
b = y[cond] - np.dot(x[cond], w)
bias = b[0]
print('support vectors: \n',x[cond])
# show data and w
fig, ax = plt.subplots()
ax.scatter(x[cond][:, 0], x[cond][:, 1], s=30, c ='black')
plot_wxb(ax, w, bias)
plot_data(x, y, ax)
plt.xlim((0.0,1.0))
plt.ylim((0.0,1.0))
plt.show()
```



```python
import numpy as np
from cvxopt import matrix, solvers
import matplotlib.pyplot as plt

def loadDataSet(filename):
    fr = open(filename)
    data = []
    label = []
    for line in fr.readlines():
        lineAttr = line.strip().split(',')
        data.append([float(x) for x in lineAttr[:-1]])
        label.append(float(lineAttr[-1]))
    return data,label

def kernel(x):
    m, n = x.shape
    k = np.zeros((m, m))
    for i, xi in enumerate(x):
        for j, xj in enumerate(x):
            k[i, j] = (np.dot(xi.T,xj) + 1)**2
    return k


def fit(x, y):
    row = x.shape[0]
    k = kernel(x)
    P = matrix(np.outer(y, y)*k)
    q = matrix(-np.ones((row, 1)))
    G = matrix(-np.eye(row))
    h = matrix(np.zeros(row))
    A = matrix(y.reshape(1, -1))
    b = matrix(np.zeros(1))
    solvers.options['show_progress'] = False
    solver = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solver['x'])
    return alphas
```



```python
def plot_data(x, y, ax):
    Color = ['red', 'blue']
    unique = np.unique(y)
    for li in range(len(unique)):
        x_sub = x[y == unique[li]]
        ax.scatter(x_sub[:, 0], x_sub[:, 1],s=5, c = Color[li])

def predict(alphas,y,x,b):
    m, n = x.shape
    ayk=[]
    pre=[]
    for i in range(len(x)):
        k= (np.dot(x[i],x.T) + 1)**2
        X=alphas[i]*y[i]*k+b
        ayk.append(list(X))
    ayk=np.array(ayk)
    wxb=np.sum(ayk,axis=0)
    for i in X:
        if i>=0:
            pre.append(1)
        else:
            pre.append(-1)
    print('equation : np.sum(alphas[i]*y[i]*(np.dot(x[i],x.T) + 1)**2+b,axis=0)')
    return pre



X, Y = loadDataSet('nonlinsep.txt')
x=np.array(X)
y=np.array(Y)
# fit svm classifier
alphas = fit(x, y)
# get weights
w = np.sum(alphas * y[:, None] * kernel(x), axis = 0)
b = y - alphas*y*kernel(x)
p=predict(alphas,y,x,b[0])
```



```
# get bias
cond = (alphas > 1e-5).reshape(-1)
b = y[cond] - alphas[cond]*y[cond]*kernel(x[cond])
bias = b[0]
print(bias)
print(w[cond])
# show data and w
fig, ax = plt.subplots()
ax.scatter(x[cond][:, 0], x[cond][:, 1], s=20,c ='black')
plot_data(x, y, ax)
print('kernel function: ',"k[i, j] = (np.dot(xi.T,xj) + 1)**2")
print('support vectors:\n',x[cond])
plt.show()
```

Challenges:

During the assignment, we meet with several challenges. Firstly, we don't know how to use the quadratic programming solver because it has lots of parameters, such as p, q, g, h, a, b. Therefore, we spent a lot of time researching the quadratic programming solver. Secondly, it is difficult to find the support vectors. When we get the alphas' values by QPP, we should judge whether these alphas' values are zero or not. Those points who have nonzero alphas should be the support vectors. However, all alphas are nonzero and we need to set a threshold to choose the support vectors. How to choose a proper threshold becomes a problem. In order to solve the problem, we set 1e-5 as the threshold and use the sklearn packet to check our results. Thirdly, for the nonlinear case, we can not draw the curve on the picture and it will be solved in the future.

Part 2: Software Familiarization

(1)How to use

In Python Scikit learn library, there are some very powerful and useful methods provided. One of them is SVM.SVC(). It can calculate support vectors with linear data and different kernel functions, like rbf and polynomials. Firstly, you need to create an object of SVM. And then, call the function fit() to calculate with your training data, you can also set your own parameters in this function, like the degree of polynomial and C. Finally, you can use any method and attributes that you need, such as predict(), support_vectors_ and so on.

Code:

```
from sklearn import svm
clf = svm.SVC(kernel='poly',degree=2, C=500') #poly, rbf, linear
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

(2)Compare and improve

Compared with our own codes, sklearn uses classes to pack SVM, and add different functions as its methods, also set a lot of common attributes. This idea makes SVM more flexible and easier to use. Besides that, their return values are easier to apply in plot functions. Thus, visualization will be easier for programmers.

Part 3: Applications

The applications of support vector machines are widespread which range from image classification to face detection, recognition of handwriting and even to bioinformatics. The support vector machine algorithms make use of training data to distribute multiple documents and flies into different categories. For the application of face detection, SVMs could classify images into two categories which are face or non-face. Then we can point to face to do some research. For the application of bioinformatics, which includes protein classification and cancer classification, can identify the genes and solve other biological problems.