INF 552 HW6

Teammates:
He Chang 5670527576
Ziqiao Gao 2157371827
Fanlin Qin 5317973858

Part 1.

Part (a).

Data Structure：

We choose the *cvxopt* package as our Quadratic Programming Problem Solver.

To begin with, we convert the data point into two np.array(), one is the Xs and the other is the label, Ys.

Then we follow the class note and the user guide of *cvxopt* to convert our data to the standard Quadratic Programming Problem input. Based on the user guide of *cvxopt,* there are 6 parameters required.

- $P := H$ a matrix of size $m \times m$
- $q := -\vec{1}$ a vector of size $m \times 1$
- $G := -diag[1]$ a diagonal matrix of -1s of size $m \times m$
- $h := \vec{0}$ a vector of zeros of size $m \times 1$
- $A := y$ the label vector of size $m \times 1$
- $b := 0$ a scalar

After using the QPP solver, we filter out those alphas that are greater than 0.0001, and then we use the target alphas index to find the support vectors.

Optimization:

1. We store the data point coordinates in np.array to leverage its fast work of matrix computation.
2. We plot the classifying line to visually examine if the results make sense.

Challenge:

1. Learning how to use the *cvxopt* Quadratic Programming Problem Solver library takes a lot of effort and we spend a long time converting our SVM QPP problem into the standard form that Quadratic Programming Problem Solver library requires.
2. Fixing the bugs that are caused by matrix multiplication is time consuming.

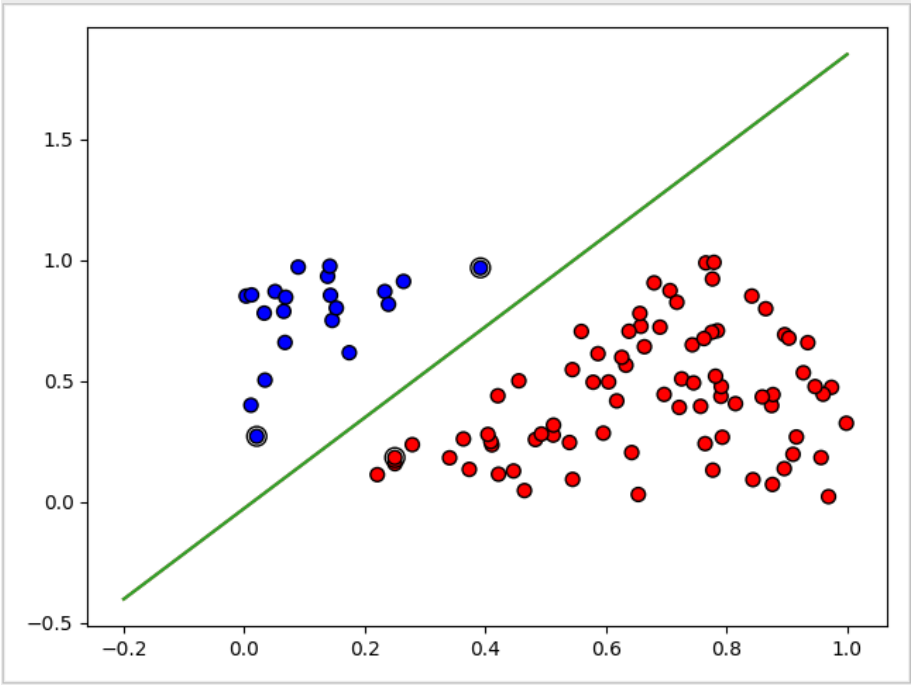
Result:

3 support vectors:
alphas:
[[33.73875192]
 [1.29468506]
 [32.4440672]]

X-Support Vector:
[[0.24979414 0.18230306]
 [0.3917889 0.96675591]
 [0.02066458 0.27003158]]

Equation: W.T * X + B = 0
W - weights:
[7.25005616 -3.86188932]
B - bias:
[-0.10698729]

Plotted classification results: support vectors are marked with an extra circle



Part (b).

Data Structure :

We use the same *cvxopt* package as our Quadratic Programming Problem Solver to handle the non-linear separable points.

We use the same data structure and algorithm as the part (a), except that we apply a new kernel function in this part to make this data separable on a higher dimension space. The kernel function is $func(x1, x2) = (1 + x1 \times x2)^2$. The trick here is that we do not directly convert the data points to a high-dimensional space but return the multiplication result of the high-dimensional data which is more efficient and more compliant with the class note.

After getting the QPP result, we then can plot the result of the classification curve using the new support vectors. To plot the curve, we create a meshgrid that covers all data points, and then we apply points on the meshgrid to the nonlinear SVM to predict the classification result --- Z. With Z we can plot the classification curve on the meshgrid.

Optimization:

1. We store the data point coordinates in np.array to leverage its fast work of matrix computation.
2. We plot the curve to visually examine if the classification results make sense.

Challenge:

1. Configuring the QPP standard variables with the kernel function increases the complexity of correctly producing the results.
2. Picking a feasible kernel function requires a lot of consideration and trail.
3. Studying to plot the classification curves takes us some time.

Result:

Kernel function: $func(x1, x2) = (1 + x1 \times x2)^2$

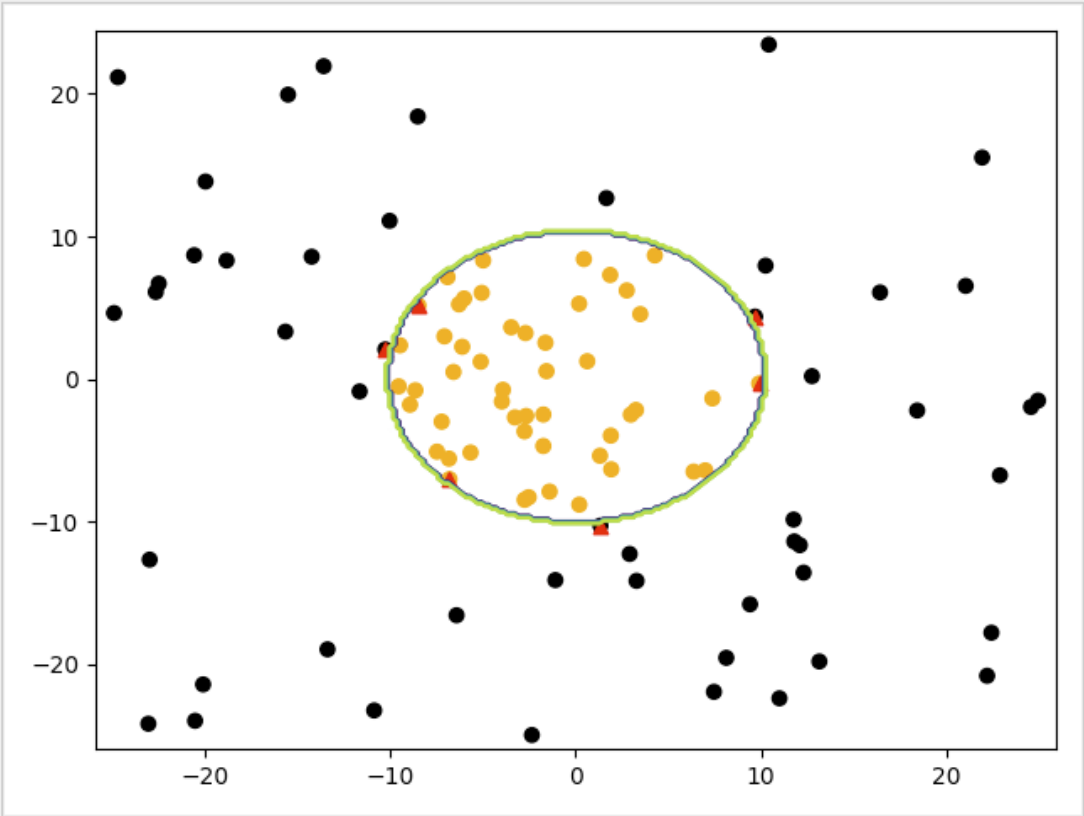
6 support vectors
alphas
[[0.0125012]
 [0.0149904]
 [0.00664213]
 [0.00466215]

```
 [0.00740837]
 [0.00638511]]

X - Support Vector:
[[ -8.47422847   5.15621613]
 [-10.260969     2.07391791]
 [  1.3393313  -10.29098822]
 [  9.67917724   4.3759541 ]
 [ -6.80002274  -7.02384335]
 [  9.90143538  -0.31483149]]

bias:
[-16.66005031]
```

Plotted classification results: support vectors are marked with triangles.



Part 2.

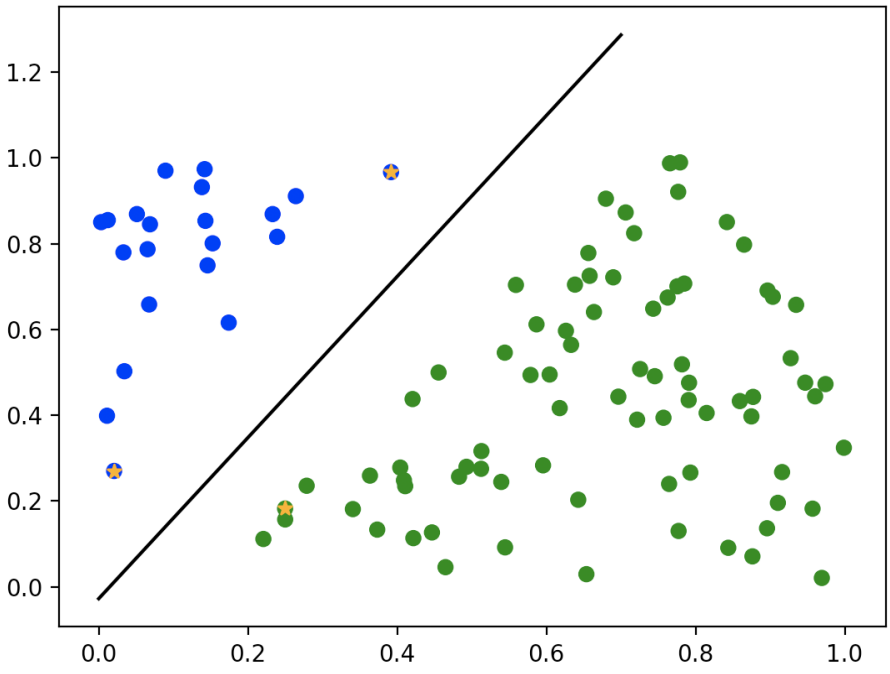
Library used: sklearn.svm import SVC

Result:

Linear:

Support Vectors:
[[0.3917889 0.96675591]
 [0.02066458 0.27003158]
 [0.24979414 0.18230306]]

Plotted classification results: support vectors are marked with stars.



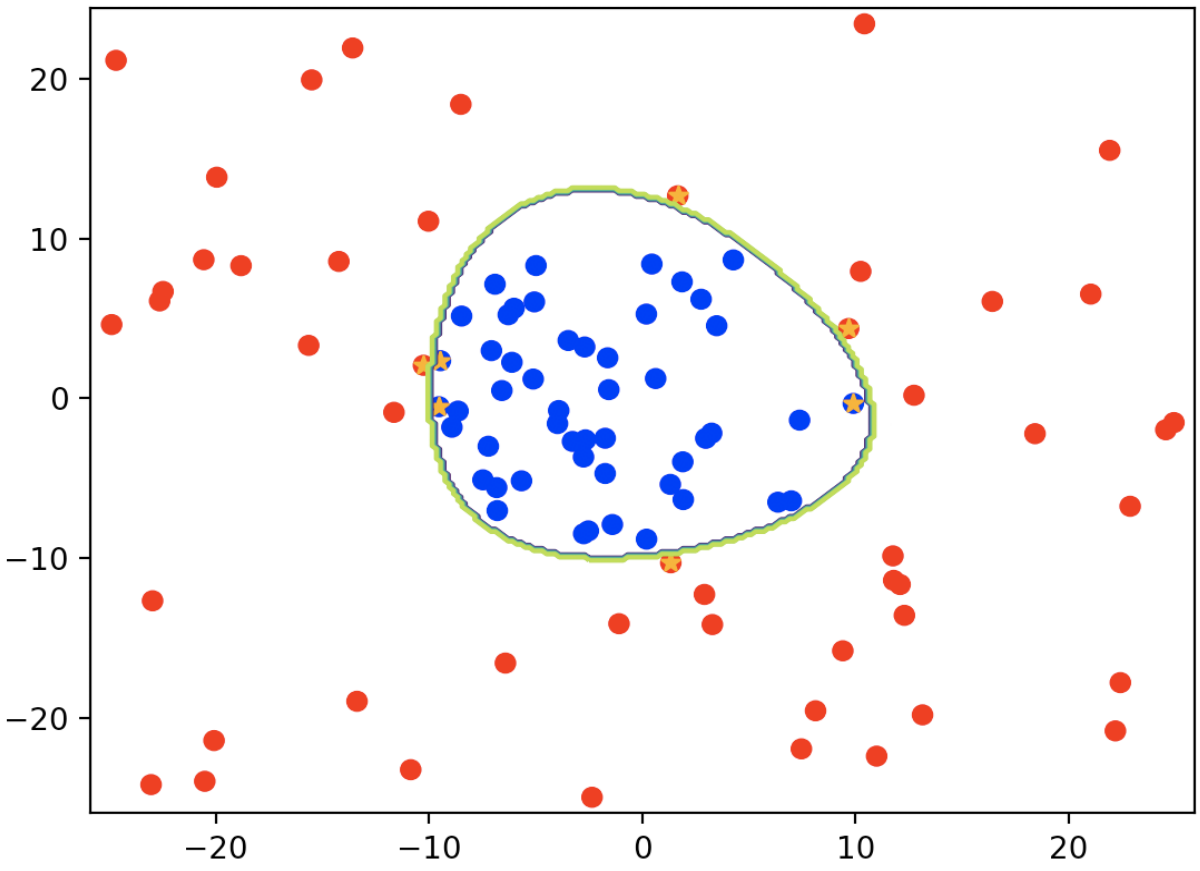
Non-Linear:

Kernel function chosen: Radial basis function

Support Vectors:
[[-9.53754332 -0.51895777]
 [-9.46760885   2.36139525]
 [9.90143538 -0.31483149]
 [-10.260969   2.07391791]
 [1.66404809 12.68562818]
 [1.3393313 -10.29098822]

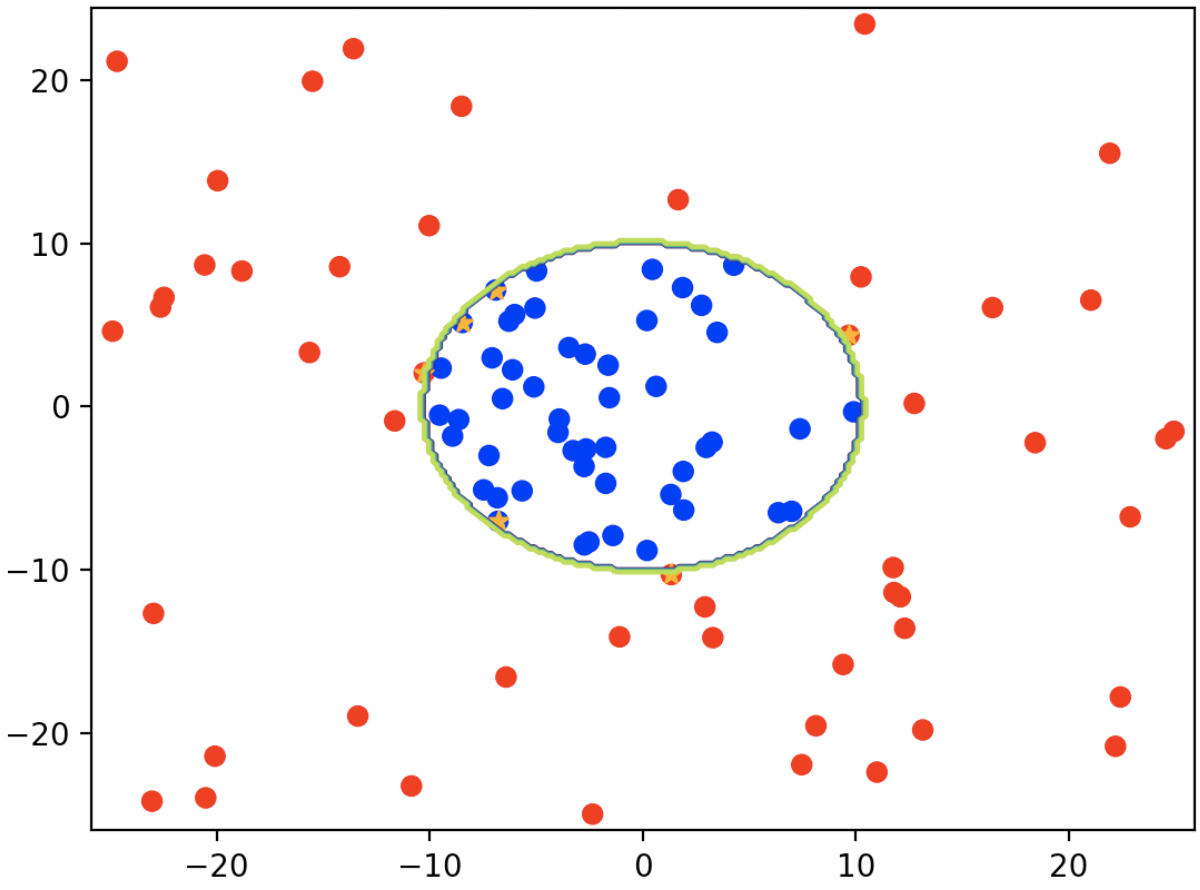
[9.67917724 4.3759541]]

Plotted classification results: support vectors are marked with stars.



Kernel function chosen: **Polynomial Kernel function with degree 2**
Support Vectors:
[[-8.47422847 5.15621613]
[-6.90647562 7.14833849]
[-6.80002274 -7.02384335]
[-10.260969 2.07391791]
[1.3393313 -10.29098822]
[9.67917724 4.3759541]]

Plotted classification results:



Summary:

Generally speaking, our part 1 results are very similar to the ones generated by sklearn SVM libraries. The sklearn.svm library is very powerful with various kinds of configuration combinations. We use orange stars to mark the support vectors on the map.

For the linear separable dataset, we just set the parameter kernel to be ‘linear’ and it works very well. For the non-linear separable dataset, we try 2 kernel functions to handle the classification, which are polynomial kernel function and radial basis kernel function. The two kernel functions both do a very good classification with similar classification results.

One thing that we learn from using sklearn.svm is that by adjusting parameter C, the penalty of misclassification, we can achieve a hard margin SVM by increasing its value. Also we find that having a higher degree for the polynomial kernel function will not guarantee a more accurate classification result.

Part 3. Application of SVM

With the development of traffic information acquisition systems, especially the application of video license plate acquisition systems, it is possible to obtain real-time and dynamic expressway travel time, and it also promotes the theoretical research and practical application requirements of rapid trip prediction. The SVM model selects the data of the first 4 periods of the prediction period as input feature values and uses genetic algorithms to establish the model parameter optimization algorithm to obtain the training model and its parameters. The accuracy of this model is higher.

[1] Juan Zhang and Jian Su, “ Research on SVM-based Urban Expressway Travel Time Prediction”. Information of Transportation, 2011

Individual contribution:
Ziqiao Gao: Optimize, implement and debug the SVM & Kernel SVM algorithm for part 1

He Chang: Optimize, implement and debug the SVM & Kernel SVM algorithm for part 1

Fanlin Qin: Research online and use the open source python library to implement SVM & Kernel SVM algorithm applications and complete part 2&3.