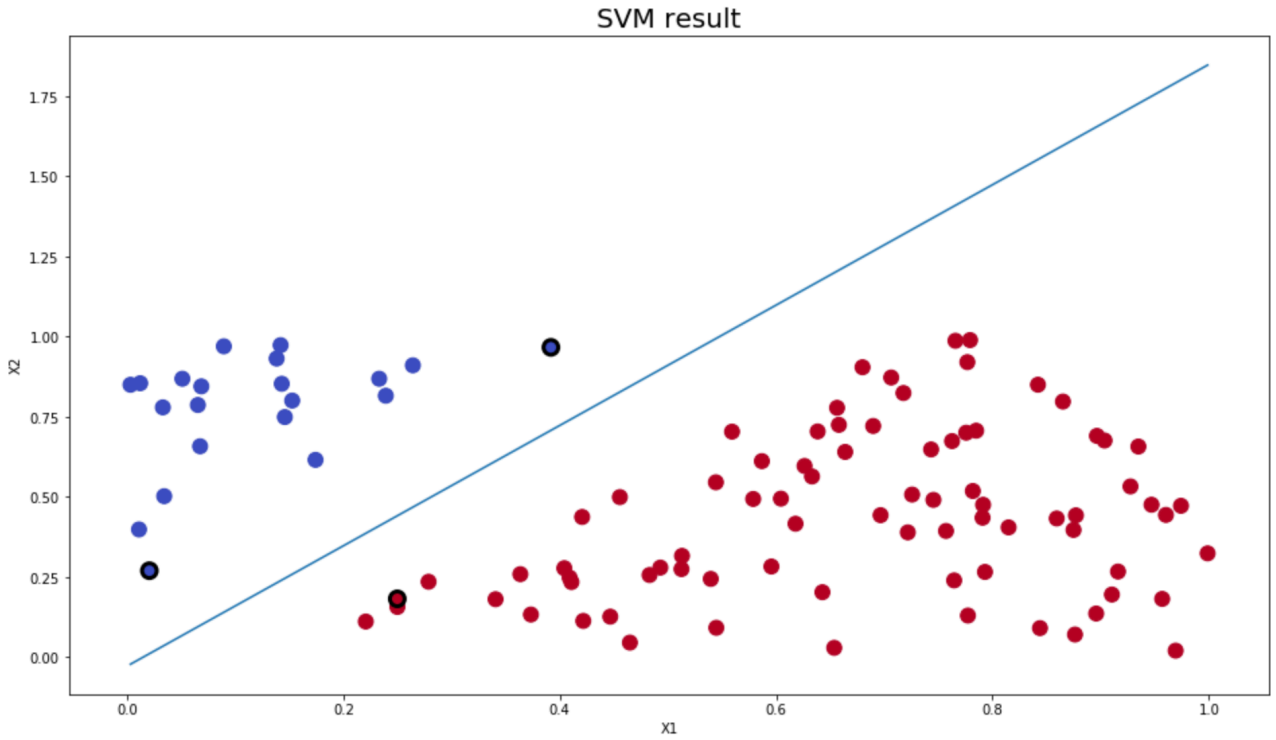
INF552: Programming Assignment 6

Student: Jiashi Chen USCID: 4684194123

Part 1: Implementation

Output of *Part (a)*:
Weight = [7.25005616, -3.86188932]. Intercept = -0.10698729
Equation of line: $y = 7.25005616 \times X_1 - 3.86188932 \times X_2 - 0.10698729$
Index of Support Vectors are [27, 83, 87]

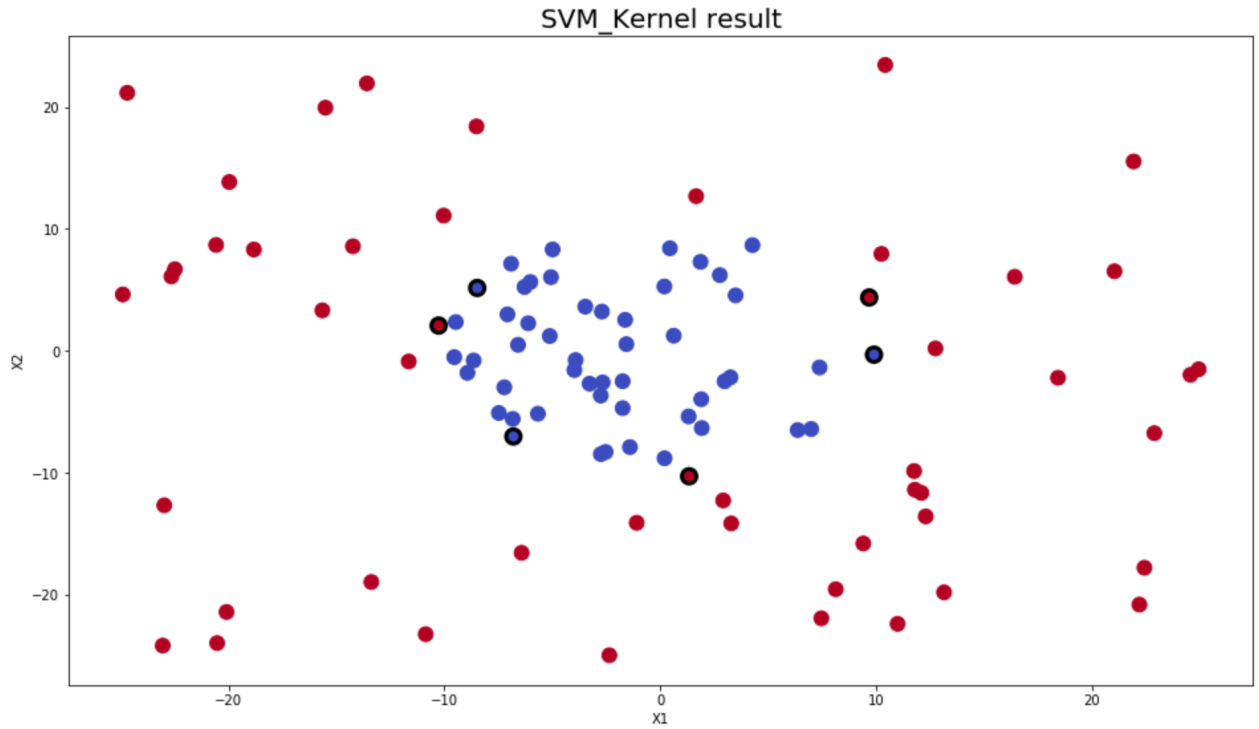


The nodes with black edge in the graph are support vectors.

Output of *Part (b)*:
Kernel function:

$$k(x_1, x_2) = (< x_1, x_2 > +1)^2$$

Weight = [8.64344220e-10, 1.60702131e-01, 1.58698040e-01, -9.47655776e-03, -3.85759106e-02, -1.10224094e-03]
Interception = -16.66005249
Equation of curve:
$y = 0.1607X_1^2 + 0.1587X_2^2 - 0.0095 \times \sqrt{2}X_1 - 0.0386 \times \sqrt{2}X_2 - 0.0011 \times \sqrt{2}X_1X_2 - 16.66$
Index of Support Vectors are [5, 36, 51, 55, 94, 95]



The nodes with black edge in the graph are support vectors.

The **data structure** I use is *matrix* from numpy module and *dataframe* from pandas module. Dataframe could help me store the orginal data. I use the matrix to store the 2D coordinates of a point, classification label and weight, which makes it easier for me to do matrix operations. Also I use *matrix* from cvxopt module because it is required when I need to use Quadratic Programming solver in that package. Only in QPP function, I use this data structure. Except the matrix I use in the QPP function, all other matrixs are from numpy module.

Any code-level **optimizations** I perform is that I try to use some vector and matrix operations to get the values at once when acculating variable *w* after getting alpha. This will be much faster than calculating the coordinates one when the dimensions of coordinates is large. What's more, I use the built-in function to get the alphas which are greater than 0 to make the code tidy. I also encapsulate each step of calculation into a function, which is good for my maintenance and finding bug.

One of the **challenges** I face is that I met a tricky problem when I tried to use the qp solver in cvxopt module to compute the Lagrange Multipliers. The console told me there was a *TypeError*. It made me think the dimension of one of my matrix was not matchable because I was new to selenium. However, I have checked so many ways which all turns out that the matrix has the right number of columns. I was stucked at this problem for a whole afternoon. And finally, I found out the reason why it keeped going wong is that I didn't change the type of element of the matrix to the double.

Part 2: Software Familiarization

SVM in Sklearn package offers a good implementation of the Support Vector Machines algorithms.

How to use

I used this package to fit the data in ***Jiashi_Chen_Homework_6_Code.ipynb*** file too.

API for Support Vector Machines algorithms(https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)

```
from sklearn.svm import SVC  ## Obtain necessary package

clf_linear = SVC(kernel="linear")  #Get linear classifier.
clf_linear.fit(X, y)  #Fit the model

clf_nonlinear = SVC(kernel='poly', degree = 2)  #Use polynomial Kernel function, and set degree
clf_nonlinear.fit(X, y)  #Fit the model
```

Beacuse the module will preprocess the data first, the output of algorithm will be diffrent.

Comparsion and Improvement

After wathcing the document of this module, I find the developer have considered many aspects of data processing to make the algorithm have a better performance. And I could try to learn and implement some of the method to improve my own code. There is a argument called *C* which controls the slackness of algorithm so that the algorithm allows some outliers exist in the dataset. It is exactly what I have learned from lecture. The penalty is a squared l2 penalty in this module which I could also add function into my code. Beyond what I have learned, the module also use one-vs-the-rest to classify the multi-label dataset which is really useful in dealing with many real life problem. And it is an upgrade based on one-against-one that I have already done.

Part 3: Applications

Support Vector Machines in recognizing human actions[1]

Local space-time features capture local events in video and can be adapted to the size, the frequency and the velocity of moving patterns. In this paper, authors demonstrate how such features can be used for recognizing complex motion patterns. They construct video representations in terms of local space-time features and integrate such representations with SVM classification schemes for recognition. For the purpose of evaluation they introduce a new video database containing 2391 sequences of six human actions performed by 25 people in four different scenarios. The presented results of action recognition justify the proposed method and demonstrate its advantage compared to other relative approaches for action recognition.

Support Vector Machines in face pose discrimination[2]

This paper describes an approach for the problem of face pose discrimination using support vector machines (SVM). Face pose discrimination means that one can label the face image as one of several known poses. Face images are drawn from the standard FERET database. The training set consists of 150 images equally distributed among frontal, approximately 33.75/spl deg/ rotated left and right poses, respectively, and the test set consists of 450 images again equally distributed among the three different types of poses. SVM achieved perfect accuracy-100%-discriminating between the three possible face poses on unseen test data, using either polynomials of degree 3 or radial basis functions (RBF) as kernel approximation functions.

References

[1] C. Schuldt, I. Laptev, B. Caputo. Recognizing human actions: a local SVM approach[C]// Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. IEEE, 2004.

[2] Huang J , Shao X , Wechsler H . Face Pose Discrimination Using Support Vector Machines (SVM) [C]// Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170). IEEE, 2002.