Assignment 6 Group Report - SVM

Group Members:
Che-Pai Kung 5999-9612-95
Chenqi Liu 2082-6026-02
Mengyu Zhang 3364-2309-80

Part 1: Implementation

Data structure

We use the same model to process both linear and non-linear data classification. Firstly, the data was read into the program by two functions– read_data() and read_data_nonlin(). They are basically the same. We use two functions just for easier calling in different functions later. After reading in, data was saved in numpy arrays. Data points coordinates are saved in X_train. Classification tags are saved in Y_train.

Since we want to use the quadratic programming function of python library CVXOPT.solver.pq, we manipulate the data further using fit() method in the class SVM. The function is shown below:

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x^\top P x + q^\top x \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b \end{aligned}$$

Parameters P, q, a, b are needed for the CVXOPT function. They are calculated using the raw data by function CVXOPT.matrix(). These parameters are all in matrix format. Weight vector and support vectors are all calculated in this method.

Then, we called test_linear() and test_non_linear() to execute reading data, running model and printing the results. Note that these two functions have different kernel, one applies linear kernel and another applies polynomial kernel.

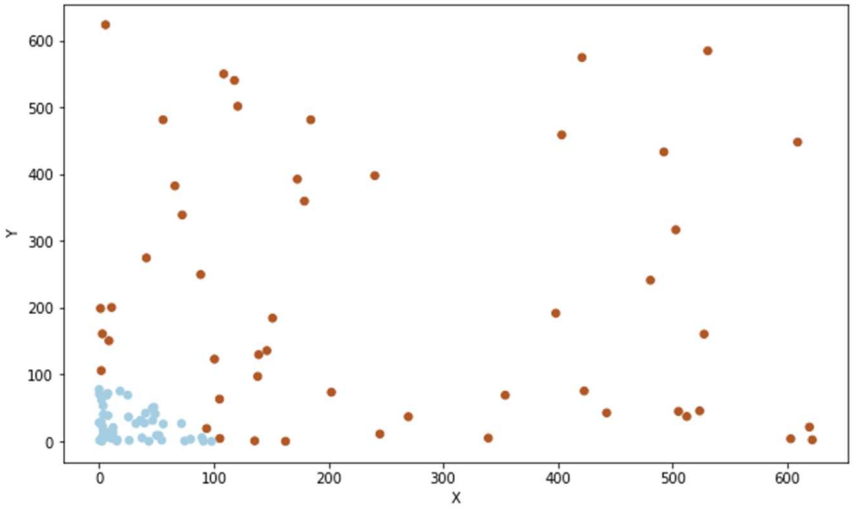
Code-level optimizations

We utilized matrix computation to reduce code length and increase its readability. Then to increase the structure of the program, the svm algorithm program was reorganized into object-oriented style by building all the functions, including plot data points with the hyperplane predicted, in SVM class.

Challenges

The biggest challenge we confronted is at the beginning of the implementation. It was uneasy to generate all the parameters for the CVXOPT function as we didn't understand it fully. Therefore, we read both lecture materials and online tutorial, then we were able to assign right value or matrix for each parameter (P, q, G, h, A, b). After that, the results of linear data were produced and are similar to that of using scikit learn library.

Once we successfully implement the quadratic programming for linear separable data, part b with non-linear separable becomes easier. All we need is just adding an additional kernel function to the program we have. After plotting the data points, we decided to use polynomial kernel with degree two to train the model as the data becomes linearly separable after squaring each vector. The result is presented below.



Another challenge we met was plotting the separate contour so that the image could be easily seen what the target value and predicted value are. Since we weren't familiar with plotting color of region, we did lots search on it and finally managed to utilize "pcolormesh" function in matplotlib.plt to draw the desired images.

Results

The results are presented below.

Linear("linsep.txt"):

We found 3 support vectors with the coefficient of [7.25005616 -3.86188932] and intercept of -0.10698726795885889

So, the equation can be written as:

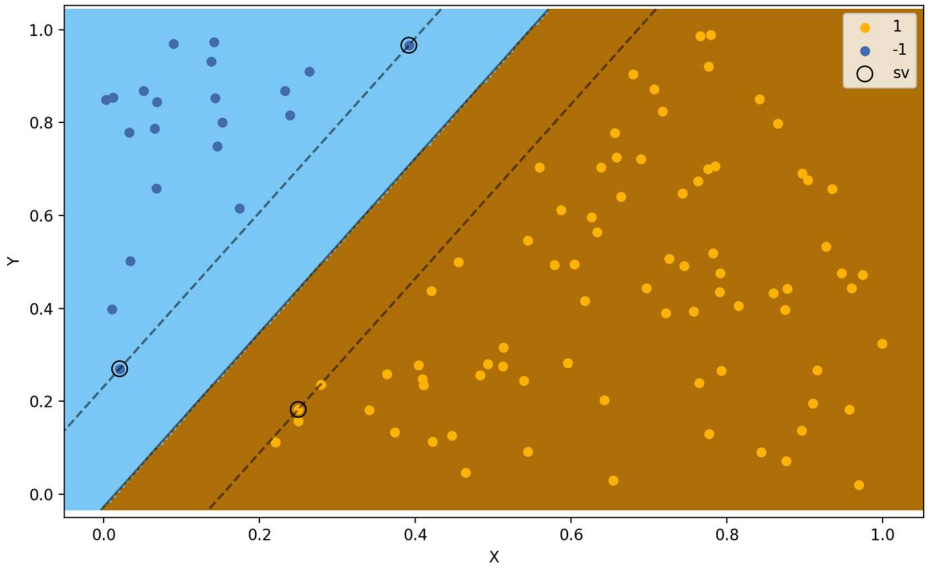
[7.25005616 -3.86188932]X - 0.10698726795885889 = 0

The support vectors are:

[0.24979414 0.18230306]
[0.3917889 0.96675591]
[0.02066458 0.27003158]

We trained the model with the whole dataset and got the accuracy rate, 100%.

From the plot, we can see that there are two support vectors on the negative side and one support vector on the positive side. Although, three points seems all very close to the margin. The program is still able to find the closest one.



Nonlinear("nonlinsep.txt"):

The kernel function we used:

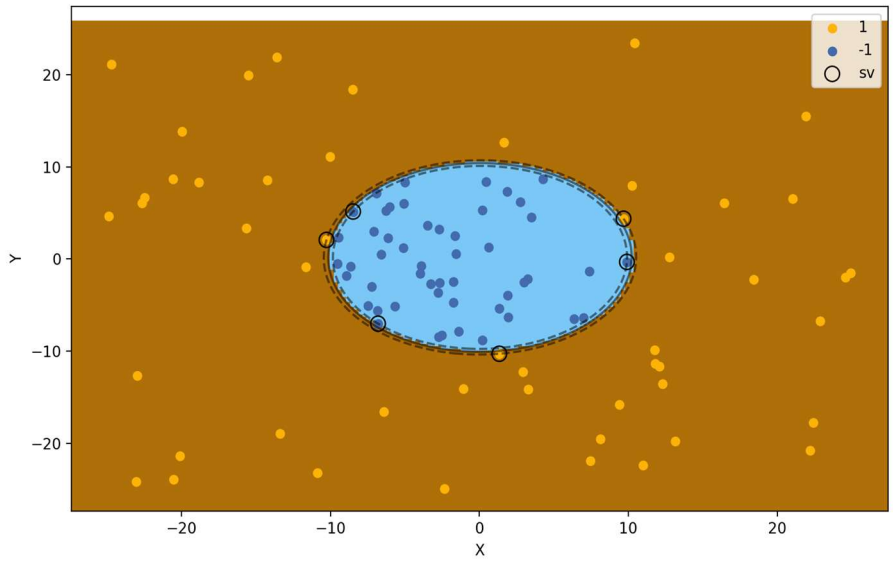
```
def polynomial_kernel(x, y, p=2):
    return (1 + np.dot(x, y)) ** p
```

We found six support vectors here in nonlinear separation with threshold of $1e^{-4}$.

The support vectors are:

[-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 trained the model with the whole dataset and got the accuracy rate, 100%.



Part 2: Software Familiarization

Implementation

The library we used this time is scikit-svm.SVC, which stands for Support Vector Classification.

In the following images, '- - - 'lines are the line whose distance to the optimal hyperplane is equal to one. Besides, each point is colored with its target value, while the color of the region means the prediction of the model.

Linearly separable file

We ran the model with a linear kernel and set C equal to 1000. The result is presented below, which is similar to what we got from our implementation.

Coefficient: [7.24837069, -3.86099178]

Intercept: [-0.10703977]

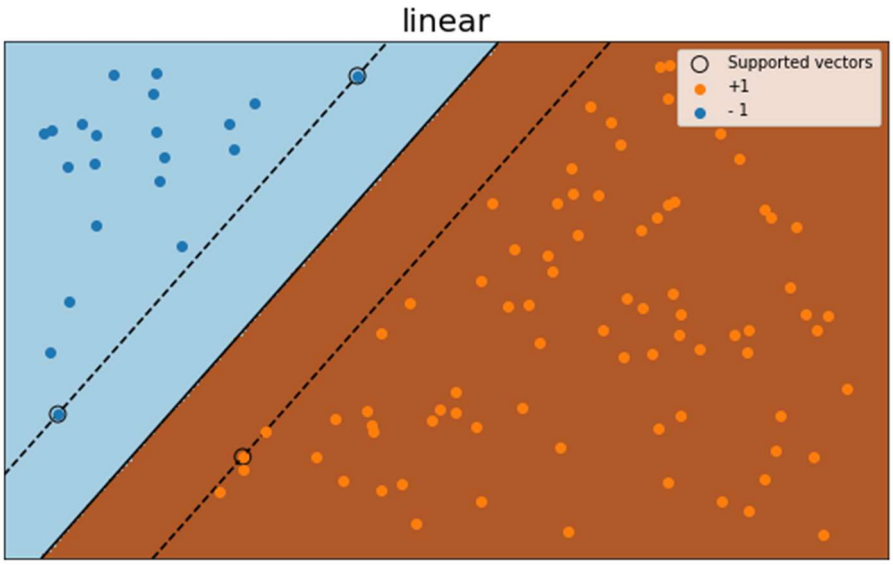
Support vectors:

[[0.3917889 ， 0.96675591],

[0.02066458， 0.27003158],

[0.24979414， 0.18230306]]

Accuracy rate: 100%



Nonlinearly separable file

We ran the model with a 'ploy' kernel with degree equal to 2 and set C equal to 1000. The result is presented below, which is also similar to what we got from our implementation.

Support vectors:

[[-8.47422847, 5.15621613],

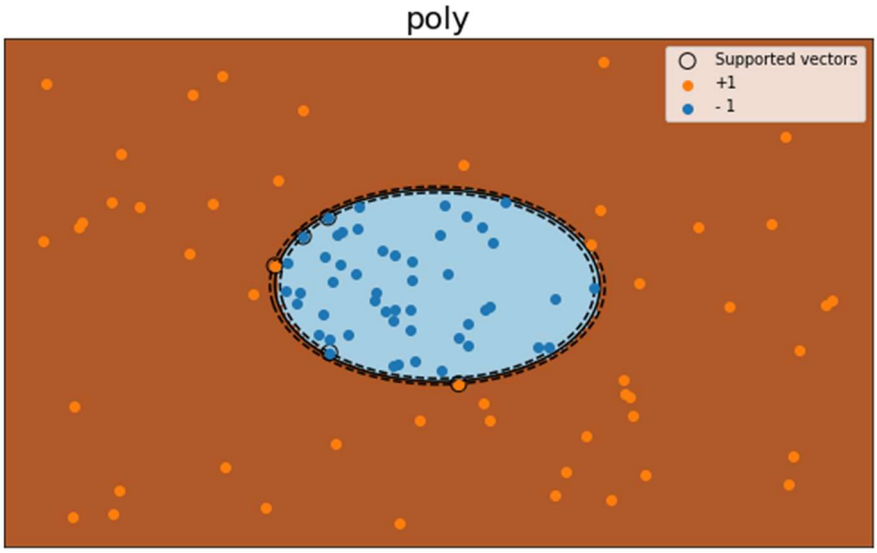
[-6.90647562, 7.14833849],

[-6.80002274, -7.02384335],

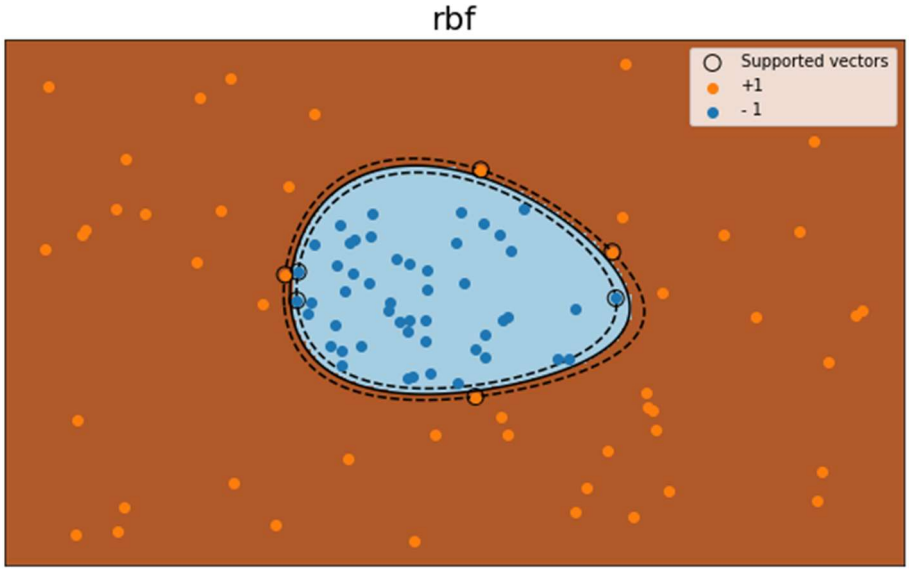
[-10.260969, 2.07391791],

[1.3393313, -10.29098822]]

Accuracy rate: 100%



We also ran the model with the 'rbf' kernel. It also classifies the dataset accurately.



Comparison and Inspiration

We think that the reason that the answers (hyperplane equation) from sklean library and our implementation are slightly different is due to the different approach to solve quadratic programming problems. We use Quadratic Programming solver to solve the formulations, while it seems that sklearn uses other iterative ways to solve it.

In addition, for the 'poly' kernel, there are other parameters, gamma and coef0, in sklearn used to adjust the coefficient of polynomial functions. If we set gamma equal to 1 and coef0 equal to 1, which is what we did in part1(b), then we got the same support vectors.

Regarding 'poly' kernel, sklearn library provides multiple parameters, such as gamma, so that the dataset with various features has a higher chance to be

separated well. Therefore, it may be helpful to add these features into our implementation to generate a more robust model.

Moreover, we can prevent the model from overfitting by setting parameter, C, in sklearn.svm.SVC. In this case, although some of the training data is misclassified, it avoids overfitting and may have a better outcome in testing data if there is any. Moreover, in reality most of the dataset cannot be separated totally by only a line. Hence, it is practical to have a soft margin rather than hard margin when training the model and our next step will be adding this parameter into our model.

Part 3: Applications

Support Vector Machine (SVM) is mainly applied on classification. It is an efficient algorithm, especially when training smaller dataset, so it is widely used in many fields.

One of the interesting applications we found is facial expression classification. With the help of SVM, it is possible to recognize whether a person is happy or sad based on the image or his/her face expression. This plays an important role when analyzing communication between humans.

Another impressive application is handwriting recognition. SVM is used to help recognize handwriting, and it is essential if we want to transfer paper documents into digital files.

Part 4: Individual Contributions

- Model discussion: Che-Pai Kung, Chenqi Liu, Mengyu Zhang
- Model implementation: Chenqi Liu, Mengyu Zhang
- Model optimization: Chenqi Liu, Mengyu Zhang
- Software Familiarization: Che-Pai Kung
- Applications: Che-Pai Kung