INF 552 Programming Assignment 6

Support Vector Machine

4472-6221-33 Yo Shuan Liu

Introduction

I complete this homework in Python3. My code can be executed from terminal, and there is no absolute path for input files. User only needs to follow below instructions to execute this script:

1) Put all the input .txt file needed (linsep.txt, nonlinsep.txt) and this code **under the same directory**.
2) Change directory to this folder and execute below line in terminal:
   python SVM.py
3) After printing out all the output, a message will show up in the terminal and ask user to **press [enter]** in order to end the execution and close all the output graph windows.

quadprog is used as the quadratic programming solver in this homework, please also pip install quadprog if it hasn't been installed. (reference: https://github.com/rmcgibbo/quadprog)

In this report, I will introduce my implementations of support vector machine using the linear kernel, the polynomial kernel and finally the gaussian kernel. I will cover following sessions:

I. Data Structure
II. Implementation, Result and Challenges I Face
III. Code Level Optimization
IV. Software Familiarization
V. Applications

I. Data Structure

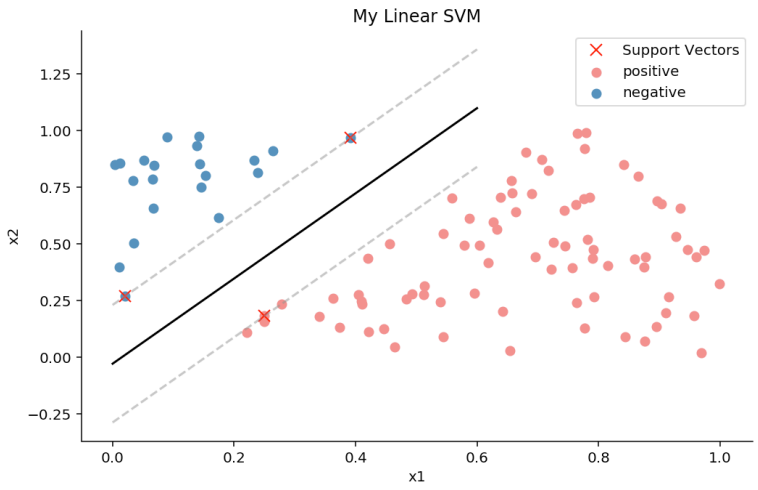
Since this homework heavily depends on matrix calculation, I take advantages of numpy matrix. I store training independent and dependent variables in numpy matrix. After finding out the support vectors, I store them in matrix as well as their Lagrange multiplier (if I choose to solve the dual problem).

II. Implementation, Result and Challenges I Face

- **Linearly Separable Data**

  Implementation

  I use linear SVM and solve its primal optimization problem to obtain the intercept and coefficients.



Result

There are three support vectors, below are their indices in input data and coordinates.
Support Vectors:

(1) 27th: [0.24979414, 0.18230306]
(2) 83th: [0.3917889 , 0.96675591]
(3) 87th: [0.02066458, 0.27003158]

Intercept: -0.10698734041931708
Coefficients: [7.2500562988083699, -3.8618892309583983]
Equation: x1 * 7.2500562988083699 + x2 * -3.8618892309583983 - 0.10698734041931708 = 0
(where x1 is the first independent variable and x2 is the second independent variable in input dataset.)

We can observe from the visualized result that we have only three support vectors. There are some points located on the boundary or close to the boundary, however, not all points on the boundary are support vectors, only points with Lagrange multiplier larger than zero could be seen as support vectors.

Challenges I Face: QP solver

There are some choices of QP solver for Python, I choose this quadprog finally. After choosing the quadratic programming solver, I have to read the instructions and the data type the solver accepts. I struggled for a while when the QP solver kept returning error because one of the matrix is not a positive definite matrix. I finally solved it by adding a small identity matrix on it.

◆ **Linearly Inseparable Data**

Implementation

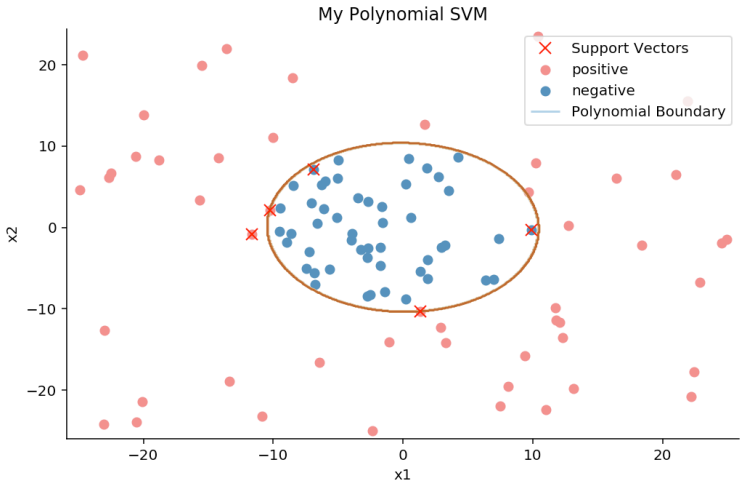
I tried both polynomial SVM and gaussian SVM and solve their dual optimization problem to obtain the decision boundary.

In conclusion, the gaussian SVM provides a more robust decision boundary but also higher computational cost than the polynomial SVM.

Result: Polynomial SVM

There are 5 support vectors which form the decision boundary of my polynomial SVM.
Support Vectors:

(1) 9th: [-11.64621294, -0.87217731]
(2) 36th: [-10.260969 2.07391791]
(3) 51th: [1.3393313 -10.29098822]
(4) 59th: [-6.90647562 7.14833849]
(5) 95th: [9.90143538 -0.31483149]



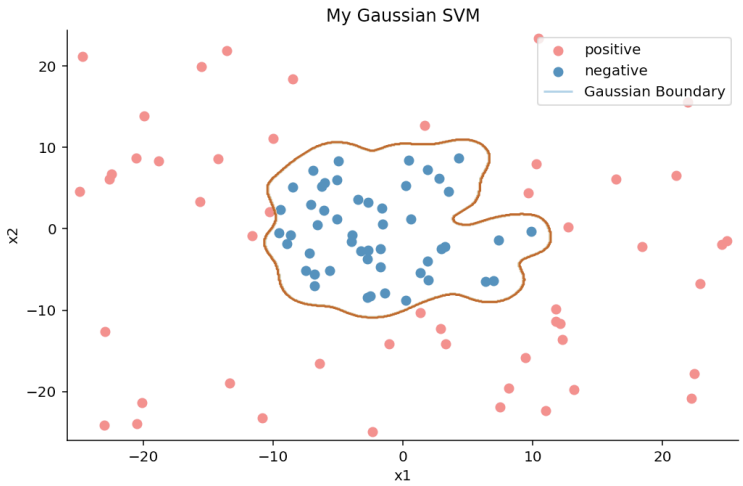
I use 2 as the degree of my polynomial SVM, and it is sufficient enough to classify all the training data points correctly. The decision boundary looks like an ellipse, however, due to its shape, there is little space between different classes of data.

Result: Gaussian SVM

There are 84 support vectors which form the decision boundary of my gaussian SVM. I am not going to list out their coordinates, but below are their indices in the training dataset for your references.

Support Vectors:

[0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 75, 76, 77, 80, 81, 84, 85, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]



For gaussian SVM, I use gamma equal to 0.2 in equation exp(-gamma * $||X_n - X_m||^2$ + intercept). A larger gamma would lead to overfitting. The boundary has an irregular shape which classified the training data in a more robust way than the polynomial SVM. However, a lot more support vectors are used in Gaussian SVM, which leads to a more heavy computational cost comparing to polynomial SVM.

Challenges I Face: turn the constraints into matrix

This one also relates to the quadratic programming solver. In order to implement polynomial SVM from scratch, I have to fully understand the dual optimization problem and turn it into matrix before feeding them into the solver. I misunderstood the goal function of dual problem at first, and couldn't return a feasible output. Finally, it turns out that I should put inner product of every pair of input X in the matrix.

III. Code Level Optimization

I write this code in an object-oriented manner. Since this homework contains different kind of kernels, I wrote different functions for them, and also another function **kernelTrick** to call them. This allows me to use different kernels without writing duplicate functions for plotting or solving equations.

IV. Software Familiarization

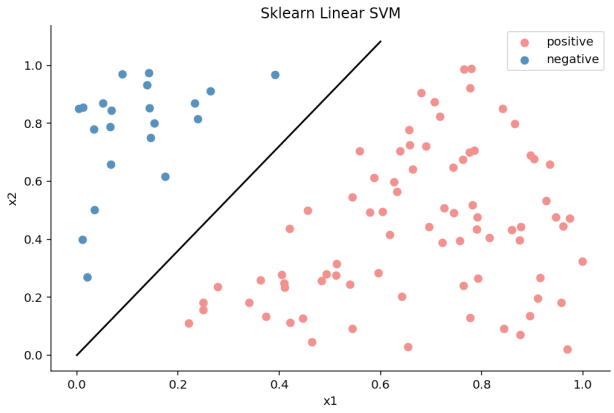
sklearn.svm.**SVC**

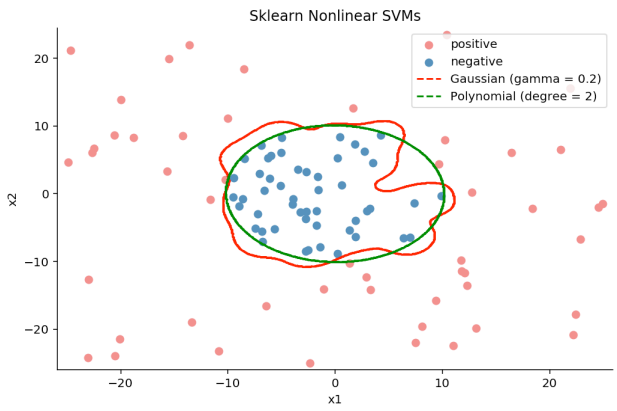
There is a parameter 'kernel' which allows user to specify which kernel to apply, 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'. 'rbf' means gaussian kernel. User can also specify the degree for polynomial

kernel and gamma for rbf kernel. Also, there is another parameter, C, which allows user to train a SVM with soft margin.

sklearn.svm.**LinearSVC**
One can also use LinearSVC to train a linear SVM model. According to the sklearn document, it has more flexibility in the choice of penalties and loss functions than the SVC described above and should scale better to large numbers of samples.

Below are the results after applying sklearn library to the input data, and both results are similar to mine.





V. Applications

SVM can be used to classify images in many fields. Today I would like to briefly introduce its application in the environmental science and the geospatial field.

Remote sensing is the process of detecting and monitoring the physical characteristics of an area by measuring its reflected and emitted radiation at a distance (typically from satellite or aircraft). Special cameras collect remotely sensed images, which help researchers "sense" things about the Earth.

And SVM could be used to classify remote sensing datasets in a robust way. Since remote sensing datasets are usually nonlinearly separable in the input space, SVM provides the technique of projecting the training data to a feature space of higher (infinite) dimension by use of a kernel function. Along with the usage of soft margin, SVM could prevent over-fitting while being robust.

SVMs are intrinsically binary classifiers, however, they can still be adopted to multi-class tasks associated with remote sensing studies.Two of the common feasible approaches to enable this adaptation are the One-Against-One (1A1) and One- Against-All (1AA) techniques.

Reference:
https://arxiv.org/pdf/0709.3967.pdf