Support Vector Machines Implementation and Applications

Chaoyu Li, chaoyuli@usc.edu, 6641732094

DSCI-552: Machine Learning for Data Science

Instructor: Satish Kumar Thittamaranahalli

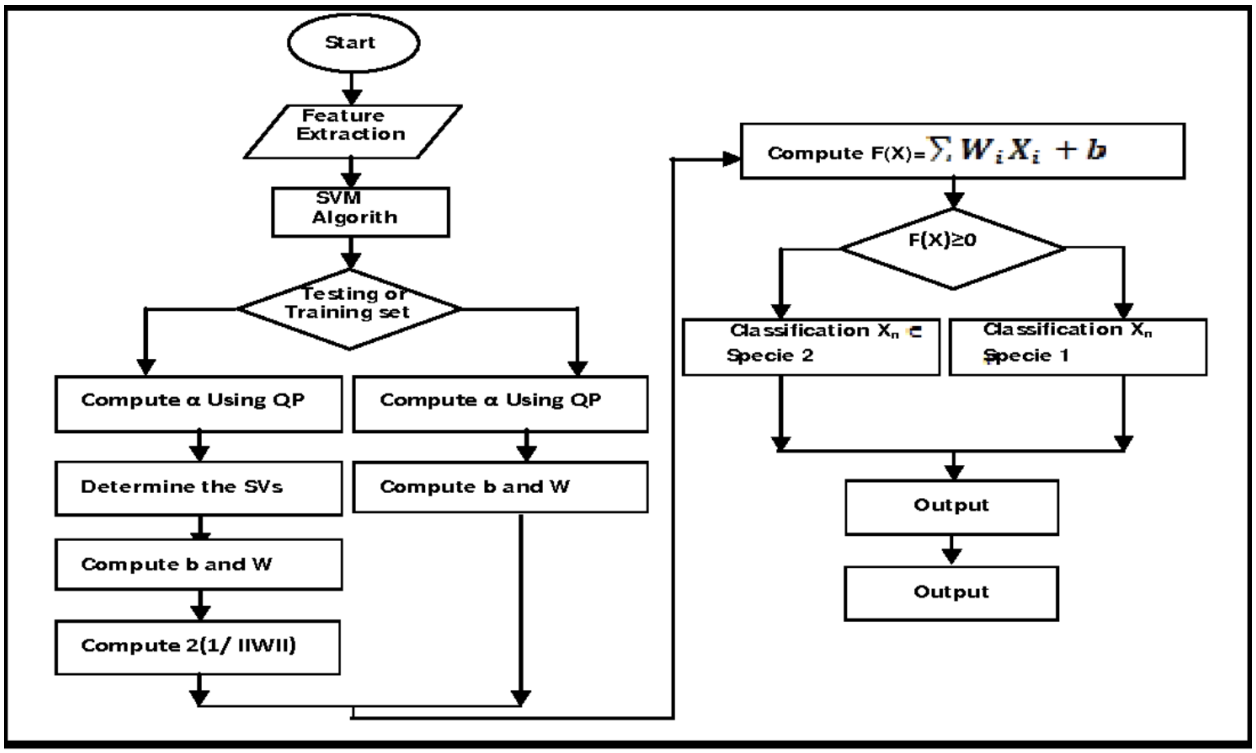
Apr. 8, 2022

Table of Contents

1 Basic Algorithm and Code Functions Description

1.1 Implementation of SVM

1.1.1 Basic Algorithm of SVM



1.1.2 Code Functions of SVM

- **train_test_split(X, y, test_size, randomseed)**

It read a txt file that contains data and labels and splits them into train and test sets. The test_size here represents the ratio of train set and test set. The random seed(I choose 7) here is used to shuffle the data.

- **linear_kernel()**

It is the kernel function I used to deal with linear data. The kernel function here is the inner product. The formula is shown below:

$$k(\vec{x}, \vec{y}) = \vec{x}^T \vec{y}$$

- **polynomial_kernel(power, coef)**

  It is the kernel function I used to deal with non-linear data. The function polynomial_kernel computes the degree-d polynomial kernel between two vectors. The polynomial kernel represents the similarity between two vectors. Conceptually, the polynomial kernels consider not only the similarity between vectors under the same dimension, but also across dimensions. When used in machine learning algorithms, this allows accounting for feature interaction. The formula is shown below:

$$k(\vec{x}, \vec{y}) = (\gamma \vec{x}^T \vec{y} + c_0)^d$$

- **accuracy_score(y_true, y_pred)**

  It calculates the accuracy by comparing the ground truth with the predicted result.

- **ClassSVM(C, kernel, power, coef, isLinear)**

  It is the init function of class SVM. It accepts C as the penalty term, kernel as the type of kernel function, isLinear as the input data is linear or non-linear.

- **SVM.fit(X, y)**

  It is the training function of SVM. Here I use a library named “CVXOPT” as the Quadratic Programming solver. It has its own standard form to input the data and get the output of the result. The standard problem form of CVXOPT is:

$$\begin{cases} min\ \frac{1}{2}x^T Px + q^T x \\ s.t.\ Gx \leqslant h \\ \quad Ax = b \end{cases}$$

  Here I defined all parameters ***P*, *q*, *G*, *h*, *A*, *b*.**

  - *P* is the *Q* matrix: $P=Q(i,j)=y_i y_j x_i T x_j$. The shape is(100,100).

- q is a matrix in which every item in it is -1. The shape is(100,1).
- G is a minus identity matrix: G=matrix(−np.eye(n_sample),tc="d"). The shape is(100,100).
- h is a matrix in which all the item is 0. The shape is (100,1).
- A is a matrix in which each item represents the corresponding label y: A=matrix(y, (1, $data_i$),tc = d). The shape is (1,100).
- b is a matrix that actually is a scalar 0: b=matrix(0.0).

Then, I used the function(sol=solvers.qp(P, q, G, h, A, b)) to get the α which is sol['x'] here, and then I used α back to get the w and b.

- **predict(X)**

  It classifies a point X via sign(b+w*x)=sign(b + $\sum_{i=1}^{n}(\alpha_i * y_i *$kernel($x_i$,x)) with i loops through all support vector points.

2 Data Structure Used

2.1 SVM data structure

A. The points data with labels are stored in the data structure of NpArray. The format is:

   $[[X_1, Y_{1,} y_1],[X_2, Y_2, y_2], \ldots ,[X_n, Y_n, y_n]]$.

B. The weight matrix is stored in the data structure of NpArray.

C. The support vectors are stored in the data structure of NpArray.

3 Challenges Faced and Optimizations

3.1 Challenges

A. The dataset is kind of small, so the ratio of the training set and testing set and the random seed will significantly impact the result.

3.2 Optimizations

A. I only considered the Lagrange multipler greater than a certain threshold to make the solution scalable.

B. I added a parameter C(penalty term) to further constrain the lagrange multiplers.

4 Execution of my algorithm and results

- Print of result:

Linear:

```
    pcost       dcost       gap    pres   dres
 0: -1.3958e+01 -3.0654e+01  2e+02  1e+01  2e+00
 1: -1.5324e+01 -2.7619e+01  6e+01  3e+00  5e-01
 2: -2.6835e+01 -3.4397e+01  3e+01  1e+00  2e-01
 3: -2.9238e+01 -3.5072e+01  8e+00  1e-01  1e-02
 4: -3.2992e+01 -3.3364e+01  5e-01  7e-03  9e-04
 5: -3.3228e+01 -3.3235e+01  9e-03  9e-05  1e-05
 6: -3.3232e+01 -3.3232e+01  1e-04  9e-07  1e-07
 7: -3.3232e+01 -3.3232e+01  1e-06  9e-09  1e-09
```

W: [[7.56855203]

[-3.0300411]]

b: -0.3381934194573146

Support_vectors:

[[0.23918196 0.81585285]

[0.24979414 0.18230306]

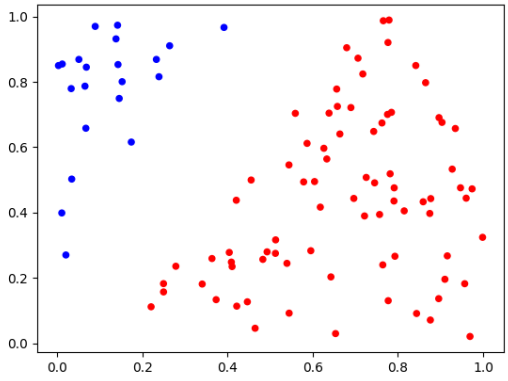
[0.23307747 0.86884518]

[0.02066458 0.27003158]]

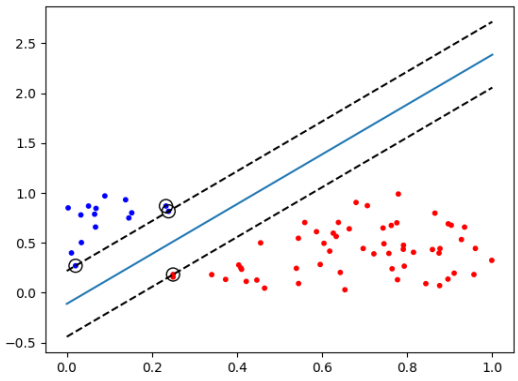
[-1. 1. -1. -1.]

Training Accuracy: 1.0

Testing Accuracy: 1.0



Original_image



SVM_result

Non-Linear:

```
    pcost       dcost       gap    pres   dres
```

0: -3.2555e+01 -7.7925e+01 4e+02 2e+01 2e+00
1: -1.2439e+02 -1.6327e+02 2e+02 1e+01 1e+00
2: -3.4296e+02 -3.6553e+02 2e+02 9e+00 1e+00
3: -8.1696e+02 -8.1715e+02 3e+02 8e+00 1e+00
4: -1.4118e+03 -1.3154e+03 5e+02 8e+00 9e-01
5: -9.3997e+02 -7.7533e+02 7e+02 7e+00 8e-01
6: -8.9169e+02 -7.0536e+02 7e+02 7e+00 8e-01
7: -4.0904e+02 -1.7937e+02 7e+02 4e+00 4e-01
8: -1.1964e+02 -1.5525e+01 2e+02 1e+00 1e-01
9: -1.7080e+00 -5.4263e-02 5e+00 2e-02 2e-03
10: -1.8426e-02 -4.9315e-02 6e-02 8e-05 1e-05
11: -3.1453e-02 -3.4348e-02 4e-03 5e-06 6e-07
12: -3.3981e-02 -3.4013e-02 5e-05 5e-08 7e-09
13: -3.4009e-02 -3.4009e-02 5e-07 5e-10 7e-11
14: -3.4009e-02 -3.4009e-02 5e-09 5e-12 7e-13

W: [[-0.40140899]

[-0.09275295]]

Support_vectors:

[[1.3393313 -10.29098822]

[-10.260969   2.07391791]

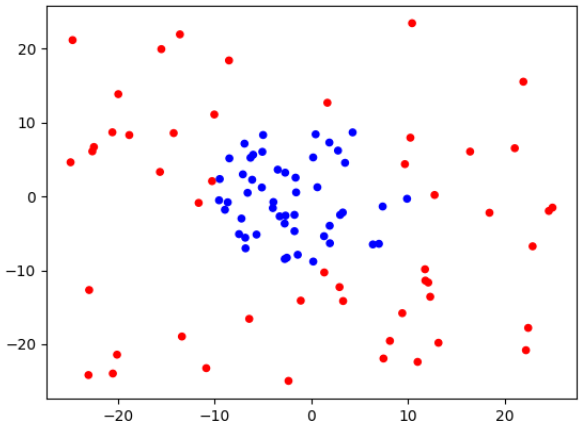
[-6.90647562   7.14833849]

[9.90143538 -0.31483149]]

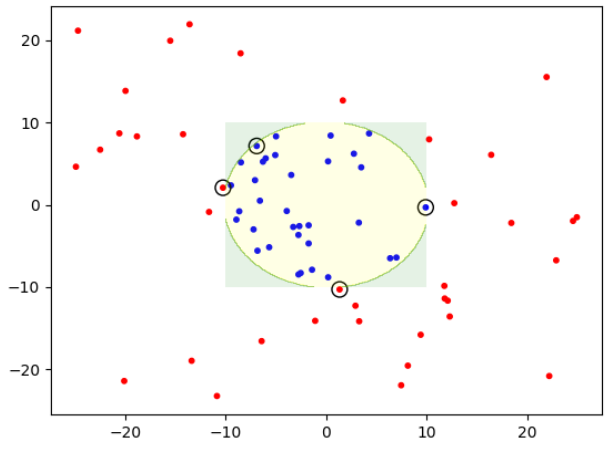
[1. 1. -1. -1.]

Training Accuracy: 1.0

Testing Accuracy: 1.0



Original_image



SVM_result

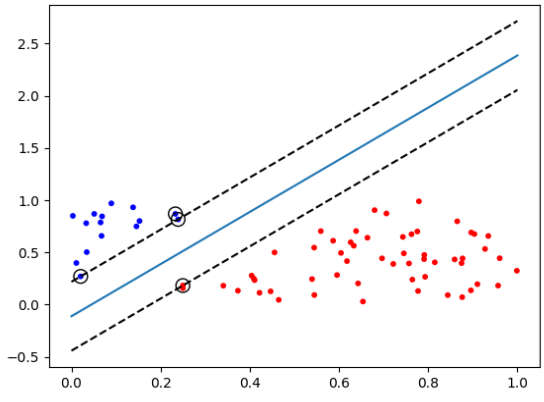
5 Comparison between my algorithm and scikit-learn

- The code of scikit-learn in Linear SVM is:

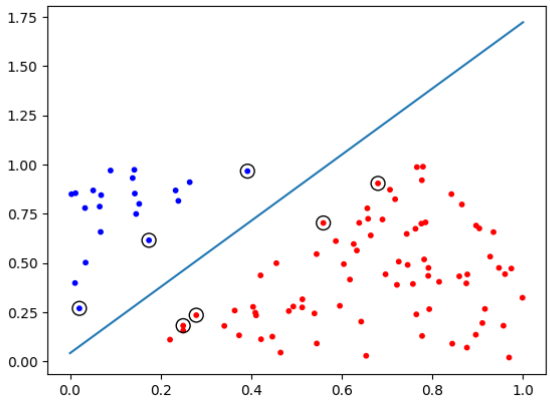
```
from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt
import sys
data_linear = np.loadtxt('linsep.txt', dtype='float', delimiter=',')
X = data_linear[:, 0:2]
y = data_linear[:, 2]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, seed=7)
clf_linear = svm.SVC(C=10.0, kernel='linear', degree=3, gamma='scale',
                     coef0=0.0, shrinking=True, probability=False, tol=0.00001,
                     cache_size=200, class_weight=None, verbose=False, max_iter=-1,
                     decision_function_shape='ovr', random_state=None)
clf_linear.fit(X_train, y_train)
```

- The result is:

Linear:
support vectors: **[[0.02066458 0.27003158]**
[0.17422964 0.6157447]
[0.3917889 0.96675591]
[0.24979414 0.18230306]
[0.55919837 0.70372314]
[0.6798148 0.90468041]
[0.27872572 0.23552777]]
weights: **[[5.99544489 -3.56898996]]**
Testing Accuracy: 1.0



My_result



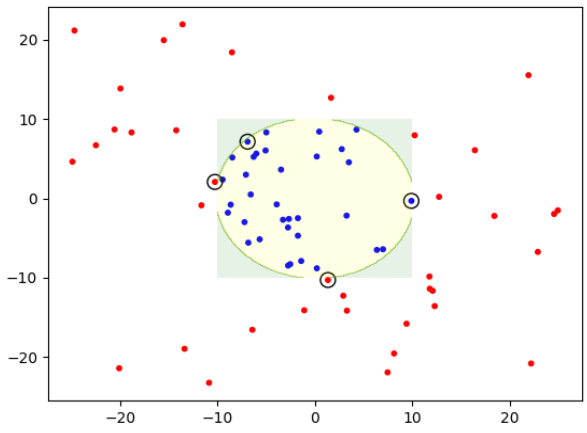
Sklearn_result

- The code of scikit-learn in Non-Linear SVM is:

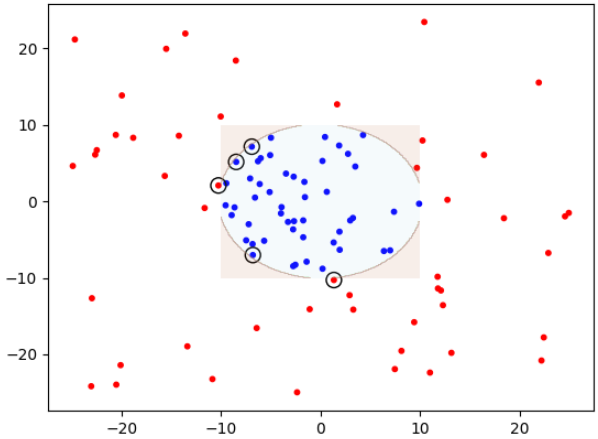
```
from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt
import sys
data_nonlinear = np.loadtxt('nonlinsep.txt', dtype='float', delimiter=',')
X = data_nonlinear[:, 0:2]
y = data_nonlinear[:, 2]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, seed=7)
clf_nonlinear = svm.SVC(C=1000.0, kernel='poly', degree=2, gamma='scale',
                        coef0=0.0, shrinking=True, probability=False, tol=0.00001,
                        cache_size=200, class_weight=None, verbose=False, max_iter=-1,
                        decision_function_shape='ovr', random_state=None)
clf_nonlinear.fit(X_train, y_train)
```

- The result is:

Non-Linear:
support vectors: **[[-8.47422847 5.15621613]**
[-6.90647562 7.14833849]
[-6.80002274 -7.02384335]
[1.3393313 -10.29098822]
[-10.260969 2.07391791]]
Testing Accuracy: 1.0



My_result



Sklearn_result

6 Application of SVM

- **Face detection:** SVMs classify parts of the image as a face and non-face. It contains training data of n x n pixels with a two-class face (+1) and non-face (-1). Then it extracts features from each pixel as face or non-face.
- **Text categorization:** SVMs allow Text and hypertext categorization for both inductive and transductive models. They use training data to classify documents into different categories. It categorizes on the basis of the score generated and then compares with the threshold value.
- **Classification of images:** SVMs provide better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.
- **Bioinformatics:** It includes protein classification and cancer classification. We can use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems.
- **Handwriting recognition:** We can use SVMs to recognize handwritten characters.

7 Contributions

All works were completed by Chaoyu Li.