Machine Learning, Spring 2020

Project Two – SVM

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

PyTorch tutorial: https://pytorch.org/tutorials/

Prof. Yi Fang (yfang@nyu.edu)

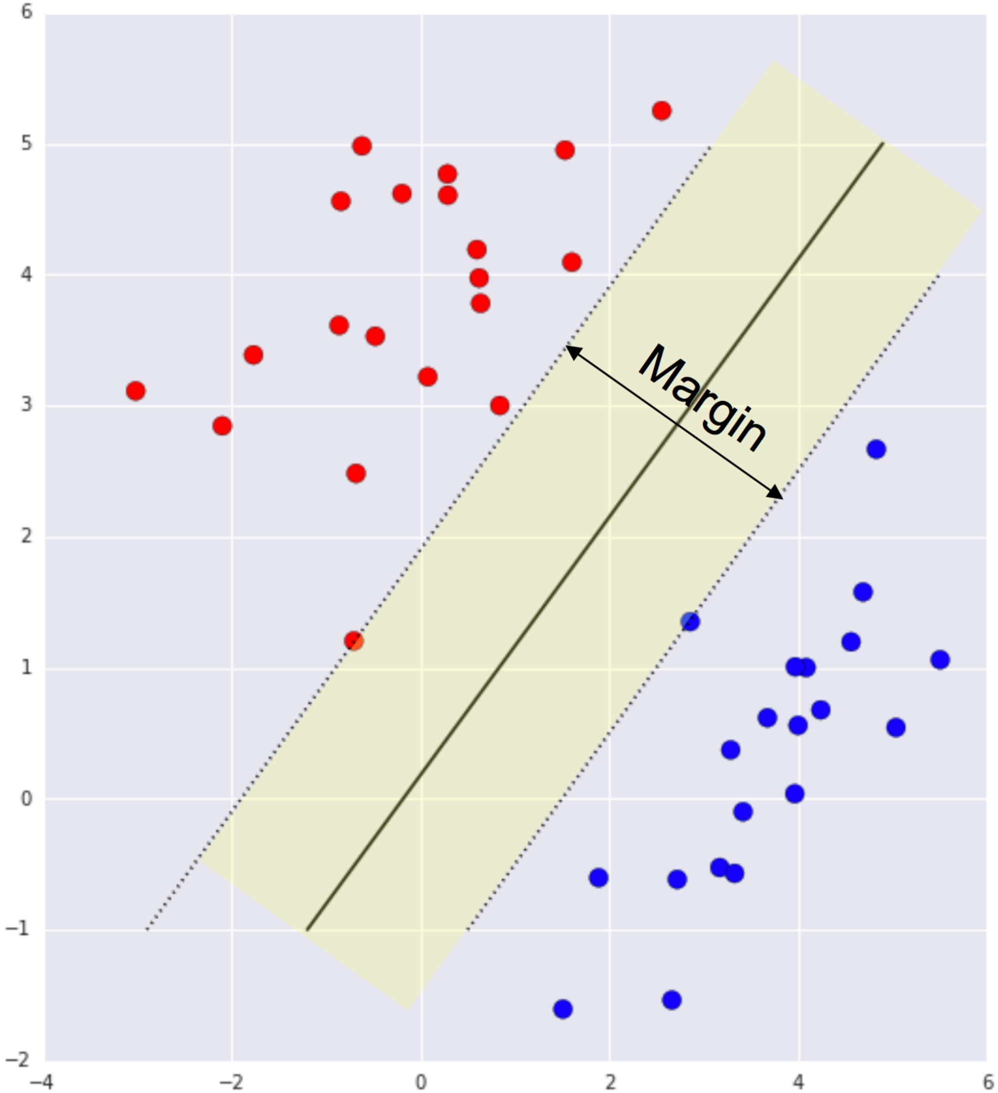
Support Vector Machine

SVM Decision Boundary:

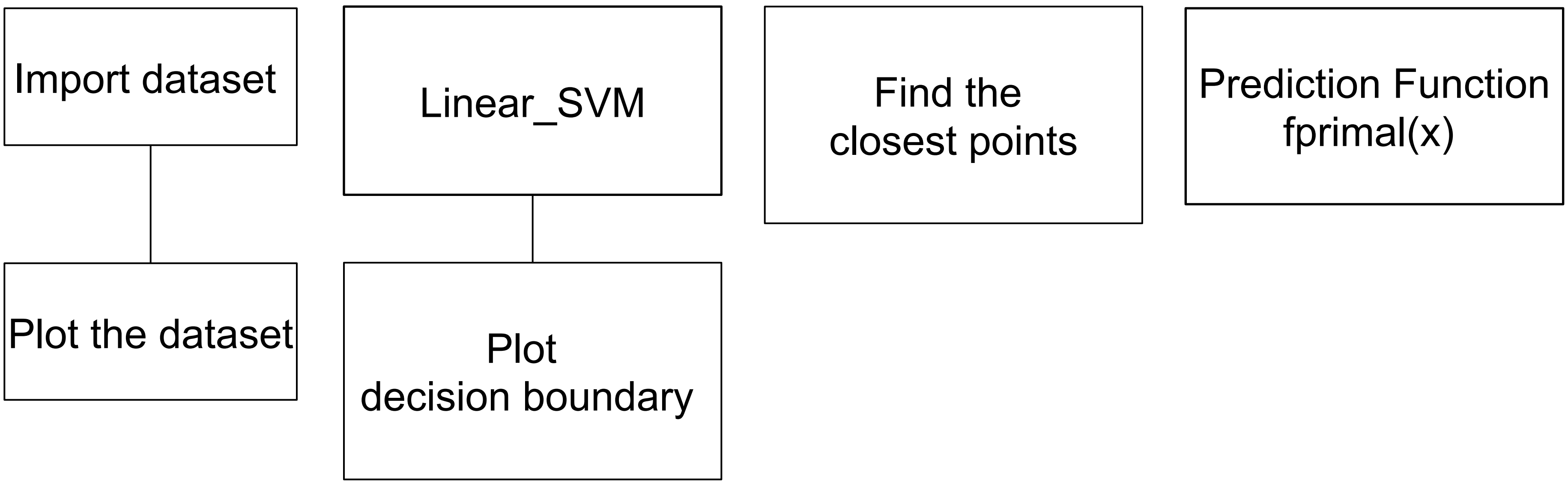
$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

$$\text{s.t.} \quad \theta^T x^{(i)} \geq 1 \qquad \text{if } y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1 \qquad \text{if } y^{(i)} = 0$$



Main Modules for SVM



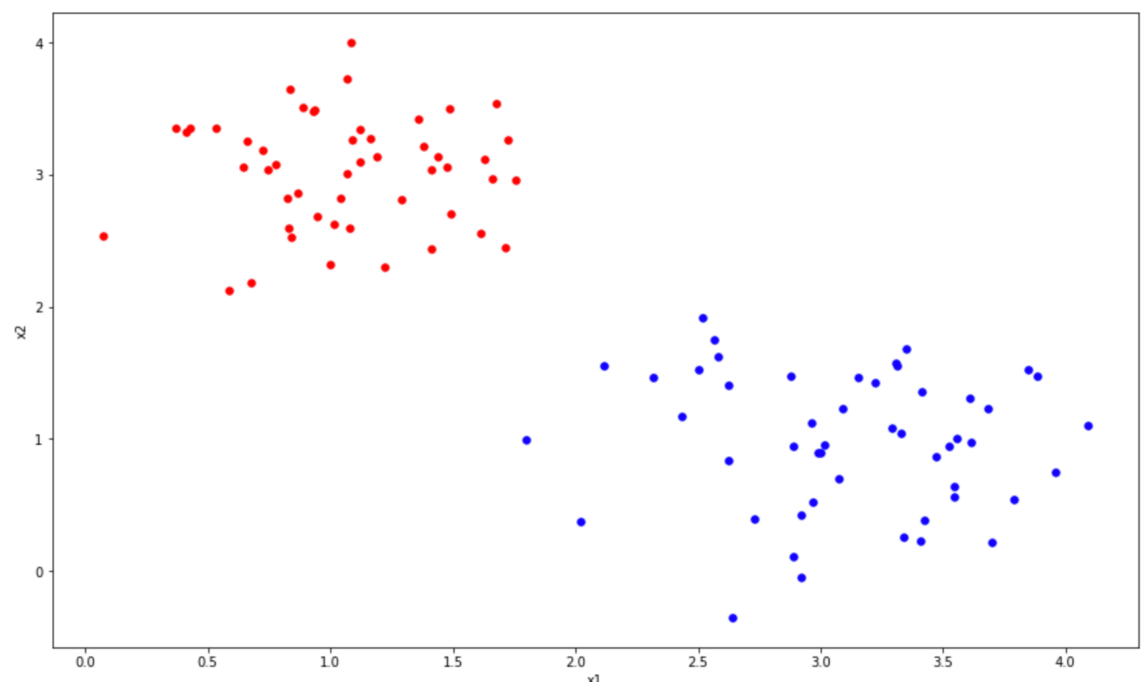
Main Modules for SVM

Import dataset

Plot the dataset

```
# Use numpy function genfromtxt(…, delimiter=…) to load from files.
# Store the data from "X.csv" to X and target from "y.csv" to y
# Use np.where to find all index of data which y=1 and store them to idx_1
# same as y=-1, store them to idx_2
# make the plot use plt.scatter(X[idx_1,0], X[idx_1,1], s=30, c='b', marker="o")
# Set the x label with x1 and y label with x2
# plt.show()
```

Expected result:



New Notation

Previously we used

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \qquad x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

$$\mathbf{w}^T x$$

$$y \in \{0,1\}$$

This lecture, we separate the intercept term from the other weights. The mathematics of this lecture makes easier. We change notation to make this clearer.

$$w_0 \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \qquad x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

$$w_0 + \mathbf{w}^T x$$

$$y \in \{-1,1\}$$

Support vector machines as a QP (Quadratic Programming)

SVM Primal Problem

$$\begin{cases} \min\limits_{\mathbf{w},b} & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{with} & y_i(\mathbf{w}^\top \mathbf{x}_i + \underline{b}_{w_0}) \geq 1, i = 1, n \end{cases}$$

$\Leftrightarrow$

QP Formulation

$$\begin{cases} \min\limits_{\mathbf{z} \in \mathbb{R}^{d+1}} & \frac{1}{2}\mathbf{z}^\top A\mathbf{z} - \mathbf{d}^\top \mathbf{z} \\ \text{with} & B\mathbf{z} \leq \mathbf{e} \end{cases}$$

$\mathbf{z} = (\mathbf{w}, b)^\top$, $\mathbf{d} = (0, \ldots, 0)^\top$, $A = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$, $B = -[\text{diag}(\mathbf{y})X, \mathbf{y}]$ and $\mathbf{e} = -(1, \ldots, 1)^\top$

Solve it using a standard QP solver such as (for instance)

```
% QUADPROG Quadratic programming.
%    X = QUADPROG(H,f,A,b) attempts to solve the quadratic programming problem:
%
%              min 0.5*x'*H*x + f'*x   subject to:  A*x <= b
%               x
%  so that the solution is in the range LB <= X <= UB
```

For more solvers (just to name a few) have a look at:

- plato.asu.edu/sub/nlores.html#QP-problem
- www.numerical.rl.ac.uk/people/nimg/qp/qp.html

Main Modules for SVM

Linear_SVM

Write the function linear_svm(X, y) that:

- takes in as arguments the data matrix X and the labels $\mathbf{y}$
- solves the SVM primal QP problem
- returns $\mathbf{w}$ and w_0

In CVXOPT, the quadratic programming problem solver, ***cvxopt.solvers.qp***, solves the following problem:

$$\begin{aligned} \min_x \quad & \frac{1}{2}x^T P x - q^T x \\ \text{s.t.} \quad & Gx \preceq h \\ \text{and} \quad & Ax = b \end{aligned}$$

Note that $Gx \preceq h$ is taken elementwise.

The solver's (simplified) API is `cvxopt.solvers.qp(P, q, G, h, A, b)` where only P and q are required.

You will need to match the solver's API.

The solver's argument's type must be CVXOPT matrices. Please look at this link for more information. I suggest you first create the arguments as NumPy arrays and matrices and then convert them to CVXOPT matrices (For example, first import the library: `from cvxopt import matrix` then convert a NumPy matrix `P` to a CVXOPT matrix using `P = matrix(P)`)

What is return by the solver is a Python dictionary. If you save the return value in a variable called `sol` (i.e. `sol = solvers.qp(...)`), you can access to the solution of the quadratic programming problem by typing `sol["x"]`.

Main Modules for SVM

Linear_SVM

In [ ]:
```
def linear_svm(X,y):
    solvers.options['show_progress'] = False
#     store the shape of X to two variables: N,F

#     create the Identity matrix using np.diag and np.ones

#     create the Q matrix using np.zeros

#     for each element in Q:

#         when row number is 0, set Q[row, col]=0

#         when col number is 0 set Q[row,col]=0

#         else, compute Identity [row-1,col-1] and set it to Q[row,col]

#     use cvxopt.matrix to create a new variable p with value Q

#     use cvxopt.matrix to create a new variable q with value np.zeros(F+1)

#     create an empty list

#     for n in range(N):

#         create a zero matric with size F+1

#         for each element in the matric above:

#             when the index=0, then set it to 1

#             else, set the value to X[n].T[i-1]

#         append the y[n]*updated matric to the empty list above (the one above the for loop

#     change the empty list to the np array and times -1

#     use cvxopt.matrix to convert above np array and store it in a variable: G

#     create a variable named h with value np.ones(N)*-1 and convert it to cvxopt

#     solve the primal using cvxopt.solvers.qp

#     return the answer.


# fit svm classifier

# print the weights
```

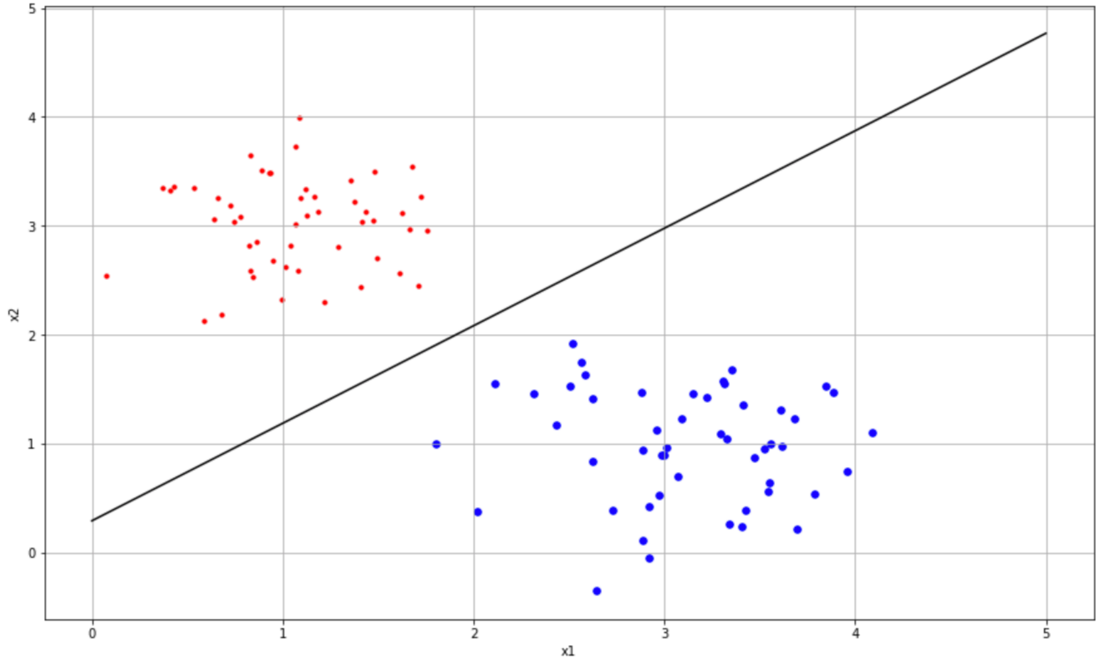
Main Modules for SVM

Plot
decision boundary

Plotting the decision boundary

```
In [ ]: def plot_data_with_decision_boundary(X, y, w, w0, fig_size=(15, 9), labels=['x1', 'x2']):
        #     plot the dataset

        #     find the slope of the decision boundary

        #     find the intercept.

        #     generate several x values np.arrange()

        #     calculate its y values using intercept and slope

        #     plot a line


        # plotting the points and decision boundary using the above function
```

Example results:



Main Modules for SVM

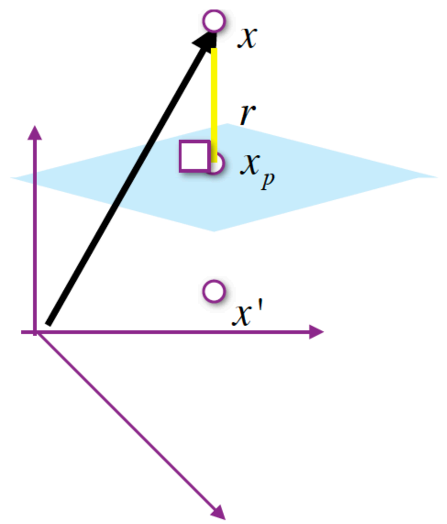
Find the closest points

- After obtain the **w** and w_0 from *Linear_svm* function
- The hyperplane should be $w^T x + w_0 = 0$
- Denote $f(x) = w^T x + w_0$
- Calculate the distance for all the points

x, a point

x_p, the normal projection of x onto **w**,

Note that
$$x = x_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|_2}$$



$$f(x) = w_0 + \mathbf{w}^T x = w_0 + \mathbf{w}^T\left(x_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|_2}\right)$$

$$= w_0 + \mathbf{w}^T x_p + \mathbf{w}^T r\frac{\mathbf{w}}{\|\mathbf{w}\|_2}$$ observe that $\mathbf{w}^T\mathbf{w} = \|\mathbf{w}\|_2^2$

$$= r\|\mathbf{w}\|_2$$

Consequently: $r = \dfrac{f(x)}{\|\mathbf{w}\|_2}$

Note that r can be positive or negative depending on which side of the hyperplane x lies

Distance for x

Main Modules for SVM

Find the closest points

- After obtain the **w** and w_0 from *Linear_svm* function
- The hyperplane should be $w^T x + w_0 = 0$
- Denote $f(x) = w^T x + w_0$
- Calculate the distance for all the points
- Return those with smallest distances (be careful for the negative values!)

Determine which points are closest to the decision boundary. What is the functional margin of the points closest to the decision boundary?

In [ ]:
```
# calculate distance from each point to the decision boundary

# find the nearest data points and its index.
```

Example results:

```
MARGIN  1.0
Points Idx:  [25 49 68]
Points:  [[2.11457352 1.5537852 ]
 [2.51879639 1.91565724]
 [1.71138733 2.45204836]]
```

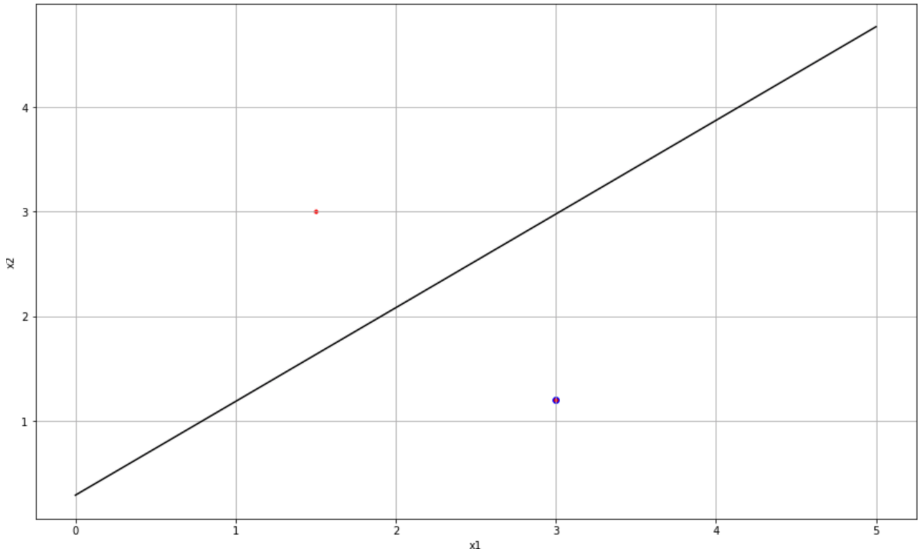
Main Modules for SVM

Prediction Function
fprimal(x)

Write the decision function $f_{\mathtt{primal}}(\mathbf{x})$ to predict examples. Use this function to predict the label of $(3.0, 1.5)^T$ and $(1.2, 3.0)^T$

In [ ]:
```
def f_primal(x):
#     return the predicted value using svm primal

# using f_ primal() to predict (3.0, 1.5) and (1.2, 3.0) and plot the figure.
```

Example result:



Pipeline for SVM