Session 7 notebook

The notebook has been written during the session
please watch the video on "Course Materials" section of iLearn for the full description

May 18, 2020

0.1 Support Vector Machine (SVM)

SVM is a supervised algorithm which can be used for classification and regression. However, it is originally proposed for binary classification.

Given a dataset ($\vec{\mathbf{x}}$ or $\vec{\mathbf{CE}}(\mathbf{x})$) which are labeled with $y_i \in \{-1, 1\}$, we want to find a linear separator with the largest margin between these two types of data. The distance of any point (vector) from a line/plane can be obtained by the projection of that point on the unit normal vector of the line/plane ($\hat{w}$).

$$D_i = \hat{w}.\vec{\mathbf{x_i}} + b$$

So, if $D > 0$ then we are in the blue zone of the figure below and if $D < 0$ then we are in the red zone. It's our decision rule. The problem is to find optimal $\hat{w}$ and b.

Source of the figure:https://svm.michalhaltuf.cz/support-vector-machines

To find the optimal seperator line, let's consider a margin between our training sample such that:

$$\vec{w}.x_{blue} + b \geq 1$$
$$\vec{w}.x_{red} + b \leq -1$$

Note that I removed the hat sign from w since it is not a unit vector anymore. We multiplied ($\hat{w}.\vec{\mathbf{x_i}} + b \geq$ arbitrary margin) by (1/arbitrary margin). If you define label such that $y_i = -1$ for reds and $y_i = 1$ for blues, then:

$$y_i(\vec{w}.\vec{\mathbf{x_i}} + b) - 1 \geq 0$$

It gives us two boundaries: Blue: $\vec{w}.\vec{\mathbf{x_i}} + b = 1$; Pink: $\vec{w}.\vec{\mathbf{x_i}} + b = -1$

The margin size can be computed using two support vectors, one on blue dashed line and the other on pink dashed line.

$$\text{Margin} = (\vec{x}_{\text{on blue dashed line}} - \vec{x}_{\text{on pink dashed line}}).\frac{\vec{w}}{||w||} = (1 - b - (-1 - b)\frac{1}{||w||} = \frac{2}{||w||}$$

To get the optimal seperator we want to maximize the margin. So, we are going to maximize $\frac{2}{||w||}$ or minimize $||w||$. We know that $||w||$ is $\sqrt{\vec{w}.\vec{w}}$. Therefore, we can minimize $\frac{1}{2}||w||^2 = \frac{1}{2}\vec{w}.\vec{w}$ which is computationally convenient.

So, we can solve the following optimization problem:

$$min\ (\frac{1}{2}w^T.w)$$

$$\text{subject to: } y_i(w.\mathbf{x_i} + b) - 1 \geq 0$$

Is that all? Not yet! Imagine we have some outliers (red points in blue region). In the above expression for optimization problem we did not take into account a level of tolerance for those outliers. We can consider this by defining a slack parameter (ξ)

$$min\ (\frac{1}{2}w^T.w + C\sum_i \xi_i)$$

$$\text{subject to: } y_i(w.\mathbf{x_i} + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

To incorporate all the constraints, we can write the following Lagrangian and minimize it:

$$L = \frac{1}{2}w^T.w + C\sum_i \xi_i + \sum_i \alpha_i[1 - \xi_i - y_i(w.\mathbf{x_i} + b)] + \sum_i \eta_i(-\xi_i)$$

To minimize the Lagrangian, take the first derivative of the function and put that equal to zero.

$$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i y_i \mathbf{x_i} = 0 \Rightarrow w = \sum_i \alpha_i y_i \mathbf{x_i}$$

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0$$

Since $\eta_i \geq 1$, it implies that $\alpha_i \leq C$.

Let's substitute $w = \sum_i \alpha_i y_i \mathbf{x_i}$ in the Lagrangian equation.

$$L = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (\vec{\mathbf{x_i}}.\vec{\mathbf{x_j}})$$

And the constraints are:

$$(1): \sum_i \alpha_i y_i = 0$$

$$(2): 0 \leq \alpha_i \leq C$$

If we define: $q = (1\ 1\ 1\ ...\ 1)$ and $P_{ij} = y_i y_j (\vec{\mathbf{x}_\mathbf{i}}.\vec{\mathbf{x}_\mathbf{j}})$, then:

$$max_\alpha (q^T \alpha - \frac{1}{2} \alpha^T P \alpha)$$

$$y^T \alpha = 0$$

$$0 \leq \alpha_i \leq C$$

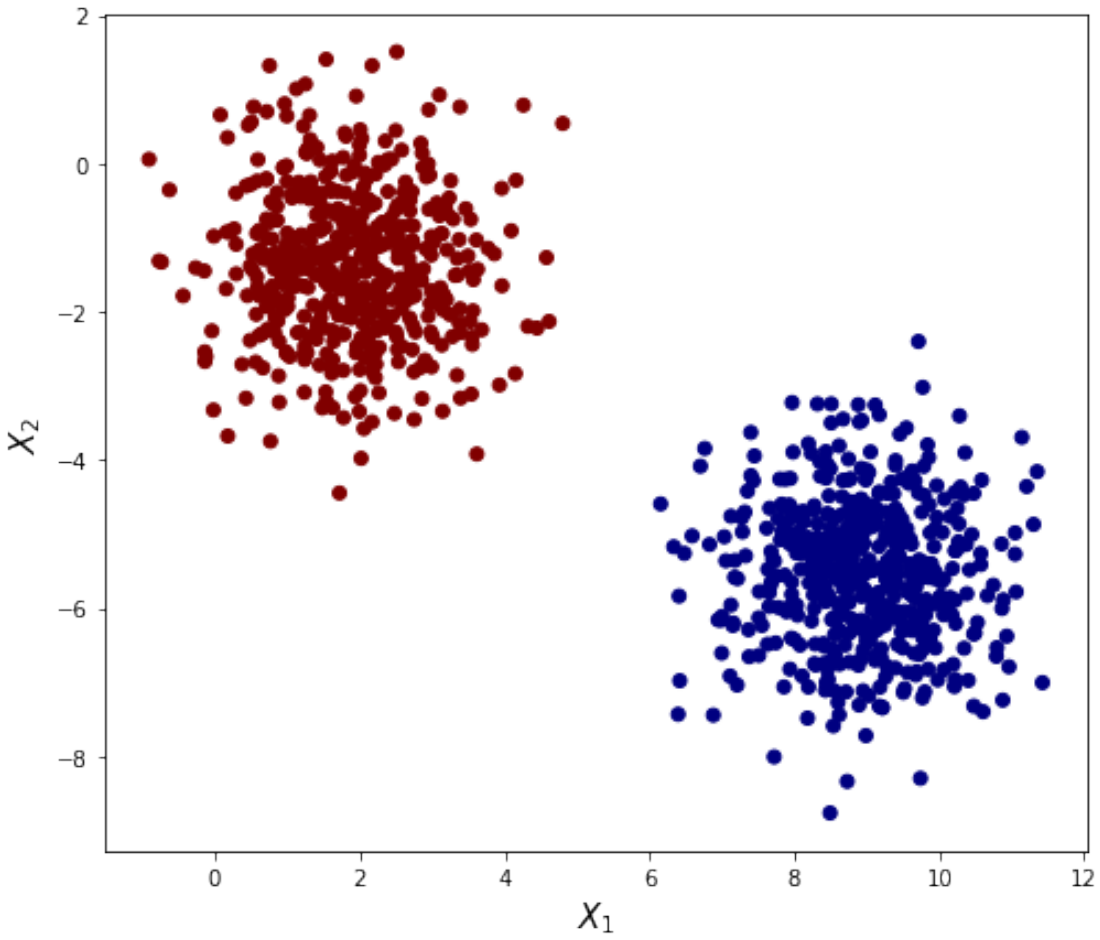
So, building an SVM model can be done by solving the above optimization problem.

We will use "cvxopt" package in Python to perform the optimization:

[19]:
```
import numpy as np
import matplotlib.pyplot as plt
import cvxopt as opt
from sklearn.datasets.samples_generator import make_blobs
```

Let's create a pseudo dataset:

[20]:
```
plt.figure(figsize=(8,7))
X,y=make_blobs(n_samples=1000,n_features=2,centers=2,cluster_std=1,random_state=200)
plt.scatter(X[:,0],X[:,1],c=y,cmap='jet')
plt.xlabel('$X_1$',fontsize=15)
plt.ylabel('$X_2$',fontsize=15)
plt.show()
```



Data are labeled with 0 and 1. To be consistent with our notation, we will map 0s to -1.

[21]:
```
y[y==0]=-1
```

[22]:
```
help(opt.solvers.qp)
```

```
Help on function qp in module cvxopt.coneprog:

qp(P, q, G=None, h=None, A=None, b=None, solver=None, kktsolver=None,
initvals=None, **kwargs)
    Solves a quadratic program

        minimize    (1/2)*x'*P*x + q'*x
        subject to  G*x <= h
                    A*x = b.


    Input arguments.
```

```
    P is a n x n dense or sparse 'd' matrix with the lower triangular
    part of P stored in the lower triangle.  Must be positive
    semidefinite.

    q is an n x 1 dense 'd' matrix.

    G is an m x n dense or sparse 'd' matrix.

    h is an m x 1 dense 'd' matrix.

    A is a p x n dense or sparse 'd' matrix.

    b is a p x 1 dense 'd' matrix or None.

    solver is None or 'mosek'.

    The default values for G, h, A and b are empty matrices with
    zero rows.


Output arguments (default solver).

    Returns a dictionary with keys 'status', 'x', 's', 'y', 'z',
    'primal objective', 'dual objective', 'gap', 'relative gap',
    'primal infeasibility, 'dual infeasibility', 'primal slack',
    'dual slack'.

    The 'status' field has values 'optimal' or 'unknown'.
    If the status is 'optimal', 'x', 's', 'y', 'z' are an approximate
    solution of the primal and dual optimal solutions

        G*x + s = h,  A*x = b
        P*x + G'*z + A'*y + q = 0
        s >= 0, z >= 0
        s'*z = o.

    If the status is 'unknown', 'x', 's', 'y', 'z' are the last
    iterates before termination.  These satisfy s > 0 and z > 0, but
    are not necessarily feasible.

    The values of the other fields are defined as follows.

    - 'primal objective': the primal objective (1/2)*x'*P*x + q'*x.

    - 'dual objective': the dual objective

          L(x,y,z) = (1/2)*x'*P*x + q'*x + z'*(G*x - h) + y'*(A*x-b).
```

```
    - 'gap': the duality gap s'*z.

    - 'relative gap': the relative gap, defined as

          gap / -primal objective

      if the primal objective is negative,

          gap / dual objective

      if the dual objective is positive, and None otherwise.

    - 'primal infeasibility': the residual in the primal constraints,
      defined as the maximum of the residual in the inequalities

          || G*x + s + h || / max(1, ||h||)

      and the residual in the equalities

          || A*x - b || / max(1, ||b||).


    - 'dual infeasibility': the residual in the dual constraints,
      defined as

          || P*x + G'*z + A'*y + q || / max(1, ||q||).

    - 'primal slack': the smallest primal slack, min_k s_k.
    - 'dual slack': the smallest dual slack, min_k z_k.

    If the exit status is 'optimal', then the primal and dual
    infeasibilities are guaranteed to be less than
    solvers.options['feastol'] (default 1e-7).  The gap is less than
    solvers.options['abstol'] (default 1e-7) or the relative gap is
    less than solvers.options['reltol'] (default 1e-6).

    Termination with status 'unknown' indicates that the algorithm
    failed to find a solution that satisfies the specified tolerances.
    In some cases, the returned solution may be fairly accurate.  If
    the primal and dual infeasibilities, the gap, and the relative gap
    are small, then x, y, s, z are close to optimal.


Output arguments (MOSEK solver).

    The return dictionary has two additional fields
    'residual as primal infeasibility certificate' and
```

```
'residual as dual infeasibility certificate', and 'status' field
can also have the values 'primal infeasible' or 'dual infeasible'.

If the exit status is 'optimal', the different fields have the
same meaning as for the default solver, but the the magnitude of
the residuals and duality gap is controlled by the MOSEK exit
criteria.  The 'residual as primal infeasibility certificate' and
'residual as dual infeasibility certificate' are None.

Status 'primal infeasible'.
- 'x', 's': None.
- 'y', 'z' are an approximate certificate of infeasibility

      G'*z + A'*y = 0,  h'*z + b'*y = -1,  z >= 0.

- 'primal objective': None.
- 'dual objective': 1.0.
- 'gap', 'relative gap': None.
- 'primal infeasibility' and 'dual infeasibility': None.
- 'primal slack': None.
- 'dual slack': the smallest dual slack min z_k.
- 'residual as primal infeasibility certificate': the residual in
  the condition of the infeasibility certificate, defined as

      || G'*z + A'*y || / max(1, ||c||).

- 'residual as dual infeasibility certificate': None.

Status 'dual infeasible'.
- 'x', 's' are an approximate proof of dual infeasibility

      P*x = 0,  q'*x = -1,  G*x + s = 0,  A*x = 0,  s >= 0.

- 'y', 'z': None.
- 'primal objective': -1.0.
- 'dual objective': None.
- 'gap', 'relative gap': None.
- 'primal infeasibility' and 'dual infeasibility': None.
- 'primal slack': the smallest primal slack min_k s_k .
- 'dual slack': None.
- 'residual as primal infeasibility certificate': None.
- 'residual as dual infeasibility certificate: the residual in
  the conditions of the infeasibility certificate, defined as
  the maximum of

      || P*x || / max(1, ||q||),
      || G*x + s || / max(1, ||h||),
      || A*x || / max(1, ||b||).
```

```
        If status is 'unknown', all the other fields are None.


    Control parameters.

        The control parameters for the different solvers can be modified
        by adding an entry to the dictionary cvxopt.solvers.options.  The
        following parameters control the execution of the default solver.

            options['show_progress'] True/False (default: True)
            options['maxiters'] positive integer (default: 100)
            options['refinement']  positive integer (default: 0)
            options['abstol'] scalar (default: 1e-7)
            options['reltol'] scalar (default: 1e-6)
            options['feastol'] scalar (default: 1e-7).

        The MOSEK parameters can me modified by adding an entry
        options['mosek'], containing a dictionary with MOSEK
        parameter/value pairs, as described in the MOSEK documentation.

        Options that are not recognized are replaced by their default
        values.
```

As you can see from the data above, we can define a seperator line without outliers. So, we will define C as infinite (e.g. 10^{10}) (hard margin)

[23]:
```
C=10**10
```

[24]:
```
y=y.reshape(1000,1)
```

[25]:
```
y.shape
```

[25]:
```
(1000, 1)
```

[26]:
```
y=y.astype('float64')
```

[27]:
```
n_train=len(X)
print(len(X))
```

```
1000
```

[28]:
```
X_=y*X
P_=np.dot(X_,X_.T)
```

[29]:
```
P_.shape
```

[29]: (1000, 1000)

[30]:
```
P=opt.matrix(P_)
b=opt.matrix([0.])
h=opt.matrix(np.hstack(([0.]*n_train,[C]*n_train)))
G=opt.matrix(np.vstack((np.eye(n_train)*-1.,np.eye(n_train))))
q=opt.matrix([-1.]*n_train)
A=opt.matrix(y.T)
```

[31]:
```
sol=opt.solvers.qp(P,q,G,h,A,b)
```

```
     pcost       dcost       gap    pres   dres
 0:  2.6702e+18 -3.5976e+22  9e+22  5e-01  1e-03
 1:  5.4273e+18 -3.8382e+21  6e+21  2e-02  2e+02
 2:  5.8166e+18 -1.5206e+20  2e+20  7e-04  5e+00
 3:  3.4546e+18 -4.9291e+18  9e+18  7e-06  5e-02
 4:  5.2821e+17 -5.6838e+17  1e+18  6e-08  1e-05
 5:  7.5724e+16 -8.3387e+16  2e+17  1e-08  4e-06
 6:  1.0851e+16 -1.1923e+16  2e+16  3e-09  2e-06
 7:  1.5549e+15 -1.7088e+15  3e+15  2e-09  6e-07
 8:  2.2281e+14 -2.4487e+14  5e+14  2e-09  2e-07
 9:  3.1928e+13 -3.5089e+13  7e+13  1e-10  9e-08
10:  4.5752e+12 -5.0281e+12  1e+13  2e-10  3e-08
11:  6.5561e+11 -7.2052e+11  1e+12  7e-11  1e-08
12:  9.3948e+10 -1.0325e+11  2e+11  6e-11  4e-09
13:  1.3462e+10 -1.4795e+10  3e+10  1e-11  2e-09
14:  1.9291e+09 -2.1201e+09  4e+09  7e-12  6e-10
15:  2.7644e+08 -3.0381e+08  6e+08  3e-12  2e-10
16:  3.9612e+07 -4.3536e+07  8e+07  3e-13  1e-10
17:  5.6760e+06 -6.2389e+06  1e+07  3e-13  3e-11
18:  8.1324e+05 -8.9414e+05  2e+06  2e-14  1e-11
19:  1.1649e+05 -1.2817e+05  2e+05  1e-15  5e-12
20:  1.6677e+04 -1.8383e+04  4e+04  5e-15  2e-12
21:  2.3835e+03 -2.6405e+03  5e+03  7e-16  7e-13
22:  3.3918e+02 -3.8074e+02  7e+02  2e-15  3e-13
23:  4.7691e+01 -5.5465e+01  1e+02  1e-15  1e-13
24:  6.4720e+00 -8.3012e+00  1e+01  4e-16  4e-14
25:  7.7215e-01 -1.3346e+00  2e+00  2e-16  1e-14
26:  9.2438e-02 -3.7598e-01  5e-01  2e-16  5e-15
27:  6.8763e-02 -3.9050e-01  5e-01  3e-16  6e-15
28: -1.4548e-01 -5.2888e-01  4e-01  3e-16  1e-14
29: -2.4999e-01 -3.9024e-01  1e-01  2e-16  2e-14
30: -3.0840e-01 -3.1053e-01  2e-03  2e-16  2e-14
31: -3.0917e-01 -3.0919e-01  2e-05  3e-16  2e-14
32: -3.0918e-01 -3.0918e-01  2e-07  2e-16  2e-14
Optimal solution found.
```

[32]:
```
alpha=np.array(sol['x'])
print(alpha[:10])
```

```
[[2.93450435e-11]
 [3.98935068e-11]
 [2.95256615e-11]
 [1.05773700e-10]
 [4.76680981e-11]
 [3.94055540e-11]
 [2.35817106e-11]
 [2.64523315e-11]
 [3.07636988e-11]
 [2.58830284e-11]]
```

[33]:
```
w=np.sum((alpha*y)*X,axis=0)
print(w)
```

```
[-0.68669636  0.38315689]
```

[34]:
```
X_r=X[(y==1)[:,0]]
```

[35]:
```
b=np.max(1-np.dot(X_r,w))
print(b)
```

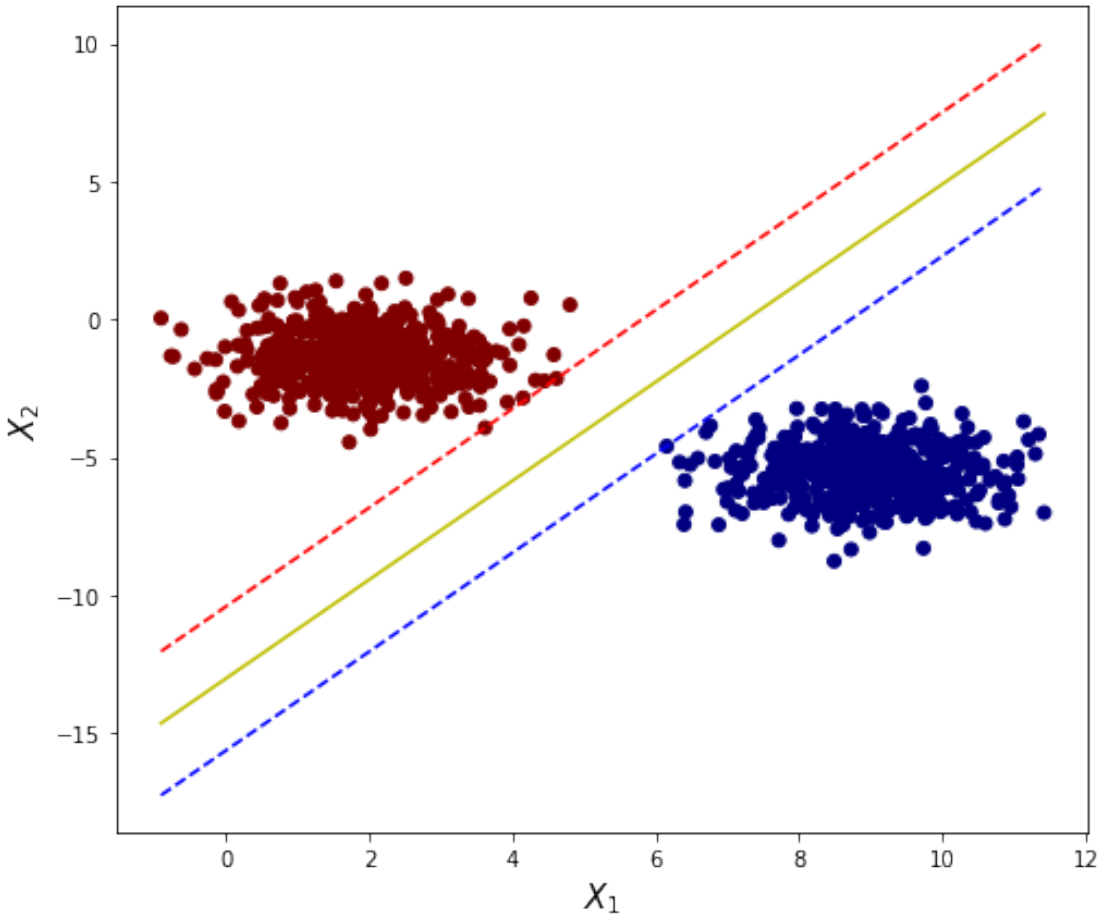
```
4.983377686567152
```

Let's visualize the result

[36]:
```
plt.figure(figsize=(8,7))
plt.scatter(X[:,0],X[:,1],c=y[:,0],cmap='jet')
plt.xlabel('$X_1$',fontsize=15)
plt.ylabel('$X_2$',fontsize=15)


x1_=np.linspace(np.min(X[:,0]),np.max(X[:,0]))
plt.plot(x1_,(-w[0]*x1_-b+1)/w[1],'r--')
plt.plot(x1_,(-w[0]*x1_-b-1)/w[1],'b--')
plt.plot(x1_,(-w[0]*x1_-b)/w[1],'y')

plt.show()
```


10
5
0
−5
−10
−15
X_2
0
2
4
6
8
10
12
X_1