Machine Learning Assignment 7

Shivangi Aneja

11-December-2017

Problem 1

The constrained optimization problem :
minimize $f_0(x)$
subject to $f_i(x) \leq 0 \; i = 1, 2, ..., M$
with f_0 convex and $f_1....f_M$ convex can be solved

Given, minimize $f_0(x) = -(x_1 + x_2)$
subject to $f_1(x) = x_1^2 + x_2^2 - 1 \leq 0$

Step 1: Calculate the Lagrangian
$L(x, \alpha) = f_0(x) + \sum_{i=1}^{M} \alpha_i f_i(x)$
$L(x, \alpha) = -(x_1 + x_2) + \alpha_1(x_1^2 + x_2^2 - 1)$

Step 2: Obtain the Lagrange Dual Function $g(\alpha)$
$x^* = arg\, min_x L(x, \alpha)$
$\Rightarrow \nabla_x L(x, \alpha) = 0$
$\Rightarrow \nabla_{x_1} L(x, \alpha) = 0 \Rightarrow -1 + \alpha_1(2x_1) = 0 \Rightarrow \boxed{x_1 = \frac{1}{2\alpha_1}}$
$\Rightarrow \nabla_{x_2} L(x, \alpha) = 0 \Rightarrow -1 + \alpha_1(2x_2) = 0 \Rightarrow \boxed{x_2 = \frac{1}{2\alpha_1}}$

Step 3: Sove the dual problem
maximize $g(\alpha) = L(x^*, \alpha)$ with $\alpha_1 \geq 0$
$\Rightarrow g(\alpha) = \frac{-1}{2\alpha_1} - \alpha_1$
$\nabla_\alpha g(\alpha) = 0 \Rightarrow \frac{1}{2\alpha_1^2} - 1 = 0 \Rightarrow \boxed{\alpha_1 = \frac{1}{\sqrt{2}}}$

Substituting value of α_1 we get , $\boxed{x_1 = \frac{1}{\sqrt{2}}, x_2 = \frac{1}{\sqrt{2}}}$

Problem 2

Similarity between SVM and Perceptron Algorithm
Both SVM and Perceptron Algorithm are used for classification of data

Difference :

Support Vector Machine	Perceptron Algorithm
SVM is a quadratic program that maximizes the margin (the sum of the squared distance of each point from the hyperplane) under the constraint that the hyperplane separates the points into two classes.	The perceptron algorithms finds a line that separates the points by class (provided such a line exists).Typically there will be more than one such separating line, and the exact line obtained through a run of the perceptron algorithm depends on the order points are processed.
SVM needs all the training data and only then starts building the classifier.	Perceptron is an online algorithm which means it can processes the data points one by one
Since SVM looks at maximizing the margin it allows you to find the most optimal solution. Thus, SVM can help you classify the test data in a better way as a large margin can help you segregate it better.	the perceptron algorithm tries to reduce error and thus , gives a good enough classification for the data points.
SVM finds the widest margin hyperplane	Perceptron finds some separating hyperplane
SVM finds the minimum cost separation. This is known as soft margin.	If the training set is not separable perceptron will find some hyperplane which tries to separate it as best as it can.

Problem 3

For SVM , we have

minimize $f_0(w,b) = \frac{1}{2}w^T w$

subject to $f_i(w,b) = y_i(w^T x_i + b) - 1 \geq 0$ for $i = 1, ..., N$

Let $\boldsymbol{x^*}$ and $\boldsymbol{\alpha^*}$ be the optimal solution for the constrained convex optimization problem of Support Vector Machine.

$\boldsymbol{x^* = [w^*}, b]$ and $\boldsymbol{\alpha^*} = [\alpha_1^*, ...\alpha_N^*]$

$\boxed{p^* = f_0(x^*) \text{ and } d^* = f_0(x^*) + \sum_{i=1}^{N} \alpha_i^* f_i(x^*)}$ [As $[x^*, \alpha^*]$is the optimal solution]

Since $g(\alpha)$ is a lower bound on p^* , we have weak duality ,

$\Rightarrow d^* \leq p^*$........(1)

Substituting these values, we have

$\Rightarrow f_0(x^*) + \sum_{i=1}^{N} \alpha_i^* f_i(x^*) \leq f_0(x^*)$........(2)

But $\boxed{\alpha_i^* f_i(x^*) = 0}$ as this Complementary Slackness for KKT condition. Substitute this in (2),we have

$\Rightarrow f_0(x^*) + 0 \leq f_0(x^*)$........(3)

But these values are equal

$\Rightarrow f_0(x^*) + 0 = f_0(x^*)$........(4)

Hence

$\boxed{\Rightarrow d^* = p^*}$

Thus strong duality holds for SVM or duality gap is zero.

Problem 4

The Dual problem is as :

maximize $g(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j$..........(1)

subject to $\sum_{i=1}^{N} \alpha_i y_i = 0$

$\alpha_i \geq 0$, for $i = 1, ..., N$

Rewriting this dual problem as :

$g(\alpha) = \frac{1}{2}\alpha^T Q \alpha + \alpha^T 1_N$...........(2)

(a) Let the number of dimensions of X be D.

$\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j x_i^T x_j \Rightarrow \sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \sum_{a=1}^{D} x_{i,a}^T x_{j,a} \Rightarrow \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{a=1}^{D} y_i x_{i,a} y_j x_{j,a} \Rightarrow \sum_{a=1}^{D}\sum_{i=1}^{N} y_i x_{i,a} \sum_{j=1}^{N} y_j x_{j,a} \Rightarrow$
$\sum_{a=1}^{D}(\sum_{i=1}^{N} y_i x_{i,a})^2$

$$\boxed{\sum_{a=1}^{D}(\sum_{i=1}^{N} y_i x_{i,a})^2 = (y \odot x)(y \odot x)^T}$$

Comparing this with Eq(2), we have $\boxed{Q = -(y \odot x)(y \odot x)^T}$

(b) $Q = -(y \odot x)(y \odot x)^T$

To show that Q is negative semi-definite, show that for a vector z , $z^T Q z \leq 0$

$\sum_{i=1}^{N}\sum_{j=1}^{N} z_i z_j y_i y_j \sum_{a=1}^{D} x_{i,a}^T x_{j,a} \Rightarrow \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{a=1}^{D} z_i z_j y_i y_j x_{i,a}^T x_{j,a} \Rightarrow \sum_{a=1}^{D}\sum_{i=1}^{N} z_i y_i x_{i,a} \sum_{j=1}^{N} z_j y_j x_{j,a}$

$\Rightarrow \boxed{\sum_{a=1}^{D}(\sum_{i=1}^{N} z_i y_i x_{i,a})^2 \geq 0}$ This is a Positive Semi definite Matrix

Taking into account $-$ sign in the original Q matrix $\Rightarrow z^T Q z \leq 0 \Rightarrow$ Q is negative Semi definite

(c) Negative Semi Definite means that $g(\alpha) \leq f_0(x)$. Thus $g(\alpha)$ is bounded above. This property is useful because it ensures that optimization problem is well defined.

Problem 5

Python file is attached on the next page

December 10, 2017

1 Programming assignment 7: SVM

```
In [14]: import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline

         from sklearn.datasets import make_blobs

         from cvxopt import matrix, solvers
```

1.1 Your task

In this sheet we will implement a simple binary SVM classifier.

We will use `CVXOPT` http://cvxopt.org/ - a Python library for convex optimization. If you use `Anaconda`, you can install it using

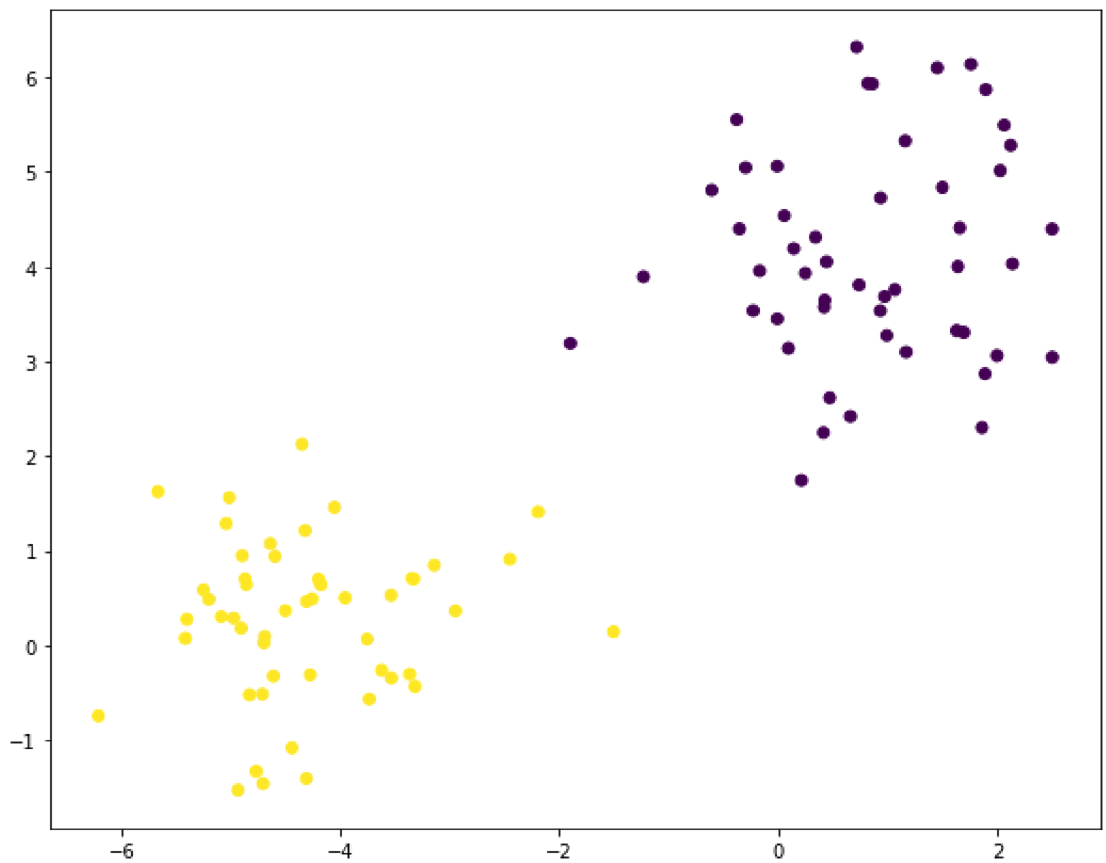
```
conda install cvxopt
```

As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it you your HW solution.

1.2 Generate and visualize the data

```
In [15]: N = 100  # number of samples
         D = 2  # number of dimensions
         C = 2  # number of classes
         seed = 3  # for reproducible experiments

         X, y = make_blobs(n_samples=N, n_features=D, centers=2, random_state=seed)
         y[y == 0] = -1  # it is more convenient to have {-1, 1} as class labels (instead of {0,
         y = y.astype(np.float)
         plt.figure(figsize=[10, 8])
         plt.scatter(X[:, 0], X[:, 1], c=y)
         plt.show()
```



1.3 Task 1: Solving the SVM dual problem

Remember, that the SVM dual problem can be formulated as a Quadratic programming (QP) problem. We will solve it using a QP solver from the `CVXOPT` library.

The general form of a QP is

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

$$\text{subject to} \quad \mathbf{G}\mathbf{x} \preceq \mathbf{h}$$

$$\text{and} \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\preceq$ denotes "elementwise less than or equal to".

Your task is to formulate the SVM dual problems as a QP and solve it using `CVXOPT`, i.e. specify the matrices $\mathbf{P}, \mathbf{G}, \mathbf{A}$ and vectors $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

```
In [16]: def solve_dual_svm(X, y):
             """Solve the dual formulation of the SVM problem.

             Parameters
```

```
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    alphas : array, shape [N]
        Solution of the dual problem.
    """
    # TODO
    # These variables have to be of type cvxopt.matrix
    NUM = X.shape[0]
    DIM = X.shape[1]
    K = y[:, None] * X
    K = np.dot(K, K.T)
    P = matrix(K)
    q = matrix(-np.ones((NUM, 1)))
    G = matrix(-np.eye(NUM))
    h = matrix(np.zeros(NUM))
    A = matrix(y.reshape(1, -1))
    b = matrix(np.zeros(1))
    solvers.options['show_progress'] = False
    solution = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solution['x'])
    return alphas
```

1.4 Task 2: Recovering the weights and the bias

```
In [17]: def compute_weights_and_bias(alphas, X, y):
    """Recover the weights w and the bias b using the dual solution alpha.

    Parameters
    ----------
    alpha : array, shape [N]
        Solution of the dual problem.
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
```

```
    """
    # get weights
    w = np.sum(alphas * y[:, None] * X, axis = 0)
    # get bias
    support_vec = (alphas > 1e-4).reshape(-1)
    b =   y[support_vec] - np.dot(X[support_vec], w)
    bias = b[0]
    return w, bias
```

1.5 Visualize the result (nothing to do here)

```
In [18]: def plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b):
             """Plot the data as a scatter plot together with the separating hyperplane.

             Parameters
             ----------
             X : array, shape [N, D]
                 Input features.
             y : array, shape [N]
                 Binary class labels (in {-1, 1} format).
             alpha : array, shape [N]
                 Solution of the dual problem.
             w : array, shape [D]
                 Weight vector.
             b : float
                 Bias term.
             """
             plt.figure(figsize=[10, 8])
             # Plot the hyperplane
             slope = -w[0] / w[1]
             intercept = -b / w[1]
             x = np.linspace(X[:, 0].min(), X[:, 0].max())
             plt.plot(x, x * slope + intercept, 'k-', label='decision boundary')
             # Plot all the datapoints
             plt.scatter(X[:, 0], X[:, 1], c=y)
             # Mark the support vectors
             support_vecs = (alpha > 1e-4).reshape(-1)
             plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs], s=250, marke
             plt.xlabel('$x_1$')
             plt.ylabel('$x_2$')
             plt.legend(loc='upper left')
```

The reference solution is

```
w = array([[-0.69192638],
           [-1.00973312]])

b = 0.907667782
```

Indices of the support vectors are

```
[38, 47, 92]
```

```
In [19]: alpha = solve_dual_svm(X, y)
         w, b = compute_weights_and_bias(alpha, X, y)
         plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
         plt.show()
```