Machine Learning: Homework #7

Due on December 11, 2017 at 09:59am

Professor Dr. Stephan Guennemann

Marc Meissner - 03691673

Problem 1

Solve the following constrained optimization problem using the recipe described in the lecture (slide 31).

minimize $f_0(x) = (x_1 + x_2)$
subject to $f_1(x) = {x_1}^2 + {x_2}^2 1 \leq 0$

Solution
Following the "recipe" from slide 31:

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

$\nabla_x L(x, \alpha) = \begin{pmatrix} -1 + 2\alpha_1 x_1 \\ -1 + 2\alpha_1 x_2 \end{pmatrix} = 0$

$x_1^* = \frac{1}{2\alpha_1}$, $x_2^* = \frac{1}{2\alpha_1}$

$L(x^*, \alpha) = -\frac{1}{\alpha_1} + \alpha_1(\frac{1}{2\alpha_1^2} - 1) = -\frac{1}{2\alpha_1} - \alpha_1$

$\frac{\partial g(\alpha)}{\partial \alpha} = \frac{1}{2\alpha_1^2} - 1 = 0$

$\alpha_1 = \frac{1}{\sqrt{2}}$ (due to $\alpha_i \geq 0$) , $x_1^* = x_2^* = \frac{\sqrt{2}}{2}$

Problem 2

Explain the similarities and differences between the SVM and perceptron algorithms.

Solution
Both approaches try to find a hyperplane that separates the two sets. On the one hand, perceptrons are satisfied after finding any hyperplane, which may or may not be optimal in any given sense. They are also closely related to their underlying training algorithm, which allows for online learning with data coming in over time.
SVM's on the other hand seek to maximize the margine between the opposing support vectors. They need the whole data "offline" to find an optimal solution.

Problem 3

Show that the duality gap is zero for SVM.

Solution

According to Slater's constraint qualification, two conditions have to hold for the duality gap to be zero:

1. $f_i(x)$ have to be convex
2. there exists an x such that $f_i(x) < 0$ for $i = 1, ..., M$
OR $f_i(x) = w_i^T x + b \leq 0$, $\forall i$

$f_0(w, b) = \frac{1}{2} w^t w$ is a (positive) quadratic function and thus convex.
$f_i(w, b) = y_i(w^T x_i + b) - 1 \geq 0$, $\forall i \in 1, ..., N$ is convex, as it is a linear function.

Problem 4

Recall, that the dual function for SVM (slide 37) can be written as:

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

(a) Show how the matrix Q can be computed. (Hint: You might want to use Hadamard product, denoted as $\odot$).

(b) Prove that the matrix Q is negative (semi-)definite.

(c) Explain, what the negative (semi-)definiteness means for our optimization problem. Why is this property important?

Solution

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

$$Q = \begin{pmatrix} -y_1 y_1 x_1^T x_1 & ... & -y_1 y_N x_1^T x_N \\ ... & ... & ... \\ -y_N y_1 x_N^T x_1 & ... & -y_N y_N x_N^T x_N \end{pmatrix} = -(yy^T \odot x^T x)$$

(b)

(c) In order to use semidefinite programming - which allows an efficient solution for the convex optimization problem - Q must be positive or negative semidefinite.

Problem 5

Load the notebook 07_*homework_svm.ipynb* from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Solution

The notebook pdf is added.

07_homework_svm

December 8, 2017

1 Programming assignment 7: SVM

```
In [1]: 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

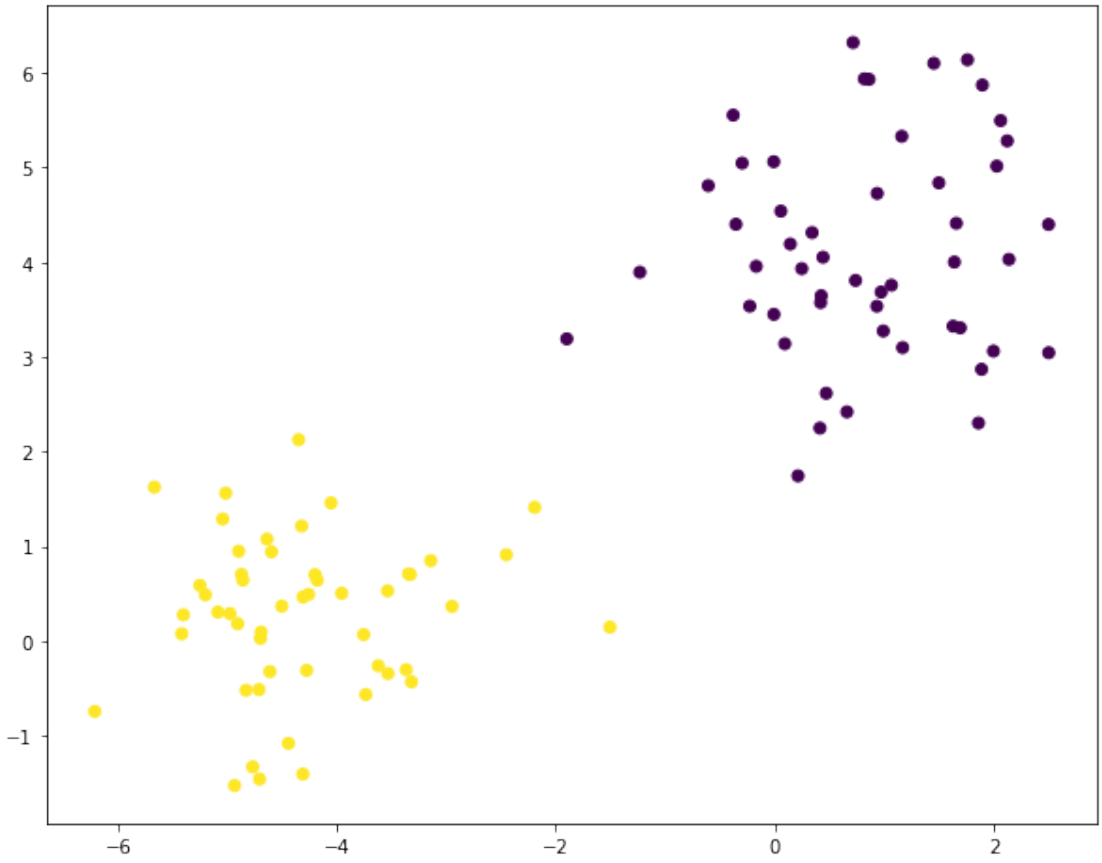
```
conda install cvxopt
```

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 [2]: 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 [3]: 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
    N = len(y)
    D = y[:, None] * X
    D = np.dot(D, D.T)
    P = matrix(D)
    q = matrix(-np.ones((N, 1)))
    G = matrix(-np.eye(N))
    h = matrix(np.zeros(N))
    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 [4]: def compute_weights_and_bias(alpha, 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.
    """
```

```
    N = len(y)
    w = 0
    for i in range(N):
        w += alpha[i] * y[i] * X[i, :]

    ind = (alpha > 1e-10).reshape(-1)
    b = np.sum(y[ind] - np.dot(X[ind,:], w))/len(ind)
    return w, b
```

1.5 Visualize the result (nothing to do here)

```
In [5]: 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, marker
    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 [6]: 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()
```