Machine Learning Worksheet Solution 7

Julius Jankowski

1 Constrained Optimization

Problem 1:

Solving the following constrained optimization problem:

$$f_0(\mathbf{x}) = -(x_1 + x_2) \tag{1.1}$$
$$f_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 \tag{1.2}$$

according to the recipe of the lecture. Calculate the Lagrange:

$$L(\mathbf{x}, \alpha) = f_0(\mathbf{x}) + \alpha f_1(\mathbf{x}) \tag{1.3}$$

Obtain the Lagrange Dual function $g(\alpha)$:

$$\nabla_x L = [2\alpha x_1 - 1, 2\alpha x_2 - 1]^T = \mathbf{0} \tag{1.4}$$
$$\implies x_1^* = x_2^* = \frac{1}{2\alpha} \tag{1.5}$$
$$g(\alpha) = L(\mathbf{x}^*, \alpha) = -(\alpha + \frac{1}{2\alpha}) \tag{1.6}$$

and solve the dual problem:

$$\frac{dg}{d\alpha} = -1 + \frac{1}{2\alpha^2} = 0 \implies \alpha = \pm\sqrt{\frac{1}{2}} \tag{1.7}$$
$$\implies x_1^* = x_2^* = \frac{\sqrt{2}}{2} \tag{1.8}$$

2 SVM

Problem 2:

The basic idea of both approaches is to find a flat hyperplane which seperates two different sets of samples in the feature space. They differ in the way of finding a solution. While the perceptron learning is trying to minimize the error of all samples of the data set, the SVM learning bases its solution on the samples which are closest to another class. It tries to maximize the margin between two classes.

Problem 3:

The SVM constrained optimization problem can be formulated as follows:

$$f_0(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{2.1}$$

$$f_i(\mathbf{w}, b) = 1 - y_i \cdot (\mathbf{w}^T\mathbf{x}_i + b) \leq 0 \tag{2.2}$$

According to Slater's constraint qualification the dual problem has a duality gap of zero when the main constraint and the side constraints are convex and there exists a set of parameters $\mathbf{w}$ for which the side constraints are always negative.

Since the constraints are quadratic and linear, they are indeed convex. The second qualification criterion is actually a reformulation of the condition for linear seperability of two data sets. Thus if the data sets are linearly seperable, the criterion is satisfied and only then the duality gap is zero for SVM.

Problem 4:

(a)
In order to compute Q based on the dual function of SVM, we can basically compare the two equations:

$$-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T\mathbf{x}_j = \frac{1}{2}\alpha^T Q \alpha = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j q_{ij} \tag{2.3}$$

$$\implies q_{ij} = -y_i y_j \mathbf{x}_i^T\mathbf{x}_j \tag{2.4}$$

Or expressed by the Hadamard product: $Q = -\mathbf{y}\mathbf{y}^T \odot X^T X$.

(b)
Due to Schur's product theorem, the result of a hadamard product is positive semidefinite if the two multiplied matrices are positive semidefinite. Hence in order to prove that Q is negative semidefinite, we can prove that $\mathbf{y}\mathbf{y}^T \odot X^T X$ is positive semidefinite which means that we have to prove that $\mathbf{y}\mathbf{y}^T$ and $X^T X$ are positive semidefinite.

For $\mathbf{y}\mathbf{y}^T$:

$$\alpha^T \mathbf{y}\mathbf{y}^T \alpha = (\alpha^T \mathbf{y})^2 \geq 0 \tag{2.5}$$

For $X^T X$

$$\alpha^T X^T X \alpha = (X\alpha)^T X\alpha \geq 0 \tag{2.6}$$

(c)

07_homework_svm

December 10, 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

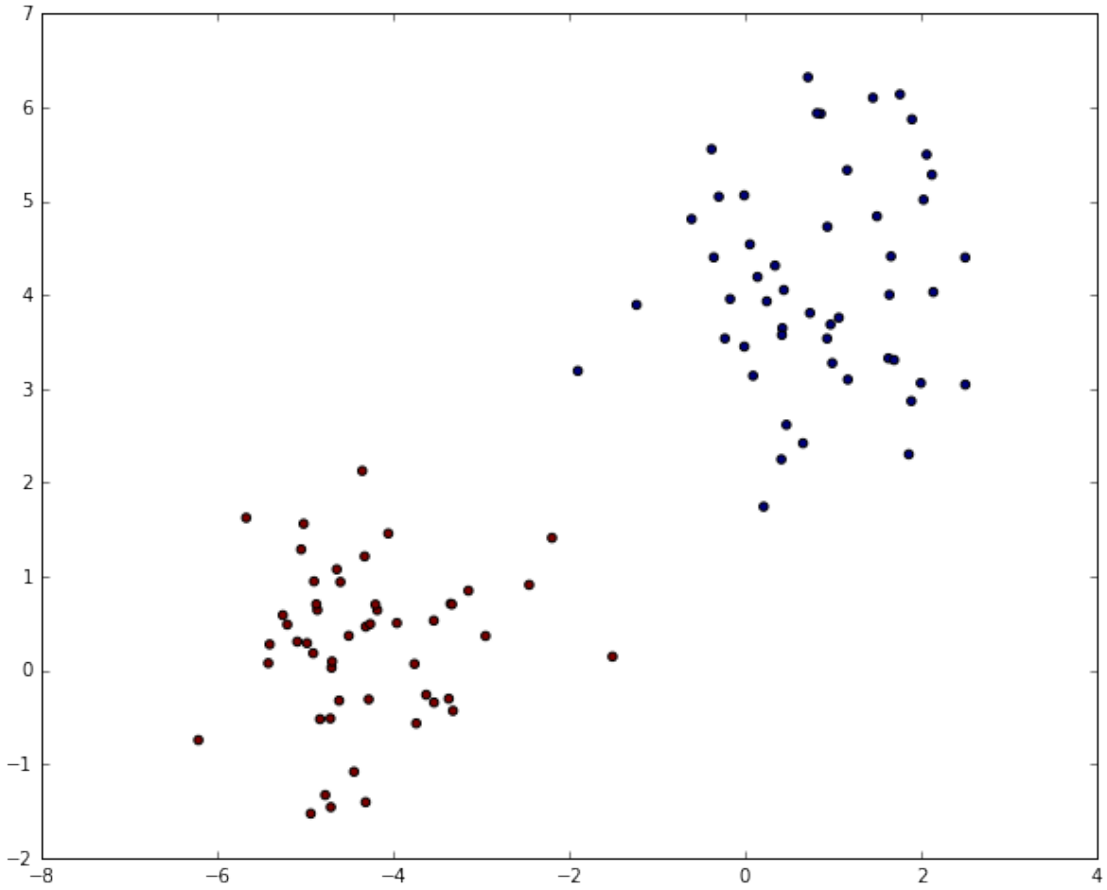
```
conda install cvxopt
```

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 [30]: 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
    N = X.shape[0]
    Q = np.multiply(np.outer(y,y), np.inner(X,X)) # hadamard product

    yT = np.zeros([1,N])
    for i in range(N): # transposing the vector
        yT[0,i] = y[i]

    # These variables have to be of type cvxopt.matrix
    P = matrix(Q)
    q = matrix(-np.ones([N,1]))
    G = matrix(-np.identity(N))
    h = matrix(np.zeros([N,1]))
    A = matrix(yT)
    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 [25]: 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 = X.shape[0]
    D = X.shape[1]

    w = 0
    b = 0
    count = 0

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

    for i in range(N):
        if alpha[i] > 1e-6:
            b = b + y[i] - np.dot(w,X[i,:])
            count = count + 1

    b = b / count

    return w, b
```

1.5 Visualize the result (nothing to do here)

```
In [6]: 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

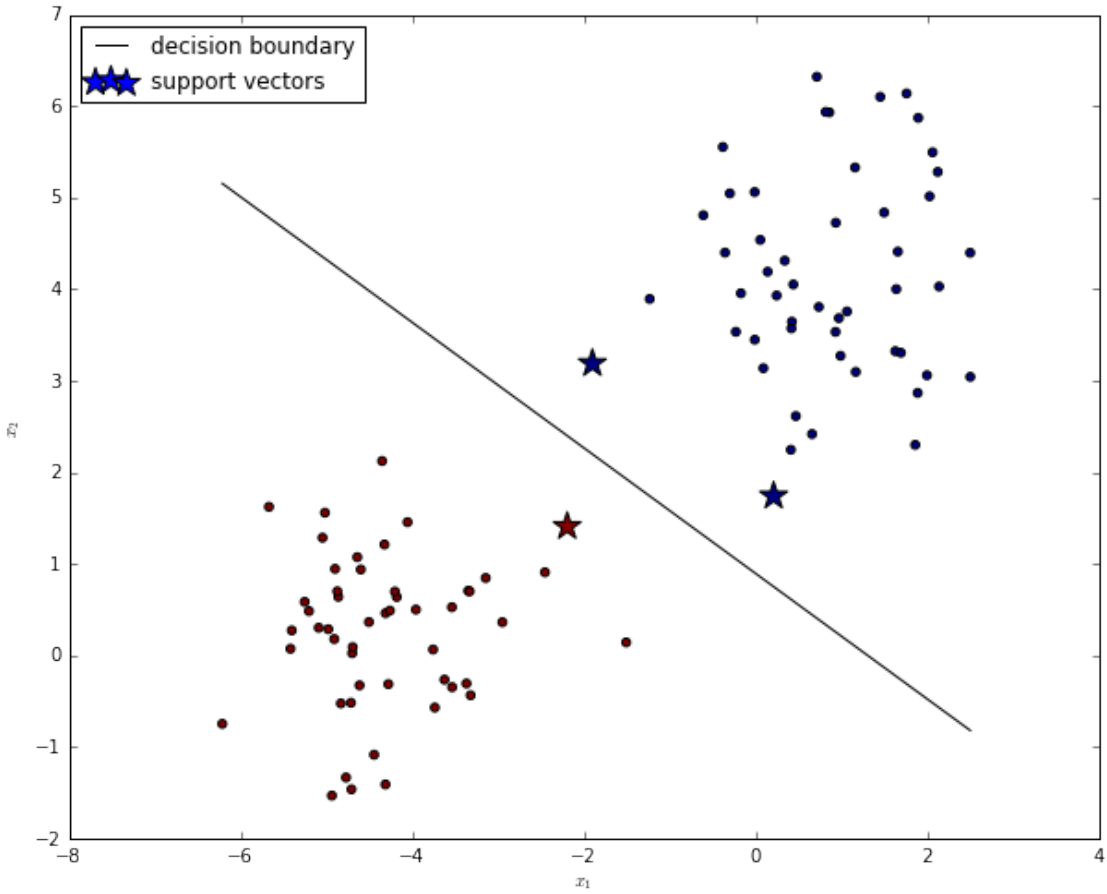
```
w = array([[-0.69192638],
           [-1.00973312]])

b = 0.907667782
```

Indices of the support vectors are

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

```
In [28]: 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()
```


decision boundary
support vectors
7
6
5
4
3
2
1
0
−1
−2
−8
−6
−4
−2
0
2
4
x_1
x_2