Constrained Optimization and SVM

Machine Learning - Prof. Dr. Stephan Günnemann

Leonardo Freiherr von Lerchenfeld

December 10, 2017

Contents

1 Constrained Optimization

1.1 Problem 1

Calculate the Lagrangian

$$\begin{aligned} L(x,\alpha) &= f_0(x) + \sum_{i=1}^{M} \alpha_i f_i(x) && (1)\\ &= -x_1 - x_2 + \alpha_1 x_1^2 + \alpha_1 x_2^2 - \alpha_1 && (2)\\ \frac{\partial L}{\partial x_1} = 0 &= -1 + 2\alpha_1 x_1 && (3)\\ \frac{\partial L}{\partial x_2} = 0 &= -1 + 2\alpha_1 x_2 && (4) \end{aligned}$$

Complementary Slackness

$$\begin{aligned} \alpha_1(x_1^2 + x_2^2 - 1) &= 0 && (5)\\ assume\ \alpha_1 &> 0 && (6)\\ x_1^2 &= 1 - x_2^2 && (7) \end{aligned}$$

$(3)^2 = (8)$ and $(7)\ in\ (8) = (9)$

$$\begin{aligned} 4\alpha_1^2 x_1^2 &= 1 && (8)\\ 4\alpha_1^2(1 - x_2^2) &= 1 && (9)\\ \alpha_1^2 &= \frac{1}{4}(1 - x_2^2)^{-1} && (10) \end{aligned}$$

$(4)^2 = (11)$ and $(10)\ in\ (11) = (12)$

$$\begin{aligned} 4\alpha_1^2 x_2^2 &= 1 && (11)\\ \frac{x_2^2}{1 - x_2^2} &= 1 && (12)\\ x_2 &= \sqrt{\frac{1}{2}} && (13)\\ x_1^2 &= 0.5 && (14)\\ \alpha_1 &= \sqrt{\frac{1}{2}} && (15) \end{aligned}$$

2 Support Vector Machine (SVM)

2.1 Problem 2

Both, the SVM and the perceptron want to classify a point x into on of two classes, for example, into class blue and green. The perceptron uses this to build a linear classifier by assigning all x with

$$w^T x + b > 0$$

to class blue and all x with

$$w^T x + b < 0$$

to class green. Thus, perceptrons are satisfied with any (not optimal) hyperplane. The SVM is a linear classifier with a large margin. Hence, we now require

$$w^T x + (b - s) > 0$$

for all x from class blue and

$$w^T x + (b + s) < 0$$

for all x from class green. The larger margin results in a better generalization.

2.2 Problem 3

Slater's constraint qualification
If $f_0, f_1, ..., f_M$ are convex and there exists an $x \in \mathbb{R}^D$ such that

$$f_i(x) < 0,\ \ i = 1, ..., M$$

or the constraints are affine, that is

$$f_i(x) = w_i^T x + b_i \leq 0,$$

the duality gap is zero.

2.3 Problem 4

(a) The dual function for SVM can be written as
$g(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i y_j x_i^T x_j \alpha_j$
Note that, α and y are vectors, while X is a matrix.
$-(yy^T \odot X^T X) = \begin{pmatrix} -y_1 y_1 x_1^T x_1 & \cdots & -y_1 y_N x_1^T x_N \\ \vdots & \ddots & \vdots \\ -y_N y_1 x_N^T x_1 & \cdots & -y_N y_N x_N^T x_N \end{pmatrix} = Q$
$g(\alpha) = \alpha^T 1_N + \frac{1}{2} \alpha^T Q \alpha$

(b) A matrix A is positiv semi-definite, if $v^T A v \geq 0$ for all $v \in \mathbb{R}^n \backslash \{0\}$.

$$\begin{aligned} v^T y y^T v = (v^T y)^2 &\geq 0 \\ v^T X^T X v = (Xv)^T X v &\geq 0 \end{aligned}$$

The Hadamard product of two positive semi-definite matrices is positive semi-definite. This is known as the **Schur product theorem**, after German mathematician Issai Schur. Hence, $(yy^T \odot X^T X)$ is positive semi-definite and $Q = -(yy^T \odot X^T X)$ is negative semi-definite.

(c) For our optimization problem, we can use an efficient algorithm like semidefinite programming. Therefore, Q must be definite.

2.4 Problem 5

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

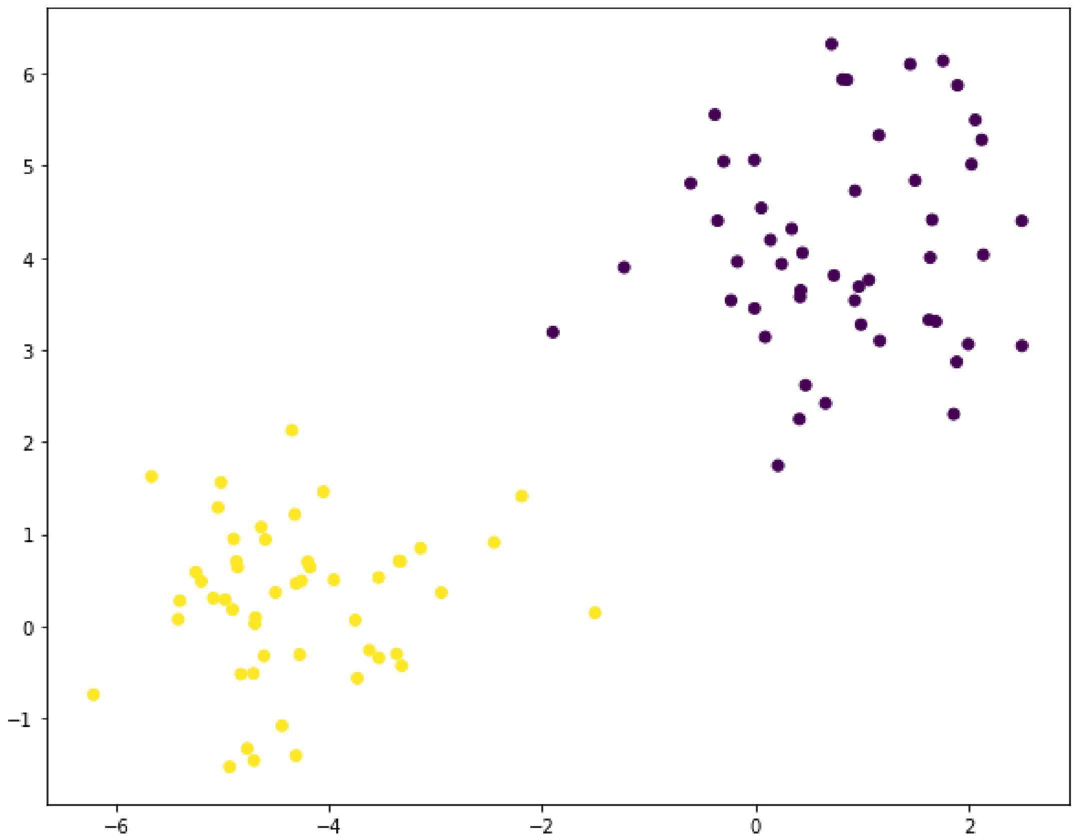
```
conda install cvxopt
```

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 (i
        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)
    P = matrix(...)
    q = matrix(-np.ones(N,1))
    G = matrix(-np.eye(N))
    h = matrix(np.zeros(N))
    A = matrix(...)
    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.
            """
            w = None
            b = None
```

```
        return w, b
```

1.5 Visualize the result (nothing to do here)

```
In [32]: def plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b):
             """Plot the data as a scatter plot together with the separating hyperp

             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],
             plt.xlabel('$x_1$')
             plt.ylabel('$x_2$')
             plt.legend(loc='upper left')
```

The reference solution is

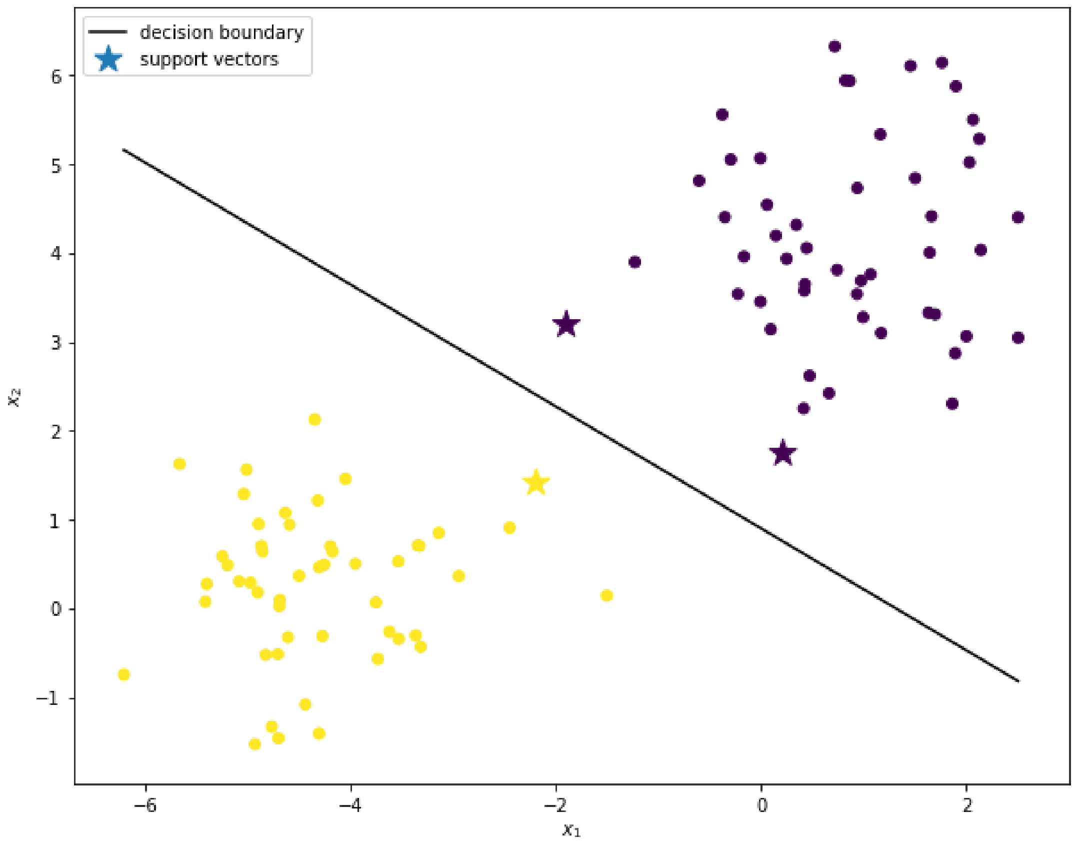
```
w = array([[-0.69192638],
           [-1.00973312]])

b = 0.907667782
```

Indices of the support vectors are

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

```
In [33]: 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
6
5
4
3
2
1
0
−1
x_2
−6
−4
−2
0
2
x_1