homework_06_notebook

December 1, 2018

1 Programming assignment 6: SVM

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        from sklearn.datasets import make_blobs

        from cvxopt import matrix, solvers
```

1.1 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Download the notebook in HTML (click File > Download as > .html) 3. Convert the HTML to PDF using e.g. https://www.sejda.com/html-to-pdf or `wkhtmltopdf` for Linux (tutorial) 4. Concatenate your solutions for other tasks with the output of Step 3. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

This way is preferred to using `nbconvert`, since `nbconvert` clips lines that exceed page width and makes your code harder to grade.

1.2 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
```

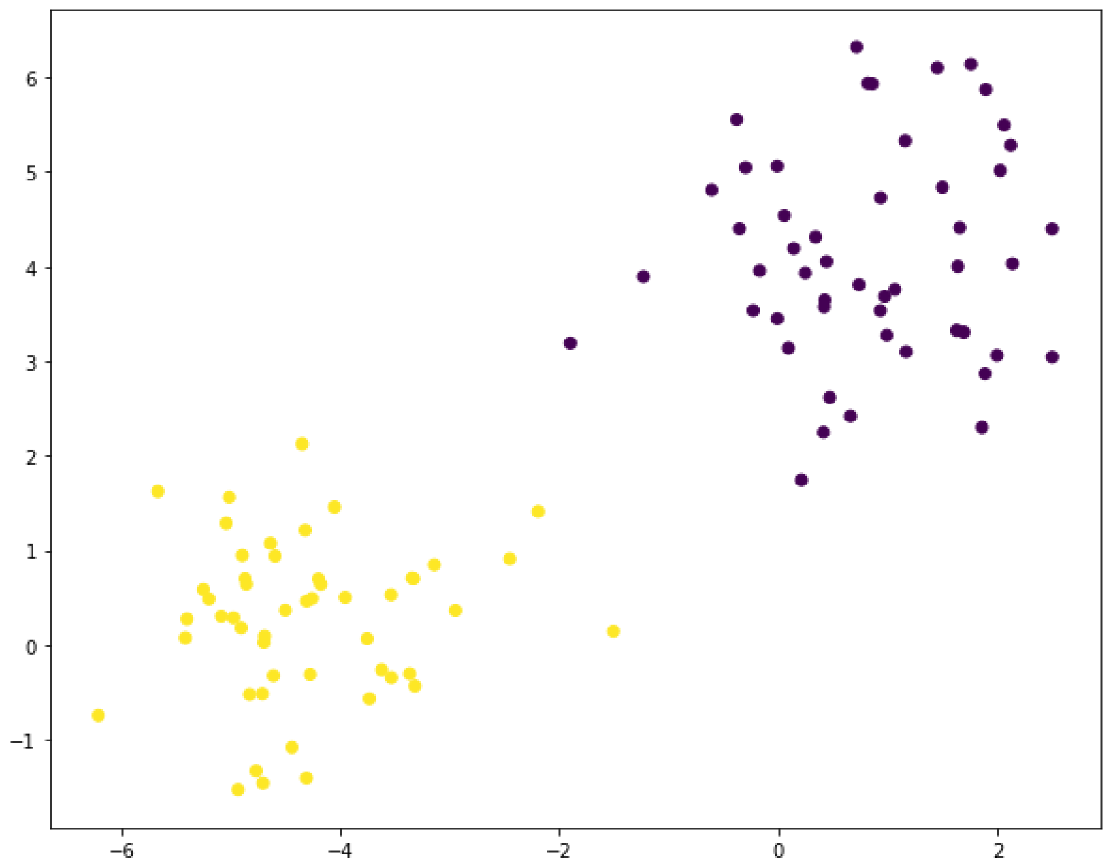
As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it you your HW solution.

1.3 Generate and visualize the data

```
In [8]: 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()
```



1.4 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 [9]: 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
            P = matrix(np.multiply(np.dot(y[:,None],np.transpose(y[:,None])),
                                   np.dot(X,np.transpose(X))))
            q = matrix(-1*np.ones((N,1)))
            G = matrix(np.diag(-1*np.ones(y.shape[0])))
            h = matrix(np.zeros(y.shape[0]))
            A = matrix(np.transpose(y[:,None]))
            b = matrix([0.0])
            solvers.options['show_progress'] = False
            solution = solvers.qp(P, q, G, h, A, b)
            alphas = np.array(solution['x'])
            return alphas
```

1.5 Task 2: Recovering the weights and the bias

```
In [10]: 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 = np.dot(np.transpose(X),np.multiply(alpha,y[:,None]))
    d=np.nonzero(alpha > 1e-4)
    d=np.array((d[0],d[1]))
    d=np.transpose(d)
    b=np.sum(np.subtract(y[d[:,0],None],np.dot(X[d[:,0],:],w)))/d.shape[0]
    return w, b
```

1.6 Visualize the result (nothing to do here)

```
In [11]: 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 [12]: 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()
```