Emanuele Belli 03720425
Thomas Brunner 03675118
Luca Corbucci 03726968

Exercise Sheet 08

Machine Learning
December 8, 2019

.

Problem 1

- The perceptron algorithm finds an hyperplane that can correctly classify all the points in the dataset but this is not the best hyperplane (the best hyperplane for generalization). SMV finds the best hyperplane maximizing the margin.
- With Perceptron we can obtain different hyperplanes depending on our starting point. SVM produces only one single hyperplane.
- The perceptron algorithm analyze each point of the dataset and for each of the points changes the hyperplane if it is necessary. SVM needs the entire dataset to find the best hyperplane.
- If the dataset is not linearly separable the perceptron algorithm will not find a solution for the problem while SVM will find an optimal hyperplane (using soft margin).
- If we look at the constrained optimization problem of the soft margin SVM and we consider the Hing loss formulation we can write the constrained optimization problem as an unconstrained optimization problem:

$$\min_{w,b} \ \frac{1}{2}w^T w + C\sum_{i=1}^{M} \max\{0, 1 - y_i(w^T x_i + b)\}$$

We can note that the second part of this minimization problem is the Hinge loss function

$$E_{hinge} = max\{0, 1 - y_i(w^T x_i + b)\}$$

This is equal to the Perceptron Hinge Loss function with $\epsilon = 1$:

$$E_{hinge} = max\{0, \epsilon - y_i(w^T x_i + b)\}$$

Problem 2

a) We define

$$Y = diagonal(y)$$

Y is the diagonal matrix with $y_1, ..., y_n$ on the diagonal and 0 elsewhere.

$$Q = -(YX \odot X^T Y)$$

b) We need to show that $g(\alpha)$ is concave:

$$Q \text{ must be negative definite} \Leftrightarrow -(-Q) \text{ must be positive definite}$$

Let $\alpha \in R^N$ 0.

$$a^R(-(-Q))a = a^T(diag(y)XX^Tdiag(y))a =$$
$$a^T(diag(y)x)(x^Tdiag(y))a =$$

We use $R = (diag(y)X)$:

$$a^TRR^Ta = (R^Ta)^T(R^Ta) = ||R^Ta||^2 > 0$$

Problem 3

$$\epsilon = \frac{\tilde{s}}{N} + \frac{\tilde{ns}}{N}$$

Where $\tilde{ns}$ is equal to the number of points which are not support vectors which were misclassified and $\tilde{s}$ is the support vector that were misclassified.

$$\implies \epsilon \leq \frac{S}{N} + \frac{\tilde{ns}}{N}$$

To prove the claim we must show that $\tilde{ns}$ is 0. This is clear because if x_i was not a support vector, leaving it in the training will not change the solution (because of the hard margin) $\implies$ it will again be classified correctly.

Problem 5

- $$k(x, y) = x_1^T x_2$$
  is a valid kernel, as scalar product in R^N;
- $$k(x_1, x_2) = (x_1^T x_2)^i$$
  with $i \in N$ is a valid kernel, as product of kernels;
- $\sum_{i=1}^{N} \alpha_i (x_1^T x_2)^i$ with $\alpha_i \geq 0$ is a valid kernel as positive comb of kernels
- $$k(x_1, x_2) = \sum_{i=1}^{N} a_i (x_1^T x_2)^i + a_0$$
  is a kernel as α_0 (w.l.o.g. $\alpha_0 > 0$) is a kernel.

α_0 is a kernel corresponding to $x- > \sqrt{\alpha_0}$ with the product in R.

Emanuele Belli 03720425
Thomas Brunner 03675118
Luca Corbucci 03726968

Exercise Sheet 08

Problem 6

Consider $\forall N \in$ the transformation $\Phi_N = (0,1) -> R^N N\ x -> (1, x, x^2, ..., x^N)$. Then consider as scalar product the classical scalar product such that:

$$K_N(x_1, x_2) = \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \\ .... \\ x_1^N \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_2^2 \\ .... \\ x_2^N \end{bmatrix} = \sum_{i=0}^{N} (x_1^T x_2)^i$$

This is a kernel.
Define $\Phi = \lim_{n->\infty} \Phi_N$

$$\Phi(x) = (1, x, x^2, ...)$$

Furthermore being $x \in (0,1)$ we can define:

$$K(x_1, x_2) = \lim_{N->\infty} \sum_{i=1}^{N} (x_1 x_2)^i$$

For the geometric series this is equal to:

$$\frac{1}{1 - x_1 x_2}$$

as the kernel of Φ. This is a kernel because limit of kernel is a kernel.

exercise_08_notebook1

December 8, 2019

1 Programming assignment 4: SVM

```
In [29]: 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. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

To solve optimization tasks 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
```

1.2 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. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. 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.

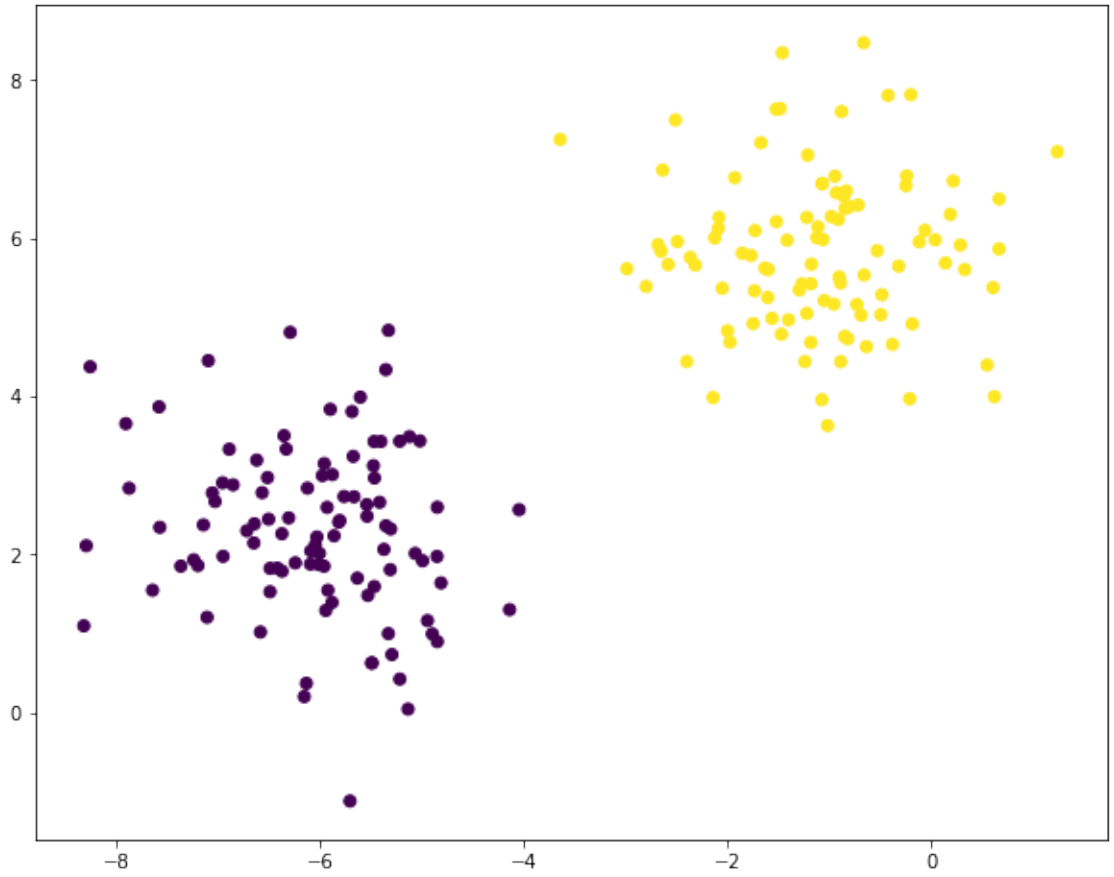
Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.3 Generate and visualize the data

```
In [30]: N = 200  # number of samples
         D = 2  # number of dimensions
         C = 2  # number of classes
         seed = 1234  # for reproducible experiments
```

```
alpha_tol = 1e-4 # threshold for choosing support vectors

X, y = make_blobs(n_samples=N, n_features=D, centers=C, random_state=seed)
y[y == 0] = -1  # it is more convenient to have {-1, 1} as class labels (instead of {
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.

We use the following form of a QP problem:

$$\text{minimize}_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x} \text{subject to} \quad \mathbf{G}\mathbf{x} \leq \mathbf{h} \text{ and } \mathbf{A}\mathbf{x} = \mathbf{b}\,.$$

Your task is to formulate the SVM dual problems as a QP of this form 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 [31]: 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)
             Y = np.diag(y)
             Q = Y.dot(X)
             P = matrix(Q.dot(Q.T))
             q = matrix(-np.ones(N))
             G = matrix(-np.identity(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.reshape(-1)
```

1.5 Task 2: Recovering the weights and the bias

```
In [32]: 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 = 0
    for i in range(len(alpha)):
        w += alpha[i]*y[i]*X[i,:]

    b_ = 0
    alpha_sum = 0
    for i in range(len(y)):
        if alpha[i] > alpha_tol:
            b_ += alpha[i]*(y[i] - np.dot(w.T, X[i,:]))
            alpha_sum += alpha[i]

    b = b_/alpha_sum
    return w, b
```

1.6 Visualize the result (nothing to do here)

```
In [33]: 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')
    plt.plot(x, x * slope + intercept - 1/w[1], 'k--')
    plt.plot(x, x * slope + intercept + 1/w[1], 'k--')
    # Plot all the datapoints
```

```
    plt.scatter(X[:, 0], X[:, 1], c=y)
    # Mark the support vectors
    support_vecs = (alpha > alpha_tol)
    plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs], s=250, mark
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.legend(loc='upper left')
```

The reference solution is

```
w = array([0.73935606 0.41780426])

b = 0.919937145
```

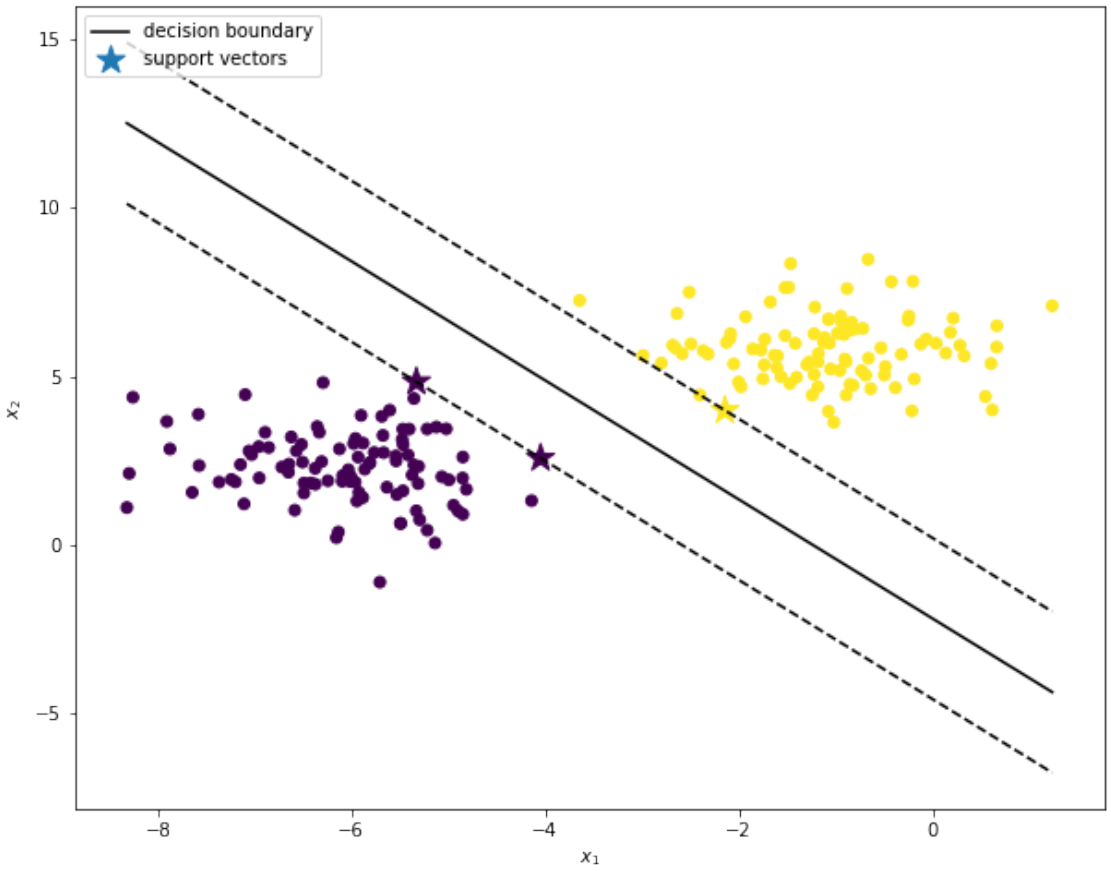
Indices of the support vectors are

```
[ 78 134 158]
```

```
In [34]: alpha = solve_dual_svm(X, y)
         w, b = compute_weights_and_bias(alpha, X, y)
         print("w =", w)
         print("b =", b)
         print("support vectors:", np.arange(len(alpha))[alpha > alpha_tol])
```

```
w = [0.73935606 0.41780426]
b = 0.9199371344144466
support vectors: [ 78 134 158]
```

```
In [35]: plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
         plt.show()
```



In [ ]:

In [ ]: