Exercise

08

TUM Department of Informatics

Supervised by Prof. Dr. Stephan Günnemann
Informatics 3 - Professorship of Data Mining and Analytics

Submitted by Marcel Bruckner (03674122)
Julian Hohenadel (03673879)
Kevin Bein (03707775)

Submission date Munich, December 8, 2019

SVM and Kernels

Problem 1:

Similarities:

Both try to find a fitting hyperplane which seperates the data classes.

Difference:

SVM tries to maximize the margin from the hyperplane to the data points, perceptron algorithms only care about a valid seperation of the data classes.

Problem 2:

a)

$g(\alpha)$ vectorized definition:

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

$g(\alpha)$ standard definition:

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

y is a vector of dimension $N \times 1$

x is a matrix of dimension $N \times M$

$\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j$ is equivalent to yy^T (dimension is $N \times N$)

$\sum_{i=1}^{N}\sum_{j=1}^{N} x_i^T x_j$ is equivalent to XX^T (dimension is $N \times N$)

$\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j x_i^T x_j$ is the Hadamard product so: $[yy^T \odot XX^T]$

Take the -1 scalar from the standard definition into the matrix: $[-yy^T \odot XX^T] = Q$

$\implies \frac{1}{2}\alpha^T Q\alpha \equiv \frac{1}{2}\alpha^T[-yy^T \odot XX^T]\alpha \equiv -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j$

$\alpha^T 1_N \equiv \sum_{i=1}^{N} \alpha_i$ is trivial.

$\implies g(\alpha)$ vectorized definition $\equiv g(\alpha)$ standard definition

b)

If the search for a local maximizer of g returns the global maximum of g, that means that the maximization problem is concave.

To proove this claim Q needs to be negativ (semi)definite (NSD).

For Q to be NSD: $\forall \alpha \in \mathbb{R}^N : \alpha^T Q\alpha \leq 0$ needs to hold.

$$\alpha^T Q \alpha \leq 0 \tag{1}$$

$$-\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j \leq 0 \tag{2}$$

$$-\sum_{i=1}^{N}\sum_{j=1}^{N} (y_i \alpha_i x_i)^T (y_j \alpha_j x_j) \leq 0 \tag{3}$$

$$-(y \odot \alpha)^T X X^T (y \odot \alpha) \leq 0 \tag{4}$$

$$-(y \odot \alpha)^T (X^T)^T X^T (y \odot \alpha) \leq 0 \tag{5}$$

$$-(X^T (y \odot \alpha))^T (X^T (y \odot \alpha)) \leq 0 \tag{6}$$

$$-(X^T (y \odot \alpha))^2 \leq 0 \tag{7}$$

$$-(\geq 0) \leq 0 \qquad \square \tag{8}$$

(3): y_i and α_i can be dragged inside here because they are scalars.

(4): The whole expression returns a scalar, that's why reshaping is done this way:

$dim((y \odot \alpha)^T) = 1 \times N$

$dim(X) = N \times 1$

$dim(X^T) = 1 \times N$

$dim((y \odot \alpha)) = N \times 1$

(5): $X = (X^T)^T$

(6): $(AB)^T = B^T A^T$

(7): $(...)^2 \geq 0$, (as long as ... is not complex)

(8): Proofs that Q is NSD.

Q is NSD, that means the maximization problem is in fact concave so local maxima = global maxima.

Problem 3:

ϵ is the LOOCV misclassification rate.

s is the number of support vectors.

N is the number of samples.

$$\epsilon \leq \frac{s}{N}$$

Case 1:

x_i from the current LOOCV is a support vector:

x_i is misclassified: $\implies$ α_i of x_i is 0 $\implies$ the misclassification of x_i will affect w^* and b^*.

$$\implies \epsilon_{\text{Case 1}} \leq \frac{s}{N}$$

Case 2:

x_i from the current LOOCV isn't a support vector:

Then x_i will have no effect on w^* and b^* anyway.

$\implies$ nothing will change.

$$\implies \epsilon_{\text{Case 2}} = 0$$

Both cases combined evaluate to the original inequality.

$$\epsilon_{\text{Case 1 and 2}} \leq \frac{s}{N} + 0$$
$$\epsilon \leq \frac{s}{N}$$

Problem 5:

"A kernel is valid if it corresponds to an inner product in some feature space."

$x_1^T x_2$ is a valid kernel because it is a scalar product of the input vectors.

a_0 is a constant term and a valid kernel because it can be generated by:

$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ with $\phi(x) = \sqrt{a_0} \implies k(x_i, x_j) = a_0$

The same holds for a_i so $a_i(x_i^T x_j)^i + a_0$ can be represented as kernels.

Sums and multiplications of kernels are kernel preserving (also scalars are ≥ 0)

$\implies$ Gram matrix is still PSD.

Problem 6:

The "trick" for this Problem:

"The Maclaurin series for $\frac{1}{1-x}$ is the geometric series: $1 + x + x^2 + x^3 + x^4$..."

$$\begin{aligned} k(x_1, x_2) &= \frac{1}{1 - x_1 x_2} \\ &= \sum_{i=0}^{\infty} x_1^i x_2^i \\ &= \phi(x_1)^T \phi(x_2) \\ \text{with } \phi(x) &= (1, x, x^2, x^3, x^4, ...) \end{aligned}$$

Problem 4:

homework_08_notebook

December 7, 2019

1 Programming assignment 4: SVM

[264]:
```
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

[265]:
```
N = 200  # number of samples
D = 2  # number of dimensions
C = 2  # number of classes
```

```
seed = 1234  # for reproducible experiments

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 {0, 1})
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 $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

[266]:
```
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

    # Store the shapes, always useful
    N, D = X.shape

    # reshape y from (200,) to (200,1)
    y = y.reshape(N,1)

    # yy^T
    y_help = y.dot(y.T)

    # XX^T
    X_help = X.dot(X.T)

    # Hadamard product
    P = matrix(np.multiply(y_help, X_help))

    # -1 because now we minimize
    q = matrix(-np.ones([N, 1]))
    G = matrix(-np.eye(N))

    # alpha >= 0 -> -alpha <= 0
    h = matrix(np.zeros(N))

    # alpha*y = 0
    A = matrix(y.reshape(1, -1))
    b = matrix(np.zeros(1))

    solvers.options['show_progress'] = True
```

```
    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

[267]:
```
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 = sum(alpha_i*x_i*y_i)
    w = np.dot(X.T, alpha * y)

    # taken from  support_vecs = (alpha > 1e-4).reshape(-1)
    support_vecs = (alpha > 1e-4)
    biases = y[support_vecs] - np.dot(X[support_vecs, ], w)

    # Better stability
    b = np.mean(biases)

    return w, b
```

1.6 Visualize the result (nothing to do here)

[268]:
```
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 > 1e-4).reshape(-1)
    plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs],␣
↪s=250, marker='*', label='support vectors')
    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]
```

[269]:
```
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 > 1e-4).reshape(-1)])
```

```
     pcost       dcost       gap    pres   dres
 0: -1.5969e+01 -2.6689e+01  6e+02  2e+01  2e+00
 1: -1.3221e+01 -2.7795e+00  6e+01  3e+00  2e-01
 2: -7.9779e-01 -4.6682e-01  2e+00  7e-02  5e-03
 3: -2.8218e-01 -3.8819e-01  1e-01  9e-17  6e-15
 4: -3.5269e-01 -3.6163e-01  9e-03  5e-17  3e-15
```

```
 5: -3.6021e-01 -3.6064e-01  4e-04  8e-17  2e-15
 6: -3.6041e-01 -3.6063e-01  2e-04  8e-17  3e-15
 7: -3.6058e-01 -3.6060e-01  2e-05  6e-17  3e-15
 8: -3.6060e-01 -3.6060e-01  2e-07  2e-16  2e-15
Optimal solution found.
w = [0.73935606 0.41780426]
b = 0.9199371454171219
support vectors: [ 78 134 158]
```

[270]:
```
plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
plt.show()
```

Appendix

We confirm that the submitted solution is original work and was written by us without further assistance. Appropriate credit has been given where reference has been made to the work of others.

Munich, December 8, 2019, Signature Marcel Bruckner (03674122)

Munich, December 8, 2019, Signature Julian Hohenadel (03673879)

Munich, December 8, 2019, Signature Kevin Bein (03707775)