Hw3

Friday, March 5, 2021 8:11 PM

1 The ℓ_2 Support Vector Machine [20pts]

In class, we discussed that if our data is not linearly separable, then we need to modify our optimization problem to include slack variables. The formulation that was used is known as the ℓ_1-norm soft margin SVM. Now consider the formulation of the ℓ_2-norm soft margin SVM, which squares the slack variables within the sum. Notice that non-negativity of the slack variables has been removed.

$$\arg\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2$$
$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \qquad \forall i \in [n]$$

Derive the dual form expression along with any constraints. Work must be shown. *Hints*: Refer to the methodology that was used in class to derive the dual form. The solution is given by:

$$\arg\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \frac{1}{2C}\sum_{i=1}^{n}\alpha_i^2$$
$$\text{s.t. } \alpha_i \geq 0 \quad \forall i \in [n] \quad \text{and} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

solution:

- form the lagrangian function:

$\|w\| = \sqrt{w^T w}$

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2 - \sum_{i=1}^{n}\alpha_i\left[y_i(w^T x_i + b) - 1 + \xi_i\right]$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i x_i = 0$$

$$\Rightarrow w = \sum_{i=1}^{n}\alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$w^T w = \sum_{i=1}^{n}\alpha_i y_i^T x_i^T$$
$$\sum_{j=1}^{n}\alpha_j y_j \alpha_j$$

$$\frac{\partial L}{\partial \xi_i} = C\xi_i - \alpha_i = 0$$

$$\Rightarrow \xi_i = \frac{\alpha_i}{C}$$

$$L(w, b, \xi) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \frac{C}{2}\sum_{i=1}^{n}\frac{\alpha_i^2}{C^2}$$
$$- \sum_{i=1}^{n}\left[\alpha_i y_i w^T x_i + \alpha_i y_i b - \alpha_i + \alpha_i\xi_i\right]$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \frac{1}{2C}\sum_{i=1}^{n}\alpha_i^2$$
$$- \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\frac{2\alpha_i^2}{2C}$$

$$-\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\frac{2\alpha_i^2}{2C}$$

$$L(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_i - \frac{1}{2C}\sum_{i=1}^{n}\alpha_i^2$$

$$\text{s.t.} \quad \alpha_i \geq 0 \;, \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$\alpha_i = C\,\xi_i$$

2 Domain Adaptation Support Vector Machines [20pts]

We now look at a different type of SVM that is designed for domain adaptation and optimizes the hyperplanes given by $\mathbf{w}_S$ (source hyperplane) before optimizing $\mathbf{w}_T$ (target hyperplane). The process begins by training a support vector machine on source data then once data from the target are available, train a new SVM using the hyperplane from the first SVM and the data from the target to solve for a new "domain adaptation" SVM.

The primal optimization problem is given by

$$\begin{aligned}
\arg\min_{\mathbf{w}_T,\xi} \quad & \frac{1}{2}\|\mathbf{w}_T\|^2 + C\sum_{i=1}^{n}\xi_i - B\mathbf{w}_T^T\mathbf{w}_S \\
\text{s.t.} \quad & y_i(\mathbf{w}_T^T\mathbf{x}_i + b) \geq 1 - \xi_i && \forall i \in \{1,\ldots,n\} \\
& \xi_i \geq 0 && \forall i \in \{1,\ldots,n\}
\end{aligned}$$

where $\mathbf{w}_S$ is hyperplane trained on the source data (*assumed to be known*), $\mathbf{w}_T$ is hyperplane for the target, $y_i \in \{\pm 1\}$ is the label for instance $\mathbf{x}_i$, C & B are regularization parameters defined by the user and ξ_i is a slack variable for instance $\mathbf{x}_i$. The problem becomes finding a hyperplane, $\mathbf{w}_T$, that minimizes the above objective function subject to the constraints. Solve/derive the dual optimization problem.

Note: I will give the class the solution to this problem prior to the due date because Problem #3 requires that you implement this algorithm in code.

$$L(w_T, b, \xi_i) = \frac{1}{2}\|w_T\|^2 + C\sum_{i=1}^{n}\xi_i - B w_T^T w_S - \sum_{i=1}^{n}\alpha_i\left[y_i(w_T^T x_i + b) - 1 + \xi_i\right] - \sum_{i=1}^{n}\mu_i\xi_i$$

$$\frac{\partial L}{\partial w_T} = w_T - B w_S - \sum_{i=1}^{n}\alpha_i y_i x_i = 0$$

$$\Rightarrow \quad w_T = B w_S + \sum_{i=1}^{n}\alpha_i y_i \alpha_i$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0$$

$$\Rightarrow C = \alpha_i + \mu_i \quad \therefore \quad 0 \leq \alpha_i \leq C$$

$$L = \frac{1}{2}\|W_T\|^2 + C\sum_{i=1}^{n}\xi_i - BW_T^T W_S - \sum_{i=1}^{n}\alpha_i\left[y_i(W_T^T x_i + b) - 1 + \xi_i\right]$$

$$- \sum_{i=1}^{n}\mu_i\xi_i$$

$$W_T^T W_T$$

$$= \frac{1}{2}\left[BW_S^T + \sum_{i=1}^{n}\alpha_i y_i x_i^T\right]\cdot\left[BW_S + \sum_{i=1}^{n}\alpha_i y_i x_i\right]$$

$$= \frac{1}{2}\Bigg[B^2 W_S^T W_S + BW_S^T\sum_{i=1}^{n}\alpha_i y_i x_i + \left(\sum_{i=1}^{n}\alpha_i y_i x_i^T\right)BW_S + \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j\Bigg]$$

$$+ C\sum_{i=1}^{n}\xi_i - B\left[BW_S^T + \sum_{i=1}^{n}\alpha_i y_i x_i^T\right]W_S$$

$$-\sum_{i=1}^{n}\alpha_i y_i BW_S^T - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i\xi_i - \sum_{i=1}^{n}\mu_i\xi_i$$

$$= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j - \frac{1}{2}B^2 W_S^T W_S$$

$$+ \frac{1}{2}BW_S^T\sum_{i=1}^{n}\alpha_i y_i x_i - \frac{1}{2}\left(\sum_{i=1}^{n}\alpha_i y_i x_i^T\right)BW_S$$

$$\text{S.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$C > \alpha_i \geq 0$$

Since $\frac{1}{2}B^2 W_S^T W_S$ is constant w.r.t. the optimization the equation reduces to

$$L = \frac{1}{2}\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{n}B\alpha_i y_i W_S^T x_i$$

$$\text{S.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$C > \alpha_i \geq 0$$

3 Density Estimation (Code) [20pts]

Implement the domain adaptation SVM from Problem #2. A data setfor the source and target domains (both training and testing) have been uploaded to D2L. There are several ways to implement this algorithm. If I were doing this for an assignment, I would implement the SVM (both the domain adaptation SVM and normal SVM) directly using quadratic programming. You do not need to build the classifier (i.e., solve for the bias term); however, you will need to find $\mathbf{w}_T$ and $\mathbf{w}_S$. To find the weight vectors, you will need to solve a quadratic programming problem and look through the documentation to learn how to solve this optiization task. The following Python packages are recommended:

- CVXOPT(https://cvxopt.org/)
- PyCVX (https://www.cvxpy.org/install/)

Note: Your solution can use any of the packages above.

Domain adaptation SVM

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib import cm
import matplotlib.mlab as ml
from scipy.interpolate import griddata
import numpy as np
import cvxopt
```

In [2]:
```
sourse_train = '/home/safwan/Downloads/source_train.csv'
target_train = '/home/safwan/Downloads/target_train.csv'
```

In [3]:
```
sourse_t_pd = pd.read_csv(sourse_train, header=None)
```

In [4]:
```
target_t_pd = pd.read_csv(target_train, header=None)
# target_t_pd
```

In [5]:
```
#seperating the features and labels.
X_s = sourse_t_pd.iloc[:, 0:-1]
y_s = sourse_t_pd.iloc[:, -1]
#conveting to array
X_s = X_s.values
y_s = y_s.values
```

In [6]:
```
#seperating the features and labels.
X_t = target_t_pd.iloc[:, 0:-1]
y_t = target_t_pd.iloc[:, -1]
#conveting to array
X_t = X_t.values
y_t = y_t.values
```

In [7]:
```
s=y_s.reshape(1,-1)
s.shape
```

Out[7]:
```
(1, 200)
```

In [8]:
```
def linear_kernel(x_i, x_j):
    return np.matmul(x_i,x_j.T)
```

Need to write SMV first to find the Ws.

$$\max_{\lambda \geq 0} -\frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j x^{(i)^T} x^{(j)} + \sum_i \lambda_i$$

such that

$$\sum_i \lambda_i y_i = 0$$

$$c \geq \lambda_i \geq 0, \text{for all } i$$

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

In [9]:

```
def svm(X_s, y_s, C):
    n = len(X_s)
    #using the kernal trick
    K = linear_kernel(X_s, X_s)
    #finding the P matrix
    P = cvxopt.matrix(np.matmul(y_s,y_s.T) * K)
    #finding the q matrix
    q = cvxopt.matrix (-1 * np.ones((len(X_s),1)))
    # alpha <= C and -alpha <= 0
    temp1 = np.diag(-(np.ones(n)))
    temp2 = np.identity(n)
    G = cvxopt.matrix(np.vstack((temp1, temp2)))
    temp1 = np.zeros(n)
    temp2 = np.ones(n) * C
    h = cvxopt.matrix(np.hstack((temp1, temp2)))
    A = cvxopt.matrix(y_s.reshape(1,-1),(1, n),'d')
    b = cvxopt.matrix(0.0)
    #solving for alpha
    a = cvxopt.solvers.qp(P, q, G, h, A, b)
    a = np.array(a['x'])
    #finding the weights
    w = np.array([[0 , 0]])
    for i in range(len(a)):
        w = w + a[i] + y_s[i] + X_s[i]

    return w
```

In [10]:

```
w_s = svm(X_s, y_s, 10)
w_s
```

```
     pcost       dcost       gap    pres   dres
```

```
 0: -1.0002e+03 -9.7814e+03  9e+03  4e-14  1e-11
 1: -1.1318e+03 -1.8739e+03  7e+02  2e-15  8e-12
 2: -1.5433e+03 -1.7047e+03  2e+02  9e-14  1e-11
 3: -1.6043e+03 -1.6448e+03  4e+01  7e-14  1e-11
 4: -1.6154e+03 -1.6336e+03  2e+01  2e-14  1e-11
 5: -1.6214e+03 -1.6280e+03  7e+00  7e-15  1e-11
 6: -1.6243e+03 -1.6252e+03  9e-01  3e-14  2e-11
 7: -1.6246e+03 -1.6248e+03  2e-01  1e-14  1e-11
 8: -1.6247e+03 -1.6247e+03  3e-02  2e-16  1e-11
 9: -1.6247e+03 -1.6247e+03  3e-04  2e-16  1e-11
Optimal solution found.
```

Out[10]:
```
array([[1659.75153226, 1652.55403706]])
```

Now we can find DA-SVM

In [11]:
```
def da_svm(X_t, y_t, C, B, w_s):
    n = len(X_t)
    #using the kernal trick
    K = linear_kernel(X_t, X_t)

    #finding the P matrix
    P = cvxopt.matrix(np.matmul(y_t,y_t.T) * K)
    #finding the q matrix
    #Adjusting the q matrix for da-svm
    temp3 = np.matmul(w_s, X_t.T)
    temp4 = np.matmul(B * np.vstack(y_t), temp3)
    temp5 = - (1 - np.diagonal(temp4))
    q = cvxopt.matrix (np.vstack(temp5))

    # alpha <= C and -alpha <= 0
    temp1 = np.diag(-(np.ones(n)))
    temp2 = np.identity(n)
    G = cvxopt.matrix(np.vstack((temp1, temp2)))
    temp1 = np.zeros(n)
    temp2 = np.ones(n) * C
    h = cvxopt.matrix(np.hstack((temp1, temp2)))
    A = cvxopt.matrix(y_t.reshape(1,-1),(1, n),'d')
    b = cvxopt.matrix(0.0)
    #solving for alpha
    a = cvxopt.solvers.qp(P, q, G, h, A, b)
    a = np.array(a['x'])
    #finding the weights
    w = np.array([[0 , 0]])

    for i in range(len(a)):

        w = w + a[i] + y_t[i] + X_t[i]

        w = B * w_s + w

    return w
```

In [12]:
```
da_svm(X_t, y_t, 10 , 1, w_s)
```

```
     pcost       dcost       gap    pres   dres
 0: -2.5868e+08 -1.7333e+06  2e+09  7e+02  1e-13
 1: -2.6673e+06 -1.7282e+06  2e+07  7e+00  3e-13
 2: -1.7172e+04 -1.3750e+06  3e+06  5e-01  1e-12
```

```
 3:  1.1707e+05 -1.4545e+05  3e+05  2e-15  1e-15
 4:  2.0034e+04 -2.0933e+04  4e+04  9e-16  4e-16
 5:  2.1743e+03 -2.3440e+03  5e+03  8e-16  3e-16
 6:  1.0083e+02 -8.0851e+01  2e+02  5e-16  5e-16
 7:  2.8074e+00 -6.1000e+00  9e+00  2e-16  4e-16
 8: -1.3106e+00 -1.9276e+00  6e-01  2e-16  3e-16
 9: -1.4032e+00 -1.4218e+00  2e-02  2e-16  4e-16
10: -1.4041e+00 -1.4043e+00  2e-04  2e-16  3e-16
11: -1.4041e+00 -1.4041e+00  2e-06  2e-16  4e-16
12: -1.4041e+00 -1.4041e+00  2e-08  2e-16  2e-16
Optimal solution found.
```

Out[12]:
```
array([[82983.35581857, 82654.23975857]])
```

In [ ]:
```

```