Statistical Machine Learning Exercise Sheet 5

- Laura Haege
- Philipp Noel von Bachmann, Matrikelnummer: 4116220

Exercise 9

(a)

The primal problem is given as: $min_{w\in(R)^d,\epsilon\in(R)^n}\frac{1}{2}\|w\|^2+\frac{C}{n}\sum_i\epsilon_i$, subject to: $Y_i(\langle w,X_i\rangle)\geq 1-\epsilon,\epsilon_i\geq 0$.

The lagrangian of the problem is : $\frac{1}{2}\|w\|^2+\frac{C}{n}\sum_i\epsilon_i+\sum_i\alpha_i(1-\epsilon_i-Y_i\langle w,X_i\rangle)-\sum_i\beta_i\epsilon_i$

Now take the derivative:

$$\begin{aligned}
&\nabla_w\frac{1}{2}\|w\|^2+\frac{C}{n}\sum_i\epsilon_i+\sum_i\alpha_i(1-\epsilon_i-Y_i\langle w,X_i\rangle)-\sum_i\beta_i\epsilon_i && (1)\\
&=w-\sum_i\alpha_iY_iX_i && (2)\\
&\Rightarrow w=\sum_i\alpha_iY_iX_i && (3)
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial}{\partial\epsilon_i}\frac{1}{2}\|w\|^2+\frac{C}{n}\sum_i\epsilon_i+\sum_i\alpha_i(1-\epsilon_i-Y_i\langle w,X_i\rangle)-\sum_i\beta_i\epsilon_i && (4)\\
&=\frac{C}{n}-\alpha_i-\beta_i && (5)
\end{aligned}$$

By setting the gradient of ϵ to 0 we can use this equation to get rid of beta, as $\beta_i=\frac{C}{n}-\alpha_i$, this imposed the constrain $\alpha_i<\frac{C}{n}$ due to the positivity of β.

No substitution in L we get:

$$\begin{aligned}
&\frac{1}{2}\|\sum_i\alpha_iY_iX_i\|^2+\frac{C}{n}\sum_i\epsilon_i\sum_i\alpha_i(1-\epsilon_i-Y_i\langle\sum_i\alpha_iY_iX_i,X_i\rangle)-\sum_i(\frac{C}{n}-\alpha_i)\epsilon_i && (6)\\
&\frac{1}{2}\|\sum_i\alpha_iY_iX_i\|^2+\sum_i(\frac{C}{n}-\alpha_i)\epsilon_i\sum_i\alpha_i(1-Y_i\langle\sum_i\alpha_iY_iX_i,X_i\rangle)-\sum_i(\frac{C}{n}-\alpha_i)\epsilon_i && (7)\\
&=\frac{1}{2}\|\sum_i\alpha_iY_iX_i\|^2+\sum_i\alpha_i-\sum_i\alpha_iY_i\langle\sum_i\alpha_iY_iX_i,X_i\rangle) && (8)\\
&=\sum_i\alpha_i-\frac{1}{2}\|\sum_i\alpha_iY_iX_i\|^2 && (9)\\
&=\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_jy_iy_j\langle x_i,x_j\rangle && (10)
\end{aligned}$$

Or dual problem is therefore given as: $\max_{\alpha\in\mathbb{R}^n}\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_jy_iy_j\langle x_i,x_j\rangle$, subject to $0\leq\alpha_i\leq\frac{C}{n}$.

(b)

1. The derivative is given by:

$$\begin{aligned}
\frac{\partial}{\partial\alpha_r}\Psi(\dots,\alpha_r,\dots)&=\frac{\partial}{\partial\alpha_r}\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_jY_iY_j\langle x_i,x_j\rangle && (11)\\
&=1-\frac{\partial}{\partial\alpha_r}\frac{1}{2}\sum_{i,j}\alpha_i\alpha_jY_iY_j\langle x_i,x_j\rangle && (12)\\
&=1-\alpha_rY_r^2\langle x_r,x_r\rangle-\frac{\partial}{\partial\alpha_r}\frac{1}{2}\sum_{i,j\neq i}\alpha_i\alpha_jY_iY_j\langle x_i,x_j\rangle && (13)\\
&=1-\alpha_rY_r^2\langle x_r,x_r\rangle-\frac{1}{2}\sum_{j\neq r}\alpha_jY_rY_j\langle x_r,x_j\rangle && (14)
\end{aligned}$$

Set to 0:

$$\begin{aligned}
\alpha_rY_r^2\langle x_r,x_r\rangle&=1-\frac{1}{2}\sum_{j\neq r}\alpha_jY_rY_j\langle x_r,x_j\rangle && (15)\\
\alpha_r&=1-\frac{1}{2}\sum_{j\neq r}\alpha_jY_rY_j\langle x_r,x_j\rangle/Y_r^2\langle x_r,x_r\rangle && (16)
\end{aligned}$$

As the dual problem is concave, we can get the best valid solution by solving the problem and projecting it to the closed point in the set of valid solutions. This results in $\alpha_r^{new}=max(0,min(\alpha_r,\frac{C}{n}))$

2. The KKT condition is given by $\alpha_i(1-\epsilon_i-Y_i\langle w,X_i\rangle)=0$.

- If $\alpha_i=0$, point is outside the margin, and correctly classified, therefore $y_i\langle w,x_i\rangle\geq 1$
- If $\alpha_i=1$, point is on margin, therefore $\epsilon_i=0$. This means that for the KKT condition to hold, $(1-Y_i\langle w,X_i\rangle)=0\Rightarrow Y_i\langle w,X_i\rangle)=1$
- If $\alpha_i=\frac{C}{n}$, point is inside margin, therefore $\epsilon_i>0$. This means that for the KKT condition to hold, $(1-\epsilon_i-Y_i\langle w,X_i\rangle)=0\Rightarrow Y_i\langle w,X_i\rangle)\leq 1$

Where the first step (meaning of each α_i) of each case is given in the lecture.

(c)

In [1]:

```python
from matplotlib import pyplot as plt
import numpy as np
from scipy.spatial.distance import cdist
```

In [10]:

```python
def CoordinateDescentSVM(Xtrain, Ytrain, C, Xtest, Ytest):
    ''' compute the solution of linear SVM
    Xtrain, Ytrain: training set
    Xtest, Ytest: test set (only to monitor test error)
    C: error parameter

    w: weights linear SVM
    TrainError, TestError: training and test errors over iterations
    '''
    n = Xtrain.shape[0]
    alpha = np.zeros([n, 1]) # initialize dual variables
    w = Xtrain.T @ (Ytrain * alpha) # initialize primal variables

    counter = 0
    converged = False
    eps = 1e-3
    TrainError, TestError = [], []

    indices = np.array(list(range(n)))
    # print(np.where(indices!=1))

    while not converged:
        # select coordinate to update
        r = counter % n

        # solve the subproblem for coordinate r without any constraints
        w_old = np.sum(alpha  * Y_train  * X_train, axis=0)
        alpha_r = alpha[r] - Y_train[r] * (w_old @ X_train[r] - Y_train[r]) / (X_train[r] @ X_train[r])

        # project the solution to the interval [0, C / n]
        alpha[r] = np.maximum(0.0, np.minimum(alpha_r, C/n))

        # monitor the progress of the method computing the dual
        # objective DualObj

        if (counter + 1) % 100 == 0:
            DualObj = np.sum(alpha) - 0.5 * np.sum(np.outer(alpha, alpha) * np.outer(Ytrain, Ytrain) * cdist
            print('iteration={} dual obj={:.3f}'.format(
                counter + 1, DualObj))

        # compute the training and test error with the current iterate alpha
        w = np.expand_dims(np.sum(alpha * Ytrain * Xtrain,   axis=0), axis=1)
        TrainError.append(np.sum(np.sign(Xtrain @ w) != Ytrain)/n)
        TestError.append(np.sum(np.sign(Xtest @ w) != Ytest)/Xtest.shape[0])
        #print(f"Train accuracy in epoch {counter + 1}: {TrainError}")

        # if the KKT conditions are satisfied up to the tolerance eps by the
        # the current iterate alpha then set converged = True
        converged = np.all(Ytrain *(Xtrain @ w) > 1-eps)
        counter += 1


    # compute the primal solution w from alpha
    # see above

    # show final dual objective
    print('final iteration={} dual obj={:.3f}'.format(counter, DualObj))

    return w, TrainError, TestError
```

(d)

In [3]:

```python
data = np.load("digits01.npy", allow_pickle=True).item()
X_train = data["Xtrain"]
Y_train = data["Ytrain"]
X_test = data["Xtest"]
Y_test = data["Ytest"]
```

In [8]:

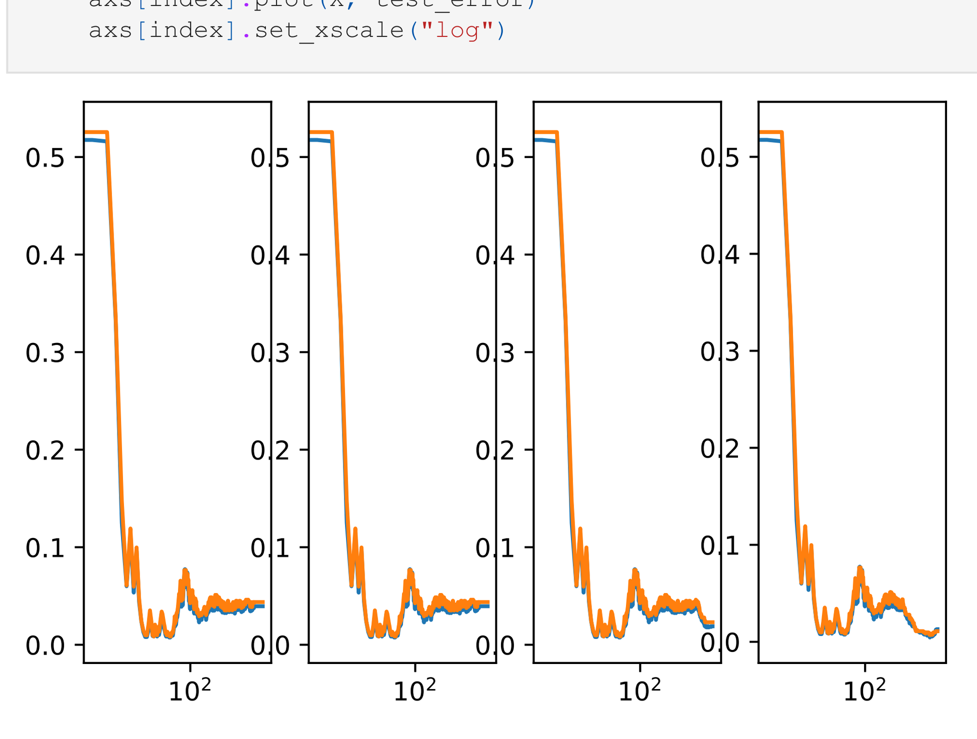
```python
C_list = [10, 100, 200, 500]
train_errors = []
test_errors = []
for index, C in enumerate(C_list):
    w, train_error, test_error = CoordinateDescentSVM(X_train, Y_train, C,  X_test, Y_test)
    train_errors.append(train_error)
    test_errors.append(test_error)
```

```
iteration=100 dual obj=0.499
iteration=200 dual obj=0.998
iteration=300 dual obj=1.495
iteration=400 dual obj=1.991
iteration=500 dual obj=2.486
iteration=600 dual obj=2.979
iteration=700 dual obj=3.472
iteration=800 dual obj=3.963
iteration=900 dual obj=4.453
iteration=1000 dual obj=4.942
iteration=1100 dual obj=5.431
iteration=1200 dual obj=5.917
iteration=1300 dual obj=6.403
iteration=1400 dual obj=6.887
iteration=1500 dual obj=7.369
iteration=1600 dual obj=7.853
iteration=1700 dual obj=8.335
iteration=1800 dual obj=8.814
iteration=1900 dual obj=9.290
iteration=2000 dual obj=9.768
iteration=2100 dual obj=9.768
iteration=2200 dual obj=9.768
iteration=2300 dual obj=9.768
iteration=2400 dual obj=9.768
iteration=2500 dual obj=9.768
iteration=2600 dual obj=9.768
iteration=2700 dual obj=9.768
iteration=2800 dual obj=9.768
iteration=2900 dual obj=9.768
iteration=3000 dual obj=9.768
final iteration=3001 dual obj=9.768
iteration=100 dual obj=4.940
iteration=200 dual obj=9.772
iteration=300 dual obj=14.481
iteration=400 dual obj=19.056
iteration=500 dual obj=23.553
iteration=600 dual obj=27.913
iteration=700 dual obj=32.172
iteration=800 dual obj=36.299
iteration=900 dual obj=40.305
iteration=1000 dual obj=44.211
iteration=1100 dual obj=48.114
iteration=1200 dual obj=51.705
iteration=1300 dual obj=55.250
iteration=1400 dual obj=58.694
iteration=1500 dual obj=61.911
iteration=1600 dual obj=65.257
iteration=1700 dual obj=68.475
iteration=1800 dual obj=71.366
iteration=1900 dual obj=73.996
iteration=2000 dual obj=76.803
iteration=2100 dual obj=76.803
iteration=2200 dual obj=76.803
iteration=2300 dual obj=76.803
iteration=2400 dual obj=76.803
iteration=2500 dual obj=76.803
iteration=2600 dual obj=76.803
iteration=2700 dual obj=76.803
iteration=2800 dual obj=76.803
iteration=2900 dual obj=76.803
iteration=3000 dual obj=76.803
final iteration=3001 dual obj=76.803
iteration=100 dual obj=9.761
iteration=200 dual obj=19.087
iteration=300 dual obj=27.926
iteration=400 dual obj=36.225
iteration=500 dual obj=44.214
iteration=600 dual obj=51.653
iteration=700 dual obj=58.688
iteration=800 dual obj=65.196
iteration=900 dual obj=71.221
iteration=1000 dual obj=76.845
iteration=1100 dual obj=82.456
iteration=1200 dual obj=86.821
iteration=1300 dual obj=91.001
iteration=1400 dual obj=94.777
iteration=1500 dual obj=97.685
iteration=1600 dual obj=101.242
iteration=1700 dual obj=104.334
iteration=1800 dual obj=106.578
iteration=1900 dual obj=108.286
iteration=2000 dual obj=110.662
iteration=2100 dual obj=110.809
iteration=2200 dual obj=110.901
iteration=2300 dual obj=110.984
iteration=2400 dual obj=111.001
iteration=2500 dual obj=111.125
iteration=2600 dual obj=111.163
iteration=2700 dual obj=111.199
iteration=2800 dual obj=111.268
iteration=2900 dual obj=111.307
iteration=3000 dual obj=111.333
final iteration=3001 dual obj=111.333
iteration=100 dual obj=23.506
iteration=200 dual obj=44.293
iteration=300 dual obj=62.037
iteration=400 dual obj=76.406
iteration=500 dual obj=88.836
iteration=600 dual obj=97.856
iteration=700 dual obj=105.191
iteration=800 dual obj=110.541
iteration=900 dual obj=115.226
iteration=1000 dual obj=119.085
iteration=1100 dual obj=123.084
iteration=1200 dual obj=125.445
iteration=1300 dual obj=128.110
iteration=1400 dual obj=131.010
iteration=1500 dual obj=132.700
iteration=1600 dual obj=135.691
iteration=1700 dual obj=137.248
iteration=1800 dual obj=138.803
iteration=1900 dual obj=140.031
iteration=2000 dual obj=141.889
iteration=2100 dual obj=146.384
iteration=2200 dual obj=149.759
iteration=2300 dual obj=152.104
iteration=2400 dual obj=153.859
iteration=2500 dual obj=155.195
iteration=2600 dual obj=155.718
iteration=2700 dual obj=156.104
iteration=2800 dual obj=156.235
iteration=2900 dual obj=156.250
iteration=3000 dual obj=156.258
final iteration=3001 dual obj=156.258
```

Note: We just ran it for 3000 epochs each (took long enough)

In [9]:

```python
fig, axs = plt.subplots(1,4)
for index, (train_error, test_error) in enumerate(zip(train_errors, test_errors)):
    X = list(range(len(train_error)))
    axs[index].plot(X, train_error)
    axs[index].plot(X, test_error)
    axs[index].set_xscale("log")
```



In [ ]: