# Stony Brook University
# CSE512 – Machine Learning – Spring 17
# Homework 3, Due: March, 28, 2017, 11:59PM
# Name : Siddharth Shah SBU ID: 110957500

## Instructions

- The homework is due on March 28, 2017. Anything that is received after the deadline will not be considered.
- The write-up **must** be prepared in Latex, including the Matlab code and figures in the report, and converted to pdf.
- We can use any Latex class you like, just report question number and your answer.
- If the question requires you to implement a Matlab function, the answer should be your code. Make sure it is sufficiently well documented that the TAs can understand what is happening.
- Each Question, regardless of how many sub-questions contains, is worth 10 points.

## Preliminaries

Suppose labeled points $(\boldsymbol{x}, y)$ are drawn from $\mathcal{X} \times \mathbb{R}$, where $\mathcal{X}$ is the feature space. Suppose you are also given a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which computes the inner product for a feature map $\phi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space for $k$. In other words,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle \ .$$

Suppose you have collected a training set of $m$ examples

$$S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m) \in \mathcal{X} \times \mathbb{R}\} \ .$$

In the following, everytime we use an algorithm with a kernel $k$, we mean using the algorithm on the transformed dataset

$$S' = \{(\phi(\boldsymbol{x}_1), y_1), (\phi(\boldsymbol{x}_2), y_2), \cdots, (\phi(\boldsymbol{x}_m), y_m) \in \mathcal{H} \times \mathbb{R}\} \ .$$

A Gaussian kernel is defined as $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2\right)$. For all the experiments below, set $\gamma = 0.001$.

For the linear kernel, simply $\phi(\boldsymbol{x}) = \boldsymbol{x}$, and $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$.

The provvided dataset is a subset of the 'Adult' dataset: the task is to predict if a person makes over 50K a year, see `https://archive.ics.uci.edu/ml/datasets/Adult`.

## 1 Support Vector Machines

In this problem, you will implement the soft-margin SVMs without bias $b$ using two different optimization techniques: (1) quadratic programming and (2) gradient descent.

## 1.1 Question 1: Primal and Dual of Kernel SVM

Quadratic programs refer to optimization problems in which the objective function is quadratic and the constraints are linear. Quadratic programs are well studied in optimization literature, and there are many efficient solvers. Many Machine Learning algorithms are reduced to solving quadratic programs, as you already saw in the previous assignments. In this question, we will use the quadratic program solver of Matlab to optimize the dual objective of a kernel SVM.

The primal objective of a kernel SVM without the bias $b$ can be written as

$$\min_{\boldsymbol{w}\in\mathcal{H}} \ \frac{1}{2}\left\|\boldsymbol{w}\right\|_2^2 + C\sum_{j=1}^{m}\ell(\boldsymbol{w},\phi(\boldsymbol{x}_j),y_j) \ . \tag{1}$$

Here $\ell(\boldsymbol{w},\boldsymbol{x}_j,y_j)$ is the *Hinge loss* of the $j$-th instance:

$$\ell(\boldsymbol{w},\boldsymbol{x}_j,y_j) = \max\left(1-y_j\langle\boldsymbol{w},\phi(\boldsymbol{x}_j)\rangle,0\right) \ .$$

The corresponding dual objective is

$$\max_{\boldsymbol{\alpha}} \ \sum_{j=1}^{m}\alpha_j - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i\alpha_i y_j\alpha_j k(\boldsymbol{x}_i,\boldsymbol{x}_j) \tag{2}$$

$$0\le\alpha_j\le C\ \forall j \ . \tag{3}$$

(a) Write the SVM dual objective as a quadratic program. Look at the `quadprog` function of Matlab, and write down what `H, f, A, b, Aeq, beq, lb, ub` are. Solution After converting (2) into the Quadratic programming standard form:
Minimizing $0.5\sum_{i=1}^{m}\sum_{j=1}^{m} y_i\alpha_i y_j\alpha_j k(\boldsymbol{x}_i,\boldsymbol{x}_j) - \sum_{j=1}^{m}\alpha_j$ subject to constraint (3)

$$\begin{aligned} H &= (y*y).*K \\ f &= -ones(m,1) \\ lb &= zeros(m,1) \\ ub &= C*ones(1,m) \\ A &= zeros(m,m) \\ b &= zeros(m,1) \\ Aeq &= zeros(m,m) \\ beq &= zeros(m,1) \end{aligned}$$

(b) From the Lagrangian of the above problem, show that $\boldsymbol{w} = \sum_{i=1}^{m}\alpha_i y_i\phi(\boldsymbol{x}_i)$.

Solution

The Lagrangian can be written as follows:

$$L(w,\alpha) = \frac{1}{2}\left\|\boldsymbol{w}\right\|_2^2 + C\sum_{i=1}^{m}\alpha_i(1-y_i\langle\boldsymbol{w},\phi(\boldsymbol{x}_i)\rangle)$$

minimize L with respect to w

$$\nabla_w(L) = \left\|\boldsymbol{w}\right\| + C\sum_{i=1}^{m}\alpha_i(-y_i\phi(\boldsymbol{x}_i)) = 0$$

$$w = C \sum_{i=1}^{m} \alpha_i (y_i \phi(\boldsymbol{x}_i))$$

By dividing by C, we reduce the norm of w further

$$w = \sum_{i=1}^{m} \alpha_i y_i \phi(\boldsymbol{x}_i)$$

## 1.2 Question 2: Implement a kernel SVM using Quadratic Programming

(a) Using the answer to part (a) in Question 1, use quadratic programming to write a function that solves the dual SVM objective. In Matlab, you can use the function `quadprog`. The prototype must be

```
alpha = train_ksvm_dual(X, y, C, kernel, gamma)
```

where `kernel` is 'gaussian' for Gaussian kernels and 'linear' for linear kernels, `gamma` is the parameter of the Gaussian kernel. Solution

```
 function [ alpha, objective_value, support_vectors ] = train_ksvm_dual(X, y, C, kernel, gamma)
K = [];
[d,m] = size(X);
if strcmp(kernel, 'linear')
    K = X'*X;
else
    if strcmp(kernel,'gaussian')
        K = zeros(m,m);
        for k = 1:m
            for l = 1:m
                K(k,l) = exp(-gamma*(norm(X(:,k)-X(:,l))^2));
            end
        end
    end
end
%   The hessian H_ij is y_i * y_j * K_ij
H = (y'*y).*K;
f = -ones(m,1);
lb = zeros(m,1);
ub = C*ones(1,m);
A = zeros(m,m);
b = zeros(m,1);
Aeq = zeros(m,m);
beq = zeros(m,1);
alpha = quadprog(H,f,A,b,Aeq,beq, lb, ub);
disp(size(alpha))
Z = ((alpha'*alpha).*H)./2;
objective_value = sum(Z(:)) - sum(alpha);
support_vectors = numel(alpha(alpha>0.001));
end
```

(b) Using the answer to part (b) in Question 1, write a function to test the solution obtained above, producing a list of predictions from the inputs $\boldsymbol{\alpha}$, the training data, and the test samples. The prototype must be

```
ypredicted = test_ksvm_dual(alpha, Xtr, ytr, Xte, kernel, gamma)
```

where `kernel` is 'gaussian' for Gaussian kernels and 'linear' for linear kernels, `gamma` is the parameter of the Gaussian kernel.

```
function [ ypredicted ] = test_ksvm_dual(alpha, Xtr, ytr, Xte, kernel, gamma)
    K = [];
    [d,n] = size(Xte);
    [d,m] = size(Xtr);
    ypredicted = zeros(n,1);
    for i = 1:n
        summation = 0;
        for j = 1:m
            if strcmp(kernel, 'linear')
                K = Xtr(:,j)' *Xte(:,i);
            else
                if strcmp(kernel,'gaussian')
                   K = exp(-gamma*(norm(Xtr(:,j)-Xte(:,i))^2));
                end
            end
            summation = summation + alpha(j)* ytr(j) * K;
        end
        ypredicted(i) = sign(summation);
    end
end
```

(c) Set $C = 10$, train and test your SVM implementation with linear and Gaussian kernel on the provvided dataset and report the accuracy, the objective value, and the number of support vectors.
Solution:
We know that if alpha is non-zero then it is a support vector.

I have used the threshold of alpha values greater than 0.001 to find the number of support vectors.

- Linear Kernel
  Objective value: 3.8876e+08
  Support Vectors: 1812
  Accuracy: 84.46%

- Gaussian Kernel
  Objective value: 6.1760e+11
  Support Vectors: 2104
  Accuracy: 85.16%

(d) Repeat the above question with $C = .1$.
Solution:
Again here, I have used the threshold of 0.001 for the value of alpha

- Linear Kernel
  Objective value: 3.8876e+08
  Support Vectors: 1926

Accuracy: 84.75%

- Gaussian Kernel
  Objective value: 7.48e+07
  Support Vectors: 2961
  Accuracy: 75.96%

### 1.3 Question 3: Implement Linear SVM using Sub-Gradient Descent

In class we saw that Sub-Gradient Descent can be used to optimize a convex and not differentiable objective function. Here, we will use it to optimize the SVM primal objective in (1) for the linear kernel.

We can use sub-gradient descent to optimize this objective. Denote by $L(\boldsymbol{w})$ the function $\frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^{m}\ell(\boldsymbol{w}, \boldsymbol{x}_j, y_j)$. The update rule for $\boldsymbol{w}$ at iteration $t$ is

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta_t \partial_{\boldsymbol{w}} L(\boldsymbol{w}_t), \tag{4}$$

where $\partial_{\boldsymbol{w}} L(\boldsymbol{w}_t)$ denotes the sub-gradient of $L$ w.r.t. $\boldsymbol{w}$ evaluated in $\boldsymbol{w}_t$. The pseudo-code is in Algorithm 1.

**Algorithm 1** Sub-gradient descent for linear SVM

$\boldsymbol{w}_1 = \mathbf{0}$
**for** $t = 1, 2, \cdots, T$ **do**
  $\eta_t \leftarrow \frac{a}{t}$
  Update $\boldsymbol{w}$ using Eq. (4)
**end for**

(a) Write the sub-gradient descent update rule for $\boldsymbol{w}$ for linear SVMs.
Solution:

The Hinge loss function is:

$$\ell(\boldsymbol{w}, \boldsymbol{x}_j, y_j) = [1 - y\langle wx\rangle]_+$$

Thus,

$$L(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^{m}[1 - y\langle w, x\rangle]_+$$

Taking partial derivative,
if $y\langle w, x\rangle > 1$

$$\partial_{\boldsymbol{w}} L(\boldsymbol{w}) = \boldsymbol{w}$$

if $y\langle w, x\rangle \leq 1$

$$\partial_{\boldsymbol{w}} L(\boldsymbol{w}) = \boldsymbol{w} + C\sum_{j=1}^{m} -y\boldsymbol{x}$$

Thus, the update rule would be as below
if $y\langle w, x\rangle > 1$

$$\boldsymbol{w}_{t+1} := \boldsymbol{w}_t - \eta_t \boldsymbol{w}_t$$

if $y\langle w, x\rangle \leq 1$

$$\boldsymbol{w}_{t+1} := \boldsymbol{w}_t - \eta_t(\boldsymbol{w}_t + C\sum_{j=1}^{m} -y\boldsymbol{x})$$

(b) Implement Algorithm 1 for linear SVMs. $a$ is a tunable parameter. The prototype must be

```
[w] = train_svm_sgd(X, y, C, a, T)
```

Solution

```
function [ w ] = train_svm_sgd(X, y, C, a, T)
  [d,m] = size(X);
  w = zeros(d,T);
  training_error = zeros(T,1);
  for t = 2:T
      rate = a/t;
      update = zeros(d,1);
      loss = 0;
      for i = 1:m
          if  y(i)*(w(:,t-1)'* X(:,i)) < 1
              update = update + y(i) * X(:,i);
              loss = loss + (1-y(i)*(w(:,t-1)'* X(:,i)));
          end
      end

      w(:,t) = w(:,t-1) - rate * ( w(:,t-1) - C*update);
      objective(t-1) = 1/2*(norm(w(:,t-1))^2) + C*loss;
      ypredicted = sign(w(:,t-1)'*X);
      Z = y.*ypredicted
      training_error(t-1) = size(Z(Z<1),2)
  end
%      Objective Value
   figure;
   loglog(1:T-3,objective(3:T-1))
   hold on;
   title('Objective Value during sub gradient descent');
   xlabel('T');
   ylabel('Objective Value');
   hold off;
%    Training Error
  figure;
  plot(training_error(2:T-1));
  hold on;
  title('Training Error during sub gradient descent');
  xlabel('T');
  ylabel(' Training error');
  hold off;
end
```

I have used the below script to plot the test error.

```
T = 10000
```

```
C = 10
w = train_svm_sgd(Xtr, ytr, C, 1, T)
[d,m] = size(Xte);
for t = 2:T
    ypredicted = sign(w(:,t-1)'*Xte);
    Z = yte.*ypredicted
    test_error(t-1) = size(Z(Z<1),2)
end
plot(test_error);
hold on;
title('Test Error during sub gradient descent');
xlabel('T');
ylabel(' Test error');
hold off;
```

(c) The theory in [**?**] tells us that for this optimization problem the optimal (worst-case) setting of the learning rate is $\eta_t = \frac{1}{t}$, that is setting $a = 1$, but if you are free to choose another learning rate if you want. Using the provvided dataset as your training set, run $T = 10000$ iterations using $C = 10$. Be patient: it might take a while to compute.

(d) Plot the value of the objective function in Eq. (1) after each iterations in a log-log plot, to better show the behaviour. Compare with the objective value obtained in Question 2.

(e) Plot the training error after each iteration, on a normal plot, and compare it with the results in Question 2.

(f) Plot the test error after each iteration, on a normal plot, and compare it with the results in Question 2.

(g) Change $C$ to .1 and repeat what you did in the previous four points. Solution:

## 1.4 Question 4: Invariance to Additive Constants in Kernels

Consider the soft-margin kernel SVM formulation, that includes the bias term $b$:

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^{m}\max(1 - y_j(\langle \boldsymbol{w}, \phi(\boldsymbol{x}_j)\rangle + b), 0) \ .$$

Its dual formulation is

$$\max_{\boldsymbol{\alpha}} \ \sum_{j=1}^{m}\alpha_j - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i\alpha_i y_j\alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{5}$$

$$\sum_{j}^{m} y_j\alpha_j = 0 \tag{6}$$

$$0 \le \alpha_j \le C \ \forall j \ . \tag{7}$$

In this question we will investigate the effect of adding a constant to the kernel function. In particular, prove that optimal value of both objective functions and the optimal solution $\boldsymbol{w}^*$ does not change if we add a constant $c$ to the kernel, i.e. $k(\cdot,\cdot) \to k(\cdot,\cdot) + c$.

*Hint:* Starts from the dual formulation and see what happens when the kernel changes as described. The proof is short...

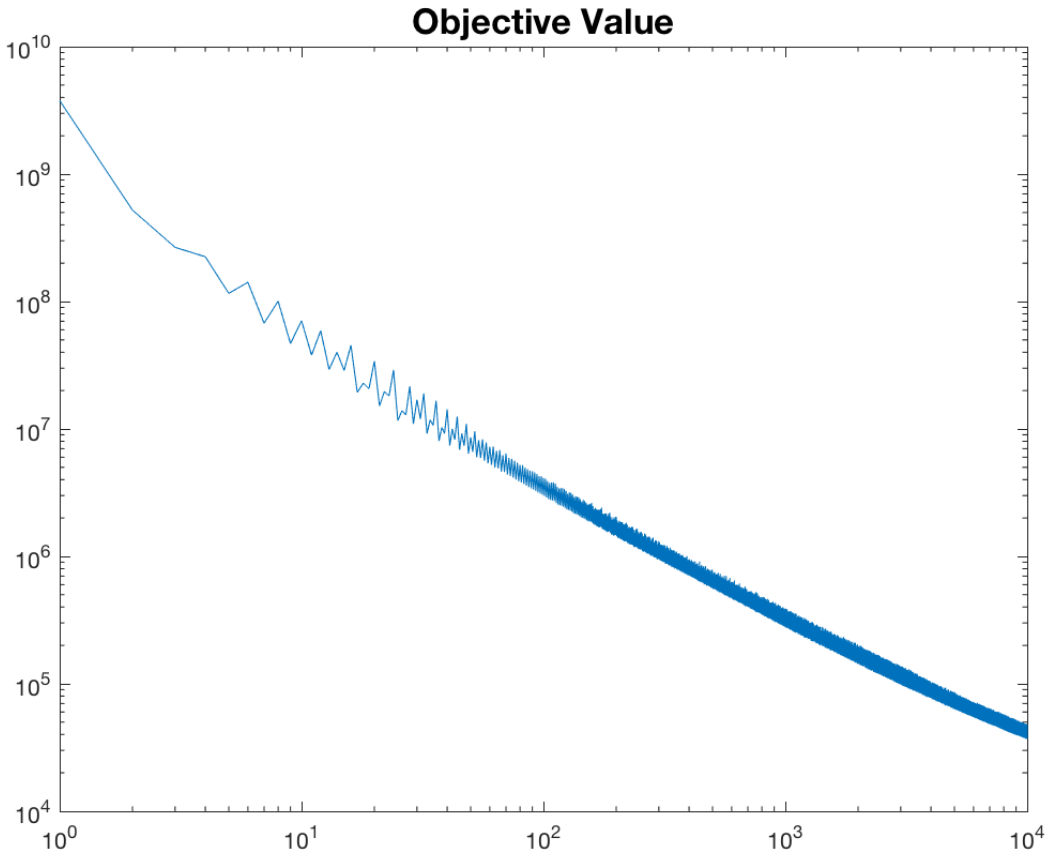


Figure 1: Objective value over iterations C=10

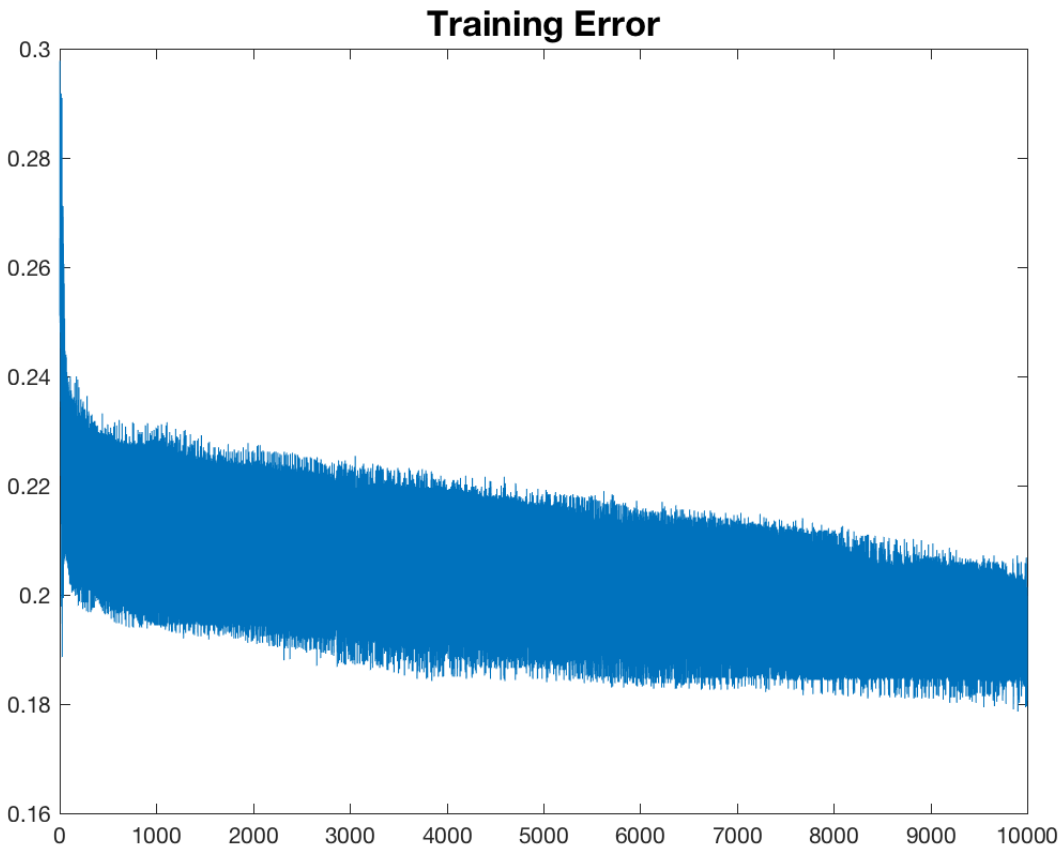


Figure 2: Training error over iterations C=10

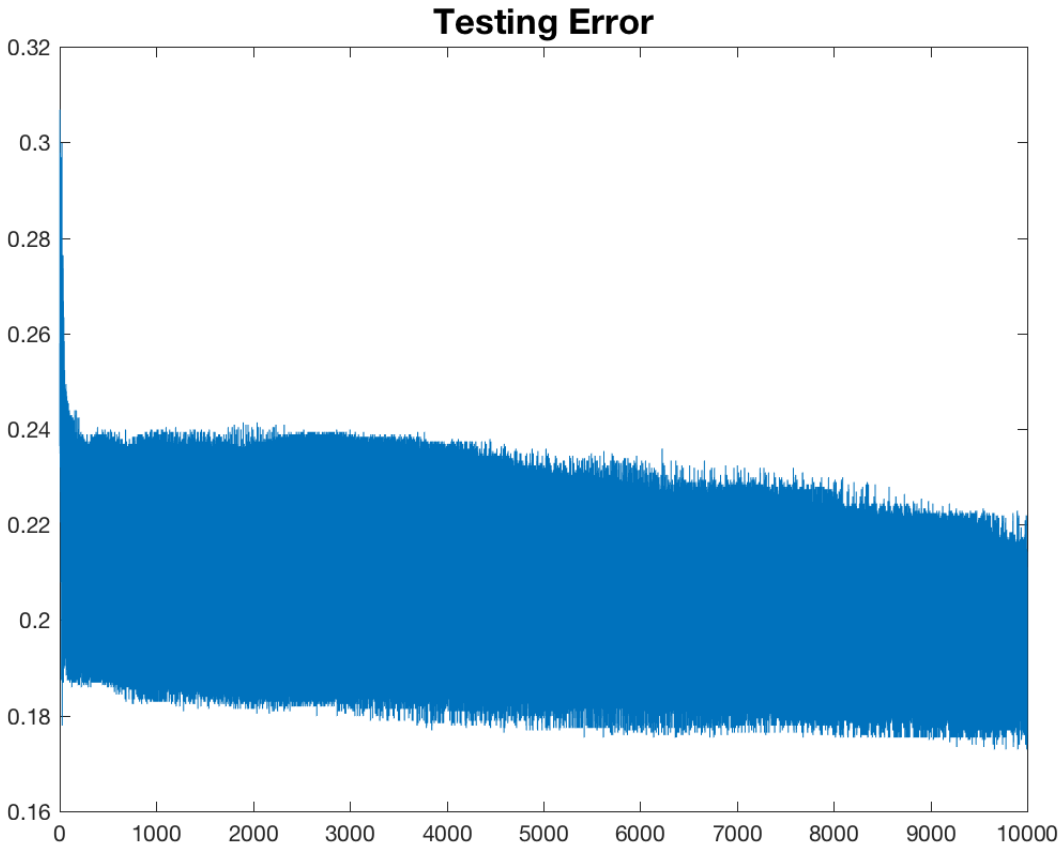


Figure 3: Test error over iterations C=10

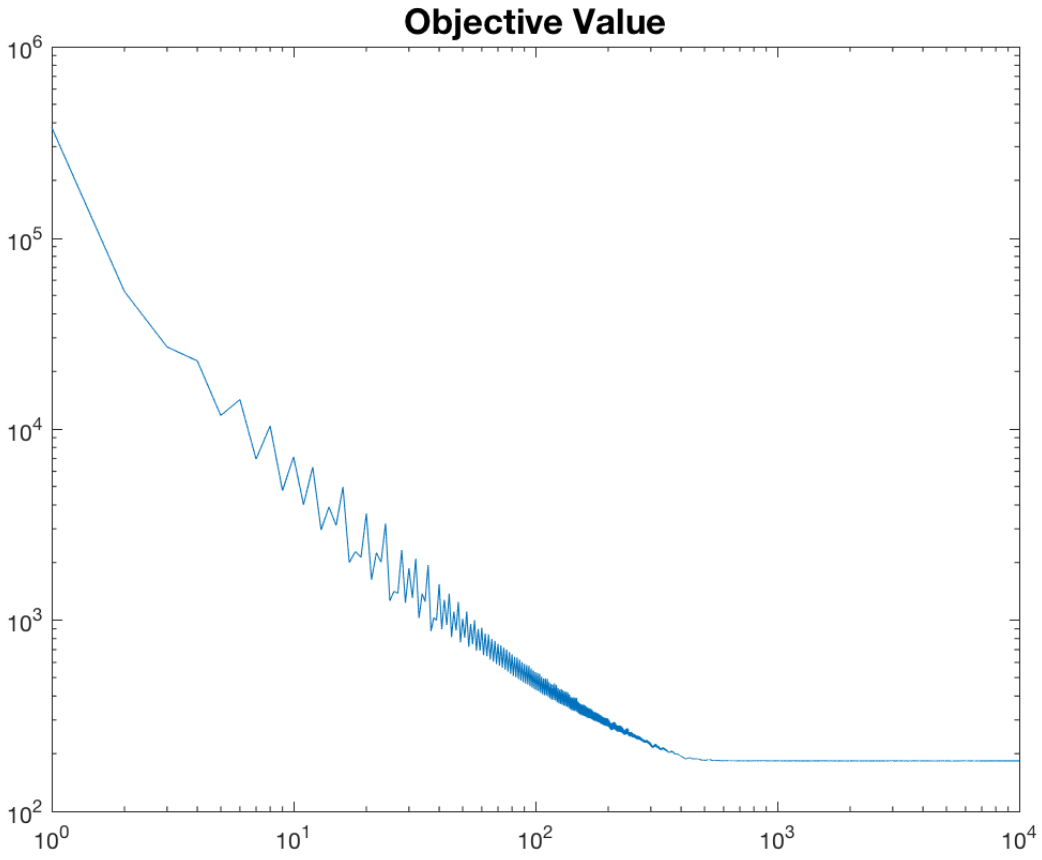


Figure 4: Objective value over iterations C=0.1

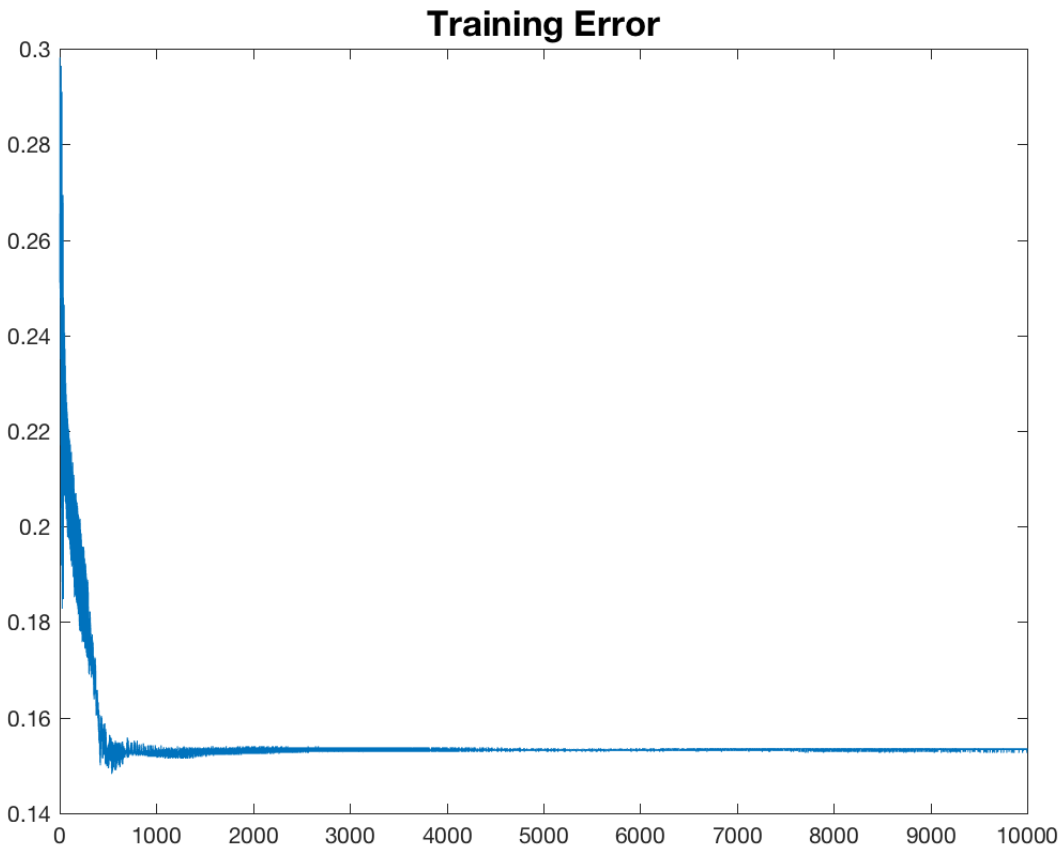


Figure 5: Training error over iterations C=0.1

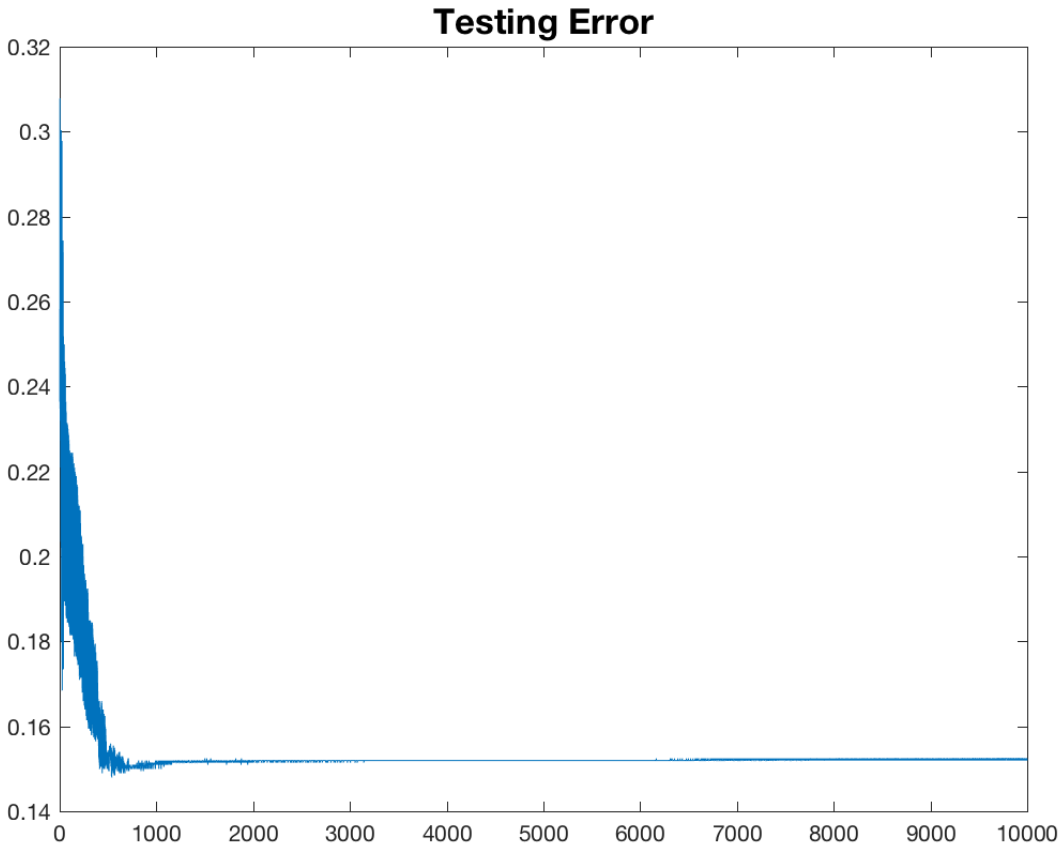


Figure 6: Test error over iterations C=0.1

Solution Strong duality holds between the primal and dual as it holds for convex problems.

$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Replace $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $k(\boldsymbol{x}_i, \boldsymbol{x}_j) + c$

$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j) + c)$$
$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j)) - c\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j$$
$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j)) - c\frac{1}{2} \sum_{i=1}^{m} y_i \alpha_i \sum_{j=1}^{m} y_j \alpha_j$$

Subsitute in above Eqn $\sum_j^m y_j \alpha_j = 0$

$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j)) - c\frac{1}{2} \sum_{i=1}^{m} y_i \alpha_i (0)$$
$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i \alpha_i y_j \alpha_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j))$$

Since adding a constant has no effect on the objective functions, it means that the alpha values won't be affected. Also, $\phi(x)$ is not affected by the adding a constant $c$ to $K$, thus, $w^*$ which is equal to the $\sum_i^m \alpha_i y_i \phi(x_i)$ is also not affected.

# 2 Kernel Ridge Regression

Consider solving the ridge regression problem with a training set

$$S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m) \in \mathbb{R}^d \times \mathbb{R}\} \ .$$

Let $\boldsymbol{w}_\lambda$ be the ridge regression solution with regularization parameter $\lambda$, i.e.

$$\boldsymbol{w}_\lambda = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\arg\min} \ \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2 + \frac{1}{m} \|\boldsymbol{y} - X\boldsymbol{w}\|_2^2$$

In class and in HW2, we derived a closed form expression for $\boldsymbol{w}_\lambda$ in terms of the input matrix $X$ and label vector $\boldsymbol{y}$, defined as

$$X = \begin{bmatrix} — & \boldsymbol{x}_1 & — \\ — & \boldsymbol{x}_2 & — \\ & \vdots & \\ — & \boldsymbol{x}_m & — \end{bmatrix} \in \mathbb{R}^{m \times d} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m \ .$$

This closed form expression turns out to be

$$\boldsymbol{w}_\lambda = (X^\top X + m\lambda I)^{-1} X^\top \boldsymbol{y} \ .$$

We will derive an alternative form for $\boldsymbol{w}_\lambda$ without using the duality nor the Representer Theorem.

## 2.1 Question 5: Alternative Formulation for the Solution of Ridge Regression

First, note that the above closed form expression comes from solving the optimality equation for $\boldsymbol{w}_\lambda$, i.e.

$$(X^\top X + m\lambda I_d)\boldsymbol{w}_\lambda = X^\top \boldsymbol{y},$$

where $I_d$ is the identity matrix of size $d \times d$. The above equation can be rewritten as

$$m\lambda \boldsymbol{w}_\lambda = X^\top (\boldsymbol{y} - X\boldsymbol{w}_\lambda) \ . \tag{8}$$

Now, define $\boldsymbol{\alpha} := \boldsymbol{y} - X\boldsymbol{w}_\lambda$.

(a) Using equation (8) and the definition of $\boldsymbol{\alpha}$, show that $\boldsymbol{\alpha}$ satisfies the following equation:

$$\frac{1}{m\lambda} XX^\top \boldsymbol{\alpha} + \boldsymbol{\alpha} = \boldsymbol{y} \ . \tag{9}$$

Solution
Substitute value of $\alpha$ in (8):

$$m\lambda \boldsymbol{w}_\lambda = X^\top \alpha$$

Substitute value of $\boldsymbol{w}_\lambda$

$$\begin{aligned} m\lambda(y-\alpha)X^{-1} &= X^\top \alpha \\ m\lambda(y-\alpha) &= XX^\top \alpha \\ y - \alpha &= \frac{1}{m\lambda} XX^\top \alpha \\ \frac{1}{m\lambda} XX^\top \boldsymbol{\alpha} + \boldsymbol{\alpha} &= \boldsymbol{y} \end{aligned}$$

(b) Solve equation (9) for $\boldsymbol{\alpha}$ and find an explicit expression of $\boldsymbol{\alpha}$ as a function of $\boldsymbol{y}, X, m$, and $\lambda$.
Solution:

$$\begin{aligned} \frac{1}{m\lambda} XX^\top + \alpha &= y \\ (XX^\top + m\lambda I_m)\alpha &= ym\lambda \\ \alpha &= \frac{ym\lambda}{XX^\top + m\lambda I_m} \\ \alpha &= (XX^\top + m\lambda I_m)^{-1} ym\lambda \end{aligned}$$

(c) Use the expression of $\boldsymbol{\alpha}$ at the previous point to show that the following alternative expression for $\boldsymbol{w}_\lambda$ is valid:

$$\boldsymbol{w}_\lambda = X^\top (XX^\top + m\lambda I_m)^{-1} \boldsymbol{y} \ . \tag{10}$$

Solution:
Substitute value of $\alpha$ in (8):

$$m\lambda \boldsymbol{w}_\lambda = X^\top \alpha$$

Substituing the value of $\alpha$ from above, we get

$$m\lambda \boldsymbol{w}_\lambda = X^\top (XX^\top + m\lambda I_m)^{-1} ym\lambda$$

Thus,

$$\boldsymbol{w}_\lambda = X^\top (XX^\top + m\lambda I_m)^{-1} y$$

## 2.2 Question 6: Implement Kernel Ridge Regression

Let $\boldsymbol{w}_\lambda$ be the solution of the ridge regression problem with regularization parameter $\lambda$ when we use a kernel $k$ corresponding to a feature mapping $\phi$.

(a) Implement a function that finds $\boldsymbol{\alpha}$ for the Kernel Ridge Regression formulation with kernels, using the expression found in the previous question. The prototype must be

```
alpha = train_krr(X, y, lambda, kernel, gamma)
```

where `kernel` is 'gaussian' for Gaussian kernels and 'linear' for linear kernels, `gamma` is the parameter of the Gaussian kernel.

*Hint:* Observe that the matrix $XX^\top$ can be computed using the kernel function $k$.

Solution

```
function [ alpha ] = train_krr(X, y, lambda, kernel, gamma)
   K = [];
   [d,m] = size(X);
   if strcmp(kernel, 'linear')
       K = X'*X;
   else
       if strcmp(kernel,'gaussian')
           K = zeros(m,m);
           for k = 1:m
               for l = 1:m
                   K(k,l) = exp(-gamma*(norm(X(:,k)-X(:,l))^2));
               end
           end
       end
   end
%
    I = eye(m);
    inverse_term = X'*X + m*lambda*I;
    inverse = inv(inverse_term);
    alpha = inverse*y'*m*lambda;
end
```

(b) Given a matrix of test points $Xte \in \mathbb{R}^{m_{te} \times d}$, use equation (10) to implement a Matlab function that computes the prediction using the ridge regression solution expressed in terms of $\boldsymbol{\alpha}$. The prototype must be

```
ypredicted = test_krr(alpha, Xtr, ytr, Xte, lambda, kernel, gamma)
```

where `kernel` is 'gaussian' for Gaussian kernels and 'linear' for linear kernels, `gamma` is the parameter of the Gaussian kernel. Solution

```
function [ ypredicted_test, ypredicted_train ] = test_krr(alpha, Xtr, ytr, Xte, lambda, kernel, gam

[d, m] = size(Xtr);
```

```
w = 1/(m * lambda) * (alpha' * Xtr');
ypredicted_train = sign(w * Xtr);
ypredicted_test= sign(w * Xte);

end
```

(c) Train and test your implementations of Kernel Ridge Regression with the provvided dataset with $\lambda = 2e-05$, Gaussian and linear kernel. Report training and test error. Solution

```
lambda = 0.0002;
alpha = train_krr(Xtr, ytr, lambda,'linear',0.0001);
[ypredicted_test, ypredicted_train] = test_krr(alpha, Xtr, ytr, Xte, lambda, 'linear', 0.001);

Z_test = yte.*ypredicted_test;
accuracy_test = 1- size(Z_test(Z_test<1))/size(Xte,2);
disp(accuracy_test);

Z_training = ytr.*ypredicted_train;
accuracy_train = 1- size(Z_training(Z_training<1))/size(Xtr,2);
disp(accuracy_train);
```

(d) Train and test your implementations of Kernel Ridge Regression with the provvided dataset with $\lambda = 2e-05$, Gaussian and linear kernel. Report training and test error.

Solution:

- Linear Kernel
  Training Error = 15.5%
  Testing Error = 15.39%

- Gaussian Kernel
  Gamma = 0.001
  Training Error = 15.9%
  Testing Error = 14.79%

(e) Repeat the above with $\lambda = 0.002$.

Solution:

- Linear Kernel
  Training Error = 15%
  Testing Error = 15.1%

- Gaussian Kernel
  Gamma = 0.001
  Training Error = 25.2%
  Testing Error = 23.8%

## References