CS-E4830 Kernel Methods in Machine Learning
Assignment 2

Kunal Ghosh, 546247

October 5, 2016

1 (a)

$K^c_{ij} = K_c(x_i, x_j) = \langle \phi_c(x_i), \phi_c(x_j) \rangle$

$\phi_c(x) = \phi(x) - \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)$.

$$\therefore = \langle \phi_c(x_i)\ \phi_c(x_j) \rangle = \left[\phi(x_i) - \frac{1}{N}\sum_{p=1}^{N} \phi(x_p)\right]^T \left[\phi(x_j) - \frac{1}{N}\sum_{q=1}^{N} \phi(x_q)\right] - \textcircled{1}$$

$$= \phi^T(x_i)\phi(x_j) - \frac{1}{N}\left\{\sum_{p=1}^{N} \phi^T(x_p)\right\}\phi(x_j)$$

$$-\frac{1}{N}\phi^T(x_i)\sum_{q=1}^{N}\phi(x_q) + \frac{1}{N^2}\left\{\sum_{p=1}^{N}\phi^T(x_p)\right\}\left\{\sum_{q=1}^{N}\phi(x_q)\right\} - \textcircled{2}$$

$$= K_{ij} - \frac{1}{N}\sum_{p=1}^{N}\phi^T(x_p)\cdot\phi(x_j) - \frac{1}{N}\sum_{q=1}^{N}\phi^T(x_i)\cdot\phi(x_q)$$

$$+ \frac{1}{N^2}\sum_{p=1}^{N}\sum_{q=1}^{N}\phi^T(x_p)\cdot\phi(x_q) - \textcircled{3}$$

$$= K_{ij} - \frac{1}{N}\sum_{p=1}^{N} K_{pj} - \frac{1}{N}\sum_{q=1}^{N} K_{iq} + \frac{1}{N^2}\sum_{p=1}^{N}\sum_{q=1}^{N} K_{pq} - \textcircled{4}$$

Here, in Eq 3. $\frac{1}{N}\sum_{p=1}^{N}\phi^T(x_p)\cdot\phi(x_j)$. for each value of p. $\phi^T(x_p)$ is a vector, of which, we find a dot product with. $\phi(x_j)$. if we have only. 1 "x_p" then we have $\phi^T(x_p)\cdot\phi(x_j) = K_{pq}$. but we have $p \in \{1, \dots, N\}$ which we are averaging over. that's why we have. $\frac{1}{N}\sum_{p=1}^{N} K_{pj}$.

1 (b)

b. $K_c(x_i, t_j) = \langle \phi_c(x_i), \phi_c(t_j) \rangle$ We need to center the data w.r.t the training dataset.

$$= \left(\phi(x_i) - \frac{1}{N}\sum_{p=1}^{N}\phi(x_p)\right)^T \left(\phi(t_j) - \frac{1}{N}\sum_{q=1}^{N}\phi(x_q)\right)$$

$$= \left[\phi^T(x_i) - \frac{1}{N}\sum_{p=1}^{N}\phi^T(x_p)\right]\left[\phi(t_j) - \frac{1}{N}\sum_{q=1}^{N}\phi(x_q)\right]$$

$$= \phi^T(x_i)\phi(t_j) - \frac{1}{N}\phi^T(x_i)\sum_{q=1}\phi(x_q) - \frac{1}{N}\left\{\sum_{p=1}\phi^T(x_p)\right\}\phi(t_j)$$

$$+ \frac{1}{N^2}\sum_{p=1}^{N}\phi^T(x_p)\sum_{q=1}^{N}\phi(x_q)$$

$$= K(x_i, t_j) - \frac{1}{N}\sum_{q=1}^{N}\phi^T(x_i)\phi(x_q) - \frac{1}{N}\sum_{p=1}^{N}\phi^T(x_p)\cdot\phi(t_j) + \frac{1}{N^2}\sum_{p=1}\sum_{q=1}\phi^T(x_p)$$

$$= \underbrace{K(x_i, t_j)}_{\text{from (train, test) Kernel.}} - \underbrace{\frac{1}{N}\sum_{q=1}^{N} K_{iq}}_{\text{Terms from (Train, Train) Kernel.}} - \underbrace{\frac{1}{N}\sum_{p=1}^{N} K(x_p, t_j)}_{\text{from (Train, Test) Kernel.}} + \underbrace{\frac{1}{N^2}\sum_{p=1}^{N}\sum_{q=1}^{N} K_{pq}}_{\text{These terms come from the (Train, Train) Kernel.}}$$

2 (b)

(b) Derivation of dual problem.

Expressing the primal problem in its canonical form.

$$\min_{\omega, \xi_s} \frac{1}{2}\|\omega\|^2 + \frac{c}{2}\sum_{i=1}^{m} \xi_{si}^2.$$

$$\text{s.t.} \quad 1 - (y_i(\omega^T x_i)) - \xi_{si} \leq 0$$

NOTE: here "b" is included as a column of constants in the vector 'x'.

let us define a lagrange variable λ and derive the Lagrangian dua

$$\text{i.e.} \quad \min_{\omega, \xi_s} \frac{1}{2}\|\omega\|^2 + \frac{c}{2}\sum_{i=1}^{m} \xi_{si}^2 + \sum_{i=1}^{m} \lambda_i(1 - y_i(\omega^T x_i) - \xi_{si}) \quad \forall\ i = 1 \ldots m. \quad — ①$$

$$= L(\omega, \xi_s, \lambda) \quad \text{s.t.} \ \lambda_i \geq 0.$$

·Finding the minimal of the lagrangian by finding the derivative w.r.t the primal variables and setting the derivative to zero.

$$\nabla_\omega L(\omega, \xi_s, \lambda) = \omega - \sum_{i=1}^{m} \lambda_i y_i x_i = 0.$$

$$\therefore \omega = \sum_{i=1}^{m} \lambda_i y_i x_i \quad — ②.$$

$$\nabla_{\xi_{si}} L(\omega, \xi_s, \lambda) = C\xi_{si} - \lambda_i = 0 \quad \therefore \xi_{si} = \frac{\lambda_i}{C} \quad — ③$$

$$\text{or } C = \frac{\lambda_i}{\xi_{si}}$$

Substituting ② and ③ in ① ~~is~~ we get.

$$~~\min~~ \quad \frac{1}{2}\left\|\sum_{i=1}^{m} \lambda_i y_i x_i\right\|^2 + \sum_{i=1}^{m} \frac{\lambda_i}{\xi_{si}} \times \frac{\xi_{si}^2}{2} + \sum_{i=1}^{m} \lambda_i\left(1 - y_i\left(\sum_{j=1}^{m} \lambda_j y_j x_j\right)^T x_i -\right.$$

$$= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_{i=1}^{m} \frac{\lambda_i \xi_{si}}{2} + \sum_{i=1}^{m} \lambda_i - \sum_{i=1}^{m} \lambda_i \xi_{si}$$

$$- \sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j x_j^T x_i$$

$$= \sum_{i=1}^{m} \lambda_i - \sum_{i=1}^{m} \frac{\lambda_i \xi_{si}}{2} - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j \underbrace{\langle x_i, x_j \rangle}_{\text{Can be replaced by a Kernel}} \ ; \text{s.t} \ \lambda_i$$

$$= \max_{\lambda_\xi} \sum_{i=1}^{m} \lambda_i\left(1 - \frac{\xi_{si}}{2}\right) - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j K(x_i, x_j). \quad \textbf{(The dual problem)}$$

$$\lambda_i \geq 0$$

2 (a)

a). By definition, the constraint $C > 0$ controls the balance between maximizing the margin & amount of slack needed. also $\lambda_i \geq 0$ $\therefore$ from Eq3.

$$C = \frac{\lambda_i}{\xi_i}$$

$$\Rightarrow \quad \xi_i = \frac{\lambda_i}{C}$$

$$\Rightarrow \frac{\lambda_i \geq 0}{C > 0} \quad \therefore \xi_i \geq 0$$

So, this constraint holds implicitly. Hence, its not needed to be included among the constraints in the original optimization problem.

2 (c)

c). In this problem, let α be the dual variable, the complementary slackness conditions are $\alpha_i \left[y_i(w^T x_i) - 1 + \xi_i \right] = 0$ for $i = 1 \dots m$.

what is the value (or range) of the margin $y_i(w^T x_i + b)$ for $\alpha_i > 0$ and $\alpha_i = 0$.

at the optimal condition. $(\overset{w=}{w^*})$. the complementary slackness condition holds.

i.e. if $\alpha_i > 0$ then $y_i \{ (w^{*T} x_i) - 1 + \xi_i \} = 0$.

$$\boxed{y_i (w^{*T} x_i) = 1 - \xi_i}$$

and when $\alpha_i = 0$ then $y_i (w^{*T} x_i) - 1 + \xi_i < 0$

$$\boxed{y_i (w^{*T} x_i) < 1 - \xi_i}$$

Solution to Question 3

```
function alpha = kernel_ridge_regression(K, y, lambda)
    % K is of size (n,n)
    % y is of size (n,1)
    % lambda is a scalar
    alpha = (lambda * eye(size(K)) + K)\y;
end
```

Solution to Question 4

```
function mse = get_prediction_mse(alpha, K_train_test, y_test)
    % alpha is the dual variable (in this case of the ridge regression kernel).
    % K_train_test is the kernel matrix between the train and the test dataset.
    % y_test is the true labels of the test dataset.
    % ---
    % Ridge regression model's prediction is given by
    % g(x) = y'(K+lambda*I)*k
    % where k is  K(train,test(x))
    % Kernel between training data and the test data
    % for which the prediction g(x) is being calculated.
    y_pred = alpha' * K_train_test;
    mse = mean((y_pred' - y_test).^2);
end
```

We can then encapsulate the `kernel_ridge_regression` method and the `get_prediction_mse` method to a more convenient single method.

```
function [ mset_train,mse_test ] = kernel_ridge_regression_on_dataset( X_train, Y_train,
 ↪  X_test, Y_test, sigma, lambda )
%kernel_ridge_regression_on_dataset Computes the kernel ridge regression
% Computes the kernel ridge regression prediction on a test set and returns
% the mean squared error.
% NOTE: This is not very optimal since we are calculating the kernel everytime
% However, the resulting code is cleaner and the gaussian_kernel function can
% be memoized so as to not compute the kernel if the input is same.

   K_train_train = gaussian_kernel(X_train,X_train,sigma);
   K_train_test = gaussian_kernel(X_train,X_test,sigma);

   alpha = kernel_ridge_regression(K_train_train, Y_train, lambda);
   mse_test = get_prediction_mse(alpha, K_train_test, Y_test);
   mset_train = get_prediction_mse(alpha, K_train_train, Y_train);

end
```

Solution to Question 5 (a)

Applying the `kernel_ridge_regression` code to the UCI forest fire dataset.

```
% =======================
% Predicting forest fires.
% =======================

load data_all.mat

lambdas = 10 .^ (-3:7);
sigmas = 10 .^ (-1:4);

% Matrices to save the value of MSE for train and test data.
sigma_len = size(sigmas,2);
lambda_len = size(lambdas,2);
result_shape = [sigma_len, lambda_len];
mse_test = zeros(result_shape);
mse_train = zeros(result_shape);
mse_cv = zeros(result_shape);

lowest_cv_error = Inf;
best_sigma  = 0;
best_lambda = 0;

K = 5; % K fold cross validation
for sigma_idx = 1:sigma_len
   for lambda_idx = 1:lambda_len
        sigma = sigmas(sigma_idx);
        lambda = lambdas(lambda_idx);

        % we keep saving the errors so that later we don't need to recompute them
        [mse_train(sigma_idx, lambda_idx), mse_test(sigma_idx, lambda_idx)] = ...
            kernel_ridge_regression_on_dataset(X_train, y_train, X_test, y_test, sigma,
             ↪  lambda);

        %-----------------------------------------
        % Calculate K-fold cross validation error
        %-----------------------------------------
        mse_cv(sigma_idx, lambda_idx) = get_k_fold_cv_error_new(K, X_train, y_train,
         ↪  sigmas(sigma_idx), lambdas(lambda_idx));
        cv_error = mse_cv(sigma_idx, lambda_idx)
        %-----------------------------------------
        % check if cross validation error lower than previous lowest
        %-----------------------------------------
        if cv_error < lowest_cv_error
            lowest_cv_error = cv_error;
            best_sigma = sigmas(sigma_idx);
            best_lambda = lambdas(lambda_idx);
        end
   end
end

% -- MSE on the full training dataset using the best parameters
best_sigma_idx = find(sigmas == best_sigma);
best_lambda_idx = find(lambdas == best_lambda);

mse_test_for_best_params = mse_test(best_sigma_idx, best_lambda_idx);
display(mse_test_for_best_params);

% -- Best parameter values from the K-fold cross validation
display(best_sigma);
display(best_lambda);

%% MSE Test 3D plot
```

```
figure();
meshc(log10(lambdas),log10(sigmas),log10(mse_test))
xlabel('Lambda');
ylabel('Sigma');
zlabel('Mean Squared Error');
%% MSE Train 3D plot
hold on;
meshc(log10(lambdas),log10(sigmas),log10(mse_cv))
% meshc(log10(lambdas),log10(sigmas),log10(mse_train))

%title('MSE of Test data (top plane) and Training data (bottom plane), all values in log10
↪  scale.');
```

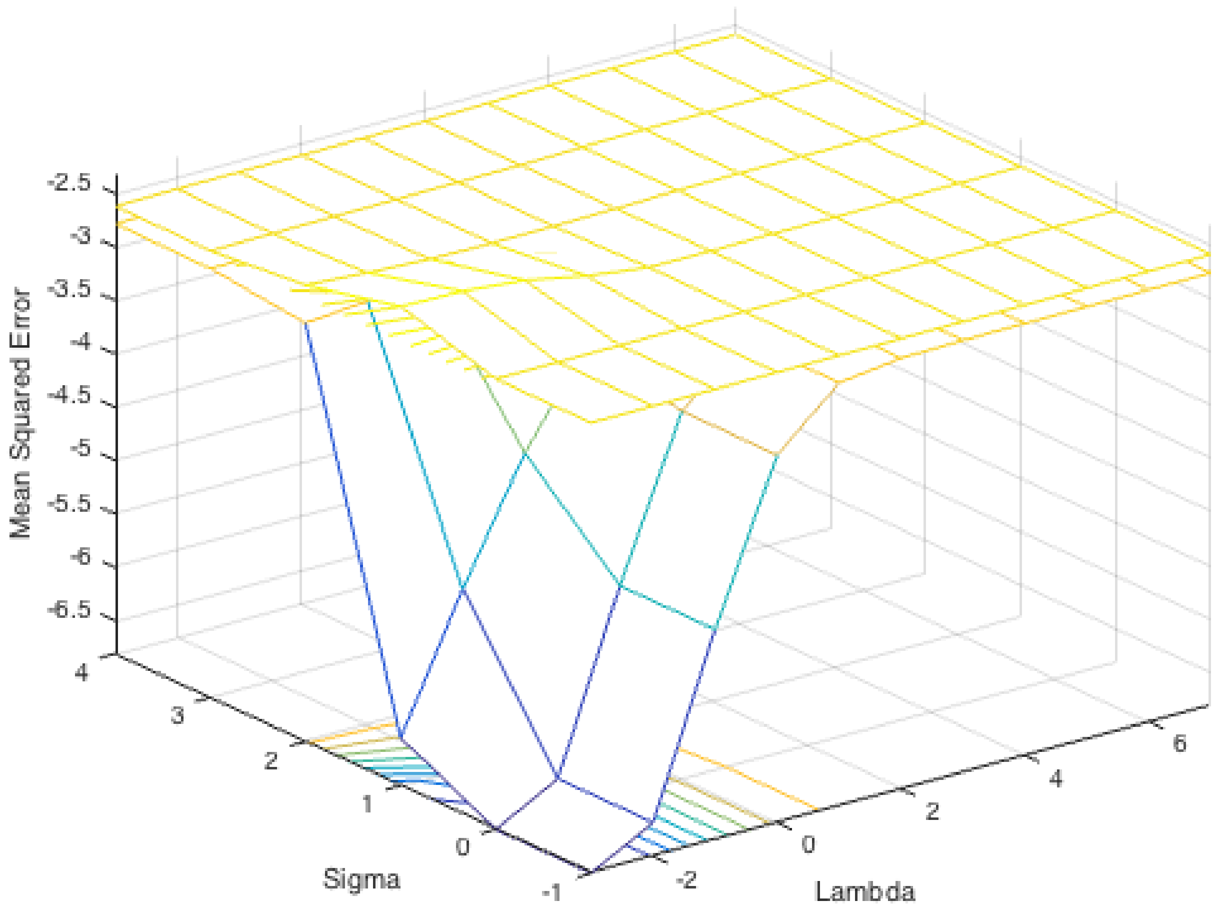

Figure 1: *MSE of Test data (top plane) and Training data (bottom plane)* **all values in log10 scale**. In the plot above, for *sigma* values between 10^{-1} and 10^{2} and *lambda* values between 10^{-3} and 1 *(all pairs)* we notice that the training error (bottom plane) is much lower than the test error (top plane). This would imply that for these values of sigma and lambda the model is overfitting to the training dataset. However, for rest of the values of sigma and lambda the training and test errors don't have a huge gap so we can say that for those parameter values the model generalizes well.

Observation: In the above plot, for a small sigma and lambda values, we see that there is overfitting. But for any given (small) sigma, as we increase the regularization (lambda increased) the overfitting reduces and that's as expected. However, for larger sigma values there is no overfitting even for small regularization and so, increasing the regularization doesn't have any effect.

Solution to Question 5 Part (b)

Parameter combination with lowest cross-validation error. Also reported is the mean-squared-error on the test set using the parameter combination. Note that we have not retrained the model using the best sigma and lambda values since we had already calculated and saved the MSE values for all sigma and lambda pairs. We have just reported the saved MSE values for the best (cross-validated) parameters which should remain same when the model is trained again using these parameters.

```
mse_test_for_best_params =

    0.0031

best_sigma =

    10

best_lambda =

     1
```

Code to compute the *k-fold* cross-validation error

The function below is being invoked in the script defined in `Solution to Question 5 part (a)`

```
function [ error ] = get_k_fold_cv_error_new(K, X_train, y_train, sigma, lamda)
% get_k_fold_cv_error Calculates the K fold cross validation

num_datapoints = size(X_train,1);
shuffle_idxs = randperm(num_datapoints);

% -- datapoints in each subset
partition_size = floor(num_datapoints / K);
start_idx = 1;

% -- initialize an empty list of mean_squared_errors
mserrors = zeros(1,K);

for k = 1:K-1
    % -- Calculations to get the Train and Validation split
    end_idx = min(start_idx+partition_size-1,length(shuffle_idxs)); % ending index of
    ↪  validation set
    validation_set_idxs = (start_idx:end_idx);

    valid_set_X = X_train(validation_set_idxs,:);
    valid_set_Y = y_train(validation_set_idxs,:);

    train_set_idxs = setdiff(shuffle_idxs, validation_set_idxs);
    train_set_X = X_train(train_set_idxs,:);
    train_set_Y = y_train(train_set_idxs,:);
    %------- Actual Training/Testing happens here.
    [~, mserrors(1,k)] = ...
        kernel_ridge_regression_on_dataset(train_set_X, train_set_Y, valid_set_X,
        ↪  valid_set_Y, sigma, lamda);
    %-------
    start_idx = end_idx+1; % starting index of next validation set
end

% -- Calculating the last Kth-validation set now.
validation_set_idxs = (start_idx:num_datapoints);

train_set_idxs = setdiff(shuffle_idxs, validation_set_idxs);
train_set_X = X_train(train_set_idxs,:);
train_set_Y = y_train(train_set_idxs,:);

%------- Actual Training/Testing of the Kth-validation set happens here.
[~, mserrors(1,K)] = ...
    kernel_ridge_regression_on_dataset(train_set_X, train_set_Y, valid_set_X, valid_set_Y,
     ↪  sigma, lamda);

%------- Calculate the Average MSE for the K validation sets.
error = mean(mserrors);
end
```