Pattern Recognition and Machine Learning

Assignment - 03

July 21, 2025

Name : Rohan Baghel
Student ID: 202116011

About

In this,we will study regularized least squares kernel regression model with the RBF kernel and implement it using Matlab.
kernel K(x,y) = exp(- ($\|x - y\|2/\sigma$).

kernel values are used to derive weights to predict outputs from given inputs. Steps involved to calculate weights and finally to use them in predicting output variable, y from predictor variable, x is explained in detail in the following sections. Let's start with an example to clearly understand how kernel regression works.

Kernel regression Model

Kernel regression is a non-parametric technique to estimate the conditional expectation of a random variable. The objective is to find a non-linear relation between a pair of random variables X and Y.

File1

I this file we are computing all the operations
The fundamental calculation behind kernel regression is to estimate weighted sum of all observed y values for a given predictor value, xi. Weights are nothing but the kernel values, scaled between 0 and 1, intersecting the line perpendicular to x-axis at given xi

- Training points = 200
- Testing points = 400

Q1-
Behavior of the kernel parameter σ and regularization on prediction in terms of RMSE.
Q2-
RMSE for regularized polynomial regression models and regularized least squares kernel regression model using the gradient descent method.
Q3-
RMSE for regularized polynomial regression models and regularized least squares kernel regression model using the k-mini batch stochastic gradient descent method.

```
clear
clc

%-------------------------------------------------------------------------%\
%Generating Random Numbers for Sigmodial Function (Training set and Testing set)

%Testing set
test_X = sort(rand(200,1));
nrm_x =sqrt(test_X.^2+(test_X.^2).^2);
test_Y = sin(2*pi*nrm_x) + randn(200,1)*0.25;

% Training set
train_X = sort(rand(400,1));
NormTr =sqrt(train_X.^2+(train_X.^2).^2);
train_Y = sin(2*pi*NormTr) + randn(400,1)*0.25;


%-------------------------------------------------------------------------%
% Q1 - Obtain the prediction on testing set and compute the RMSE


%for regularized least squares kernel regression model with the RBF kernel K(x,y...
    ) = exp(- (||x-y|| 2 /  )
% For sigma = 1

sigma = 0.00002;
kernel_Mat_test = exp(-((((test_X)-(test_Y')).^2+((test_X-test_Y').^2)))/(sigma)...
    );


% Calculating the RMSE of Kernel
```

```
y_cap = kernel_Mat_test*test_Y;
Y_W = test_Y - y_cap;
RMSE_kernel = sqrt(mean(Y_W.^2));
disp("RMSE value of kernel")
disp(RMSE_kernel)

% Calculating the RMSE of regularization
lambda = 0.01;
A = [test_X, ones(200,1)];
coeff = inv(A'*A+lambda*eye(size(A'*A)))*(A'*test_Y);
pred_test= A*coeff;
RMSE_lambda = sqrt(mean((pred_test-test_Y).^2));
disp("RMSE value of regularization   ")
disp(RMSE_lambda)

%-------------------------------------------------------------------------%
%Plot Graph

kernel_Mat_test = exp(-((((test_X)-(test_Y')).^2+((test_X-test_Y').^2)))/(1));
ax1 = nexttile;
plot((test_X- test_Y'),kernel_Mat_test);
title(ax1,'Sigma = 1')

kernel_Mat_test = exp(-((((test_X)-(test_Y')).^2+((test_X-test_Y').^2)))/(0));
ax2 = nexttile;
plot((test_X-test_Y'),kernel_Mat_test);
title(ax2,'Sigma = 0')

kernel_Mat_test = exp(-((((test_X)-(test_Y')).^2+((test_X-test_Y').^2)))/(0.02))...
    ;
ax3 = nexttile;
plot((test_X-test_Y'),kernel_Mat_test);
title(ax3,'Sigma = 0.02')

%-------------------------------------------------------------------------%
%Q2

% Regularized polynomial regression models for order M =1,2,and 5
%M=1;
lambda = 0.01;
mat1 = [test_X, ones(200,1)];
C_1 = inv(mat1'*mat1+lambda*eye(size(mat1'*mat1)))*(mat1'*test_Y);
pred_val1= mat1*C_1;
RMSE_test1 = sqrt(mean(pred_val1-test_Y).^2 );


% M=2
mat2 = [test_X,test_X.^2 ones(200,1)];
C_2 = inv(mat2'*mat2+lambda*eye(size(mat2'*mat2)))*(mat2'*test_Y);
pred_val2= mat2*C_2;
RMSE_test2 = sqrt(mean(pred_val2-test_Y).^2 );


% M=5
mat3 = [test_X,test_X.^2,test_X.^3,test_X.^4,test_X.^5,ones(200,1)];
C_5 = inv(mat3'*mat3+lambda*eye(size(mat3'*mat3)))*(mat3'*test_Y);
pred_val3= mat2*C_2;
RMSE_test5 = sqrt(mean(pred_val3-test_Y).^2 );

```

```
88
89 % Regularized least squares kernel regression model using the
90 % gradient descent method.
91 sigma = 0.0002;
92
93 add_col = ones(200,1);
94 H = [kernel_Mat_test add_col];
95
96 U = H'*test_Y;
97 G_D = (0.002*U) - H'*(test_Y - H*U);
98
99 U1 = U - sigma*G_D;
100 RMSE_GD = sqrt(mean((U1-U).^2));
101 disp("RMSE of Gradient descent method")
102 disp(RMSE_GD)
103
104 %-------------------------------------------------------------------------%
105 %Q3
106
107 % Regularized polynomial regression models for order M =1,2,and 5
108 %M=1;
109 lambda = 0.01;
110 mat1 = [test_X, ones(200,1)];
111 C_1 = inv(mat1'*mat1+lambda*eye(size(mat1'*mat1)))*(mat1'*test_Y);
112 pred_val1= mat1*C_1;
113 RMSE_test1 = sqrt(mean(pred_val1-test_Y).^2 );
114
115
116 % M=2
117 mat2 = [test_X,test_X.^2 ones(200,1)];
118 C_2 = inv(mat2'*mat2+lambda*eye(size(mat2'*mat2)))*(mat2'*test_Y);
119 pred_val2= mat2*C_2;
120 RMSE_test2 = sqrt(mean(pred_val2-test_Y).^2 );
121
122
123 % M=5
124 mat3 = [test_X,test_X.^2,test_X.^3,test_X.^4,test_X.^5,ones(200,1)];
125 C_5 = inv(mat3'*mat3+lambda*eye(size(mat3'*mat3)))*(mat3'*test_Y);
126 pred_val3= mat2*C_2;
127 RMSE_test5 = sqrt(mean(pred_val3-test_Y).^2 );
128
129
130
131 % Regularized least squares kernel regression model using the k-mini batch ...
      stochastic gradient descent method.
132
133 K = 5;
134 P=randi([1, 200], K, 1);
135 H_T = H';
136 a1 =  H_T(P, :);
137 S_G_D = (0.002*U) - A(P,1)'*(test_Y(P, :) - H(P, :)*U);
138
139 U2 = U - sigma*S_G_D;
140 RMSE_SGD = sqrt(mean((U2-U).^2));
141 disp("RMSE of Stochastic Gradient descent method")
142 disp(RMSE_SGD)
```

Output of Regularized polynomial regression model

For Testing set

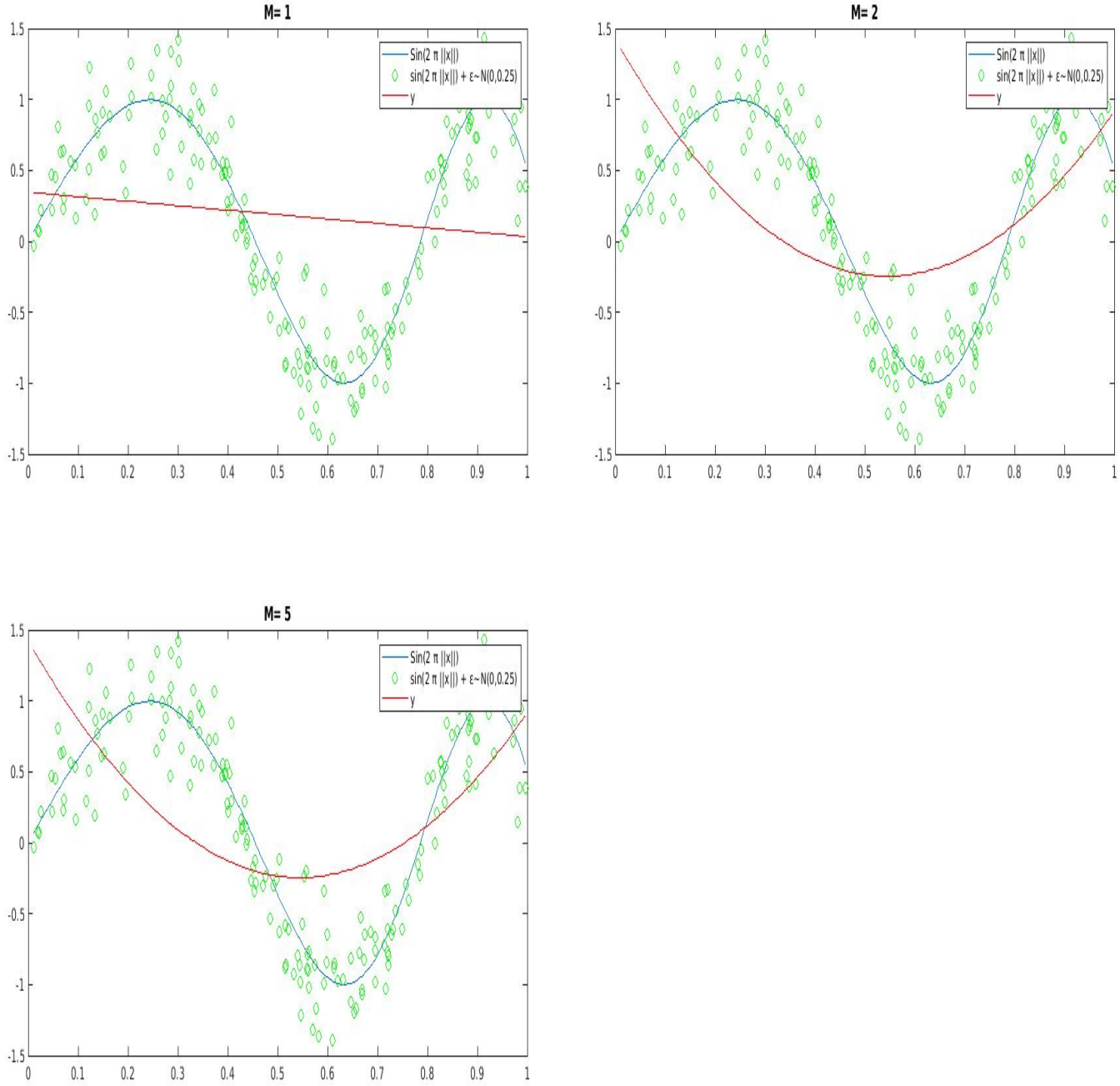

Figure 1: for Training set

Difference in changing eta values

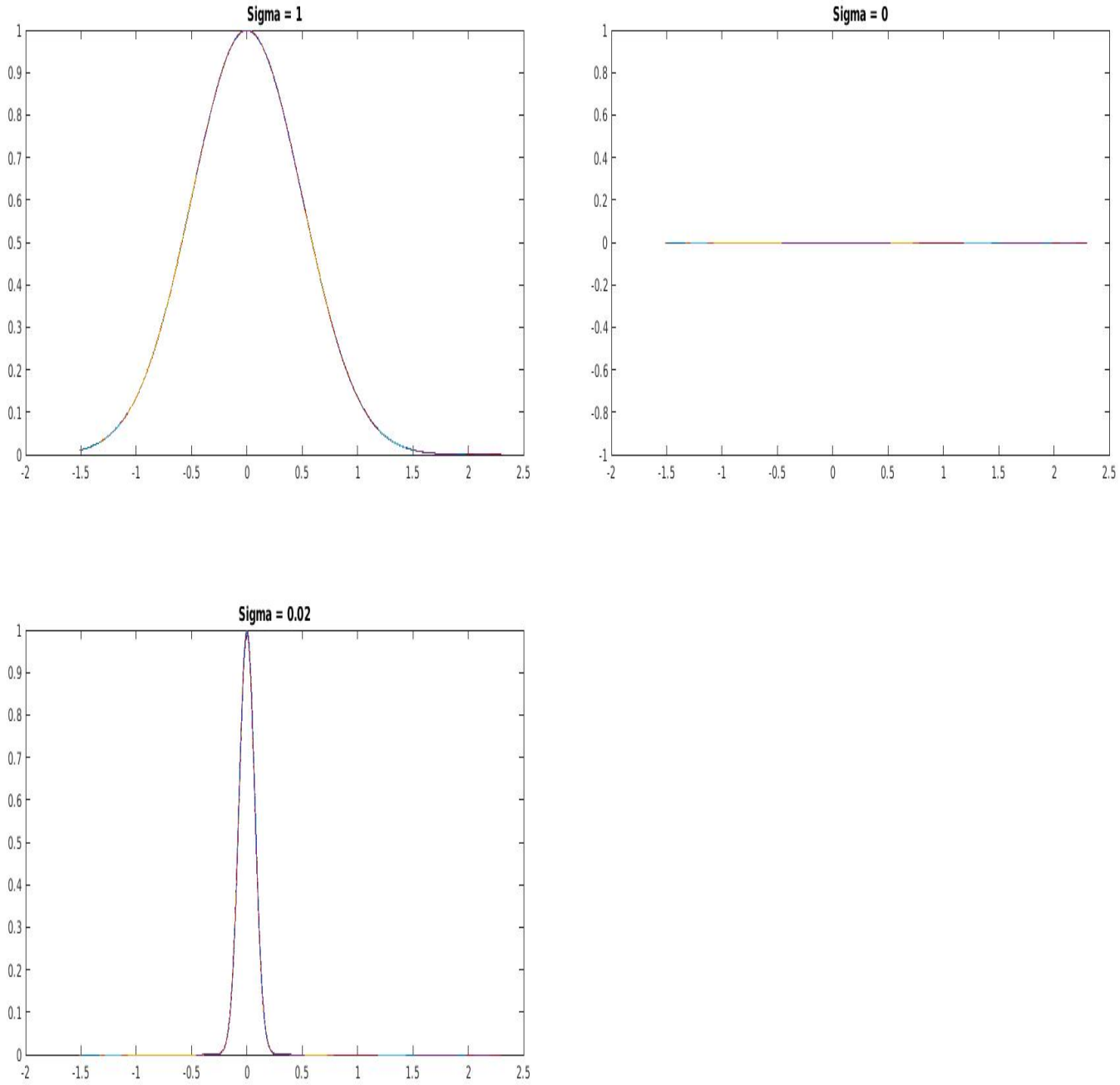

Figure 2: Eta Values

The Graph shows the behaviour of eta for different values.

Observation

Q4:
Gradient descent

- More stable convergence and error gradient than Stochastic Gradient descent.
- Embraces the benefits of vectorization.
- produces no noise and gives a lower standard error
- It produces an unbiased estimate of the gradients

Stochastic Gradient descent

- It is computationally fast as only one sample is processed at a time.
- Due to frequent updates, the steps taken towards the minima of the loss function have oscillations that can help to get out of the local minimums of the loss function

Gradient descent Slower learn the data.
RMSE value of Stochastic Gradient descent is less than the RMSE value of Gradient descent.
Hence Stochastic Gradient descent converge more accurate values.

This regression will provide a smooth (and exact, if there's no error) interpolation between the observed data points; how and how sharply the function changes between the data points is informed by the choice of kernel and kernel hyper-parameters.

Q5:
Effect of chosen step length eta regarding the convergence to actual solution in case of all used algorithms.

- Fewer oscillations and noisy steps are taken towards the global minima of the loss function because of updating the parameters by computing the average of all the training samples rather than the value of a single sample.
- It produces a more stable gradient descent convergence and stable error gradient than stochastic gradient
- For a small data subset, we get a worse estimate of the gradient but the algorithm computes the solution faster.
- Fewer oscillations and noisy steps are taken towards the global minima of the loss function because of updating the parameters by computing the average of all the training samples rather than the value of a single sample.