The University of British Columbia

CPSC 340

Machine Learning and Data Mining

Homework #3

Student name:

ALI SIAHKOOHI

Student ID:

84264167

Instructor:

Professor Schmidt

Date Submitted:

October 19th, 2016

Question #1
Part 1.1

1.

$$x^T x = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = 4 + 9 = 13.$$

2.

$$\|x\|^2 = \|x\|_2^2 = \left(\sqrt{4+9}\right)^2 = 13.$$

3.

$$x^T (x + \alpha y) = x^T x + \alpha x^T y = 13 + 5 \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = 13 + 5(2 + 12) = 83.$$

4.

$$Ax = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2+6 \\ 4+9 \\ 6+6 \end{bmatrix} = \begin{bmatrix} 8 \\ 13 \\ 12 \end{bmatrix}$$

5.

$$z^T Ax = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 \\ 13 \\ 12 \end{bmatrix} = 16 + 0 + 12 = 28.$$

6.

$$A^T A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1+4+9 & 2+6+6 \\ 2+6+6 & 4+9+4 \end{bmatrix} = \begin{bmatrix} 14 & 14 \\ 14 & 17 \end{bmatrix}$$

7. True

$$yy^T y = y(y^T y) = y\|y\|^2 = \|y\|^2 y.$$

8. True

$x^T A^T (Ay + Az) = x^T A^T Ay + x^T A^T Az$

And since $x^T A^T Az$ is a scalar therefore $x^T A^T Az = \left(x^T A^T Az\right)^T = z^T A^T Ax.$

$\Rightarrow x^T A^T (Ay + Az) = x^T A^T Ay + z^T A^T Ax.$

9. False

$A^T (B + C) = A^T B + A^T C$

Since $A^T B$ and $A^T C$ are not necessarily symmetric.

10. False

Matrix dimensions are not the same on left and right side of equality!

11. True

$\left(A + BC\right)^T = A^T + C^T B^T$ because $\left(BC\right)^T = C^T B^T$

12. False

$(x - y)^T (x - y) = (x^T - y^T)(x - y) = x^T x - x^T y - y^T x + y^T y = \|x\|^2 - 2x^T y + \|y\|^2$

13. True

$(x - y)^T (x + y) = (x^T - y^T)(x + y) = x^T x + x^T y - y^T x - y^T y = \|x\|^2 - \|y\|^2$

Part 1.2

1.

$\sum_{i=1}^{n} \left| w^T x_i - y_i \right| = \|Xw - y\|_1$

2.

$\max_{i \in \{1,2,\ldots,n\}} \left| w^T x_i - y_i \right| + \frac{\lambda}{2} \sum_{j=1}^{n} w_j^2 = \|Xw - y\|_\infty + \frac{\lambda}{2} \|w\|^2$

3.

$$\sum_{i=1}^{n} z_i \left(w^T x_i - y_i \right)^2 + \lambda \sum_{j=1}^{d} \left| w_j \right| = \left(Xw - y \right)^T Z \left(Xw - y \right) + \lambda \left\| w \right\|_1$$

Part 1.3

1.

$$f(w) = \frac{1}{2} \left\| w - v \right\|^2 = \frac{1}{2} (w - v)^T (w - v) = \frac{1}{2} (w^T w - v^T w - w^T v + v^T v)$$

$$\Rightarrow \nabla f(w) = \frac{1}{2} \left(w^T (I + I) - v^T - v^T + 0 \right) = w^T - v^T \Rightarrow w = v.$$

2.

$$f(w) = \frac{1}{2} \left\| w \right\|^2 + w^T X^T y = \frac{1}{2} w^T w + w^T X^T y$$

$$\Rightarrow \nabla f(w) = \frac{1}{2} w^T (I + I) + y^T X = 0 \Rightarrow w = -X^T y.$$

3.

$$f(w) = \frac{1}{2} \left\| Xw - y \right\|^2 + \frac{1}{2} w^T \Lambda w = \frac{1}{2} \left(Xw - y \right)^T \left(Xw - y \right) + \frac{1}{2} w^T \Lambda w$$

$$\Rightarrow \nabla f(w) = X^T (Xw - y) + \frac{1}{2} w^T \left(\Lambda^T + \Lambda \right) = \left(X^T (Xw - y) + \Lambda w \right)^T$$

$$\Rightarrow \left(X^T X + \Lambda \right) w = X^T y \Rightarrow w = \left(X^T X + \Lambda \right)^{-1} X^T y.$$

4.

$$f(w) = \frac{1}{2} \sum_{i=1}^{n} z_i \left(w^T x_i - y_i \right)^2 = \frac{1}{2} \left(Xw - y \right)^T Z \left(Xw - y \right) = \frac{1}{2} \left(w^T X^T - y^T \right) \left(ZXw - Zy \right)$$

$$\Rightarrow f(w) = \frac{1}{2} \left(w^T X^T ZXw - w^T X^T Zy - y^T ZXw + y^T Zy \right)$$

$$\Rightarrow \nabla f(w) = \frac{1}{2} \left(2w^T X^T ZX - 2y^T ZX \right)$$

$$\nabla f(w) = 0 \Rightarrow \left(X^T ZX \right) w = X^T Zy \Rightarrow w = \left(X^T ZX \right)^{-1} X^T Zy.$$

Question #2
Part 2.1

```
function [model] = leastSquaresBias(X,y)

X = [ones(size(X, 1), 1) X];
% Solve least squares problem
w = (X'*X)\X'*y;

model.w = w;
model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
Xhat = [ones(size(Xhat, 1), 1) Xhat];
w = model.w;
yhat = Xhat*w;
end
```

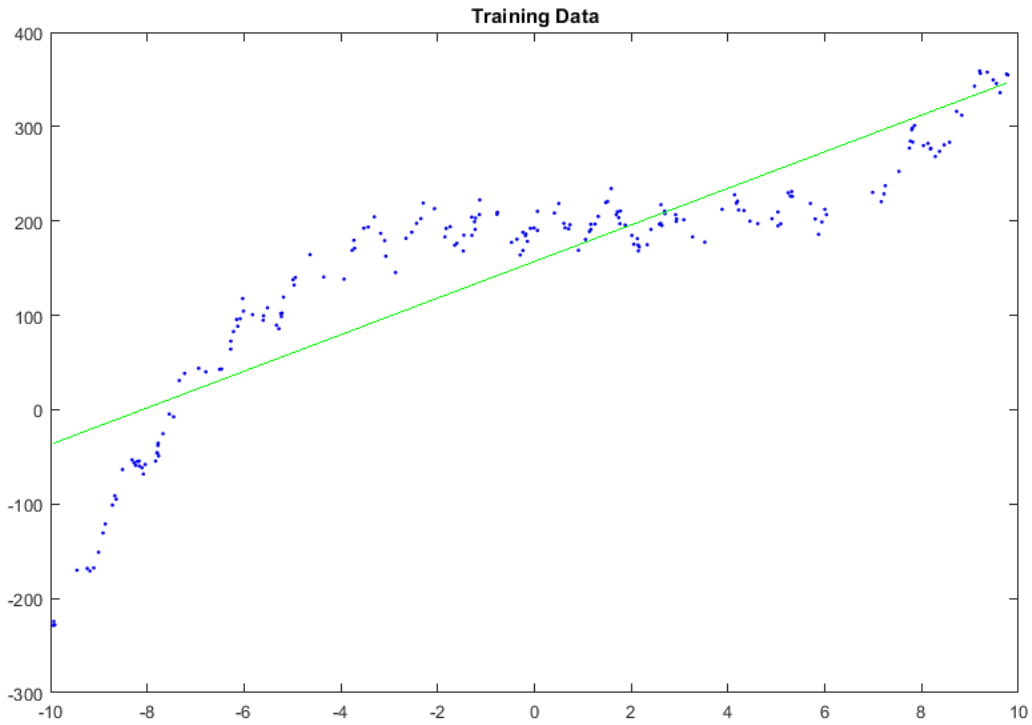

Figure (2 – 1)

Par 2.2

```
function X = polyBasis(x, p)

X = ones(length(x), p+1);
for i = 2:p+1
    X(:, i) = x.^(i-1);
end
end
```

```
function [model] = leastSquaresBasis(x, y, p)

X = polyBasis(x, p);
% Solve least squares problem
w = (X'*X)\X'*y;

model.w = w;
model.predict = @predict;

end

function [yhat] = predict(model,xhat)
w = model.w;
p = length(w)-1;
Xhat = polyBasis(xhat, p);

yhat = Xhat*w;
end
```

Training and test error for p = 0
Training error = 15480.52
Test error = 14390.76
Training and test error for p = 1
Training error = 3551.35
Test error = 3393.87
Training and test error for p = 2
Training error = 2167.99
Test error = 2480.73
Training and test error for p = 3
Training error = 252.05
Test error = 242.80
Training and test error for p = 4
Training error = 251.46
Test error = 242.13
Training and test error for p = 5
Training error = 251.14

Test error = 239.54
Training and test error for p = 6
Training error = 248.58
Test error = 246.01
Training and test error for p = 7
Training error = 247.01
Test error = 242.89
Training and test error for p = 8
Training error = 241.31
Test error = 245.97
Training and test error for p = 9
Training error = 235.76
Test error = 259.30
Training and test error for p = 10
Training error = 235.07
Test error = 256.30

As it can be seen, as p increases, training error decreases. Test error decreases but after p = 4, test error starts to increase, which means for p>4 we are over fitting.

Part 2.3

I tried to fit a polynomial $+ \sin(\frac{2\pi x}{1.258})$ to data set. I estimated the frequency by subtracting the data by the polynomial from the last part, and then by taking a fourier transform, I picked the frequency with maximum magnitude. After training I got these result for p = 0:10

Training and test error for p = 0
Training error = 15233.51
Test error = 14217.55
Training and test error for p = 1
Training error = 3423.26
Test error = 3285.91
Training and test error for p = 2
Training error = 2055.47
Test error = 2376.49
Training and test error for p = 3
Training error = 48.07
Test error = 50.14
Training and test error for p = 4
Training error = 48.06
Test error = 49.98
Training and test error for p = 5
Training error = 48.03

Test error = 50.02
Training and test error for p = 6
Training error = 47.84
Test error = 50.45
Training and test error for p = 7
Training error = 46.92
Test error = 51.70
Training and test error for p = 8
Training error = 46.85
Test error = 51.74
Training and test error for p = 9
Training error = 46.83
Test error = 51.54
Training and test error for p = 10
Training error = 46.79
Test error = 51.63

So my model is going to be a polynomial with p=4 plus $\sin(\frac{2\pi x}{1.258})$ with some amplitude which I found by training the model on data set. I choose p=4 based on the result above. Because it has the lowest test error.

Question #3

Part 3.1

The problem is with choosing the training and validation set! Since the model goes to zero apart from data points, it's better not too choose the first half of the points for training, and the second half for validation. I would choose them in a random order like below:

```
p = randperm(n);
Xtrain = X(p(1:n/2),:);
ytrain = y(p(1:n/2));
Xvalid = X(p(n/2+1:end),:);
yvalid = y(p(n/2+1:end),:);
```

Part 3.2

This is the output:

Value of sigma that achieved the lowest validation error was 1.000e+00
Refitting on full training set...
Training error = 39.49
Test error = 71.17

My MATLAB code:

```
% Clear variables and close figures
clear all
close all

% Load data
load basisData.mat % Loads X and y
[n,d] = size(X);


% Find best value of RBF kernel parameter,
%   training on the train set and validating on the validation set
minErr = inf;
for sigma = 2.^[-15:15]
    validError = 0;
    for CV = 1:10
        % Train on the training set
        [Train, Test] = crossvalind('HoldOut', size(X,1), .1);
        model = leastSquaresRBF(X(Train, :),y(Train, :),sigma);

        % Compute the error on the validation set
        yhat = model.predict(model,X(Test, :));
        validError = validError + sum((yhat - y(Test, :)).^2)/(n/2);
%         fprintf('Error with sigma = %.3e = %.2f\n',sigma,validError);
    end
    validError = validError/10;
    % Keep track of the lowest validation error
    if validError < minErr
        minErr = validError;
        bestSigma = sigma;
    end
end
fprintf('Value of sigma that achieved the lowest validation error was
%.3e\n',bestSigma);

% Now fit the model based on the full dataset.
fprintf('Refitting on full training set...\n');
model = leastSquaresRBF(X,y,bestSigma);

% Compute training error
yhat = model.predict(model,X);
trainError = sum((yhat - y).^2)/n;
fprintf('Training error = %.2f\n',trainError);

% Finally, report the error on the test set
t = size(Xtest,1);
yhat = model.predict(model,Xtest);
testError = sum((yhat - ytest).^2)/t;
fprintf('Test error = %.2f\n',testError);

% Plot model
figure(1);
plot(X,y,'b.');
title('Training Data');
hold on
Xhat = [min(X):.1:max(X)]'; % Choose points to evaluate the function
yhat = model.predict(model,Xhat);
plot(Xhat,yhat,'g');
```

```
ylim([-300 400]);
```

Part 3.3

(a) Forming the normal equations + solving the linear system $O(nd^2 + d^3)$

(b) Forming the normal equations + solving the linear system $O(nn^2 + n^3) = O(n^3)$

For classification:

(a) For linear: Multiplying X to w : $O(td)$

(b) For RBF: Multiplying Z to w: $O(t^2)$

When are RBF's cheaper to train? Obviously when d is very very larger than n. In other words when d>>n RBF's are cheaper to train.

When RBF's are cheaper to test? Bsaed on the above O-notations, when d<t!

Part 3.4

In both cases, we would need different σ's. because when the period alters, the distance between points in different places are not equal. Also when we don't have evenly sampled data along on dimension, as like the previouse case, different parts of data need to have different σ's.
I would propose to fit different models in different areas of data.

Question #4

Part 4.1

Below is the updated plot:

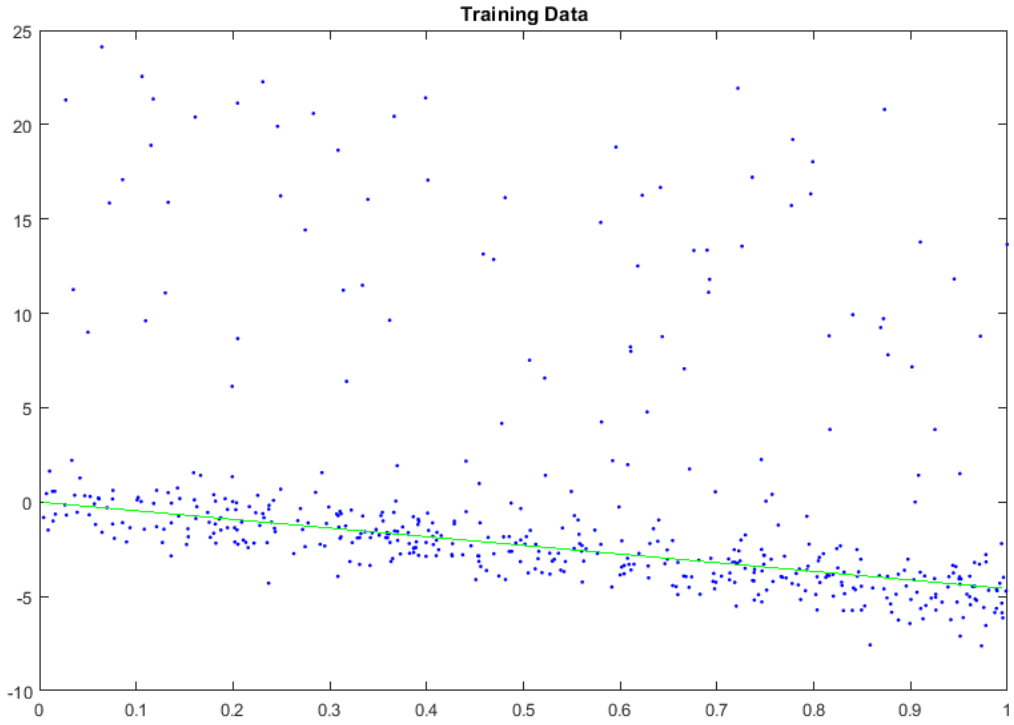

Figure 4 – 1

Here is my function:

```
function [model] = weightedLeastSquares(X,y,z)
Z = diag(z);

% Solve least squares problem
w = (X'*Z*X)\X'*Z*y;

model.w = w;
model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
w = model.w;
yhat = Xhat*w;
end
```

Part 4.2

$$f(w) = \sum_{i=1}^{n} \log(e^{w^T x_i - y_i} + e^{-w^T x_i + y_i})$$

$$\Rightarrow \nabla f(w) = \sum_{i=1}^{n} \nabla\left(\log(e^{w^T x_i - y_i} + e^{-w^T x_i + y_i})\right) = \sum_{i=1}^{n} \frac{\nabla\left(e^{w^T x_i - y_i} + e^{-w^T x_i + y_i}\right)}{e^{w^T x_i - y_i} + e^{-w^T x_i + y_i}}$$

$$= \sum_{i=1}^{n} \frac{\nabla\left(w^T x_i - y_i\right)e^{w^T x_i - y_i} + \nabla\left(-w^T x_i + y_i\right)e^{-w^T x_i + y_i}}{e^{w^T x_i - y_i} + e^{-w^T x_i + y_i}} = \sum_{i=1}^{n} \frac{x_i{}^T e^{w^T x_i - y_i} - x_i{}^T e^{-w^T x_i + y_i}}{e^{w^T x_i - y_i} + e^{-w^T x_i + y_i}}$$

Part 4.3

The plot obtained is in the next page:

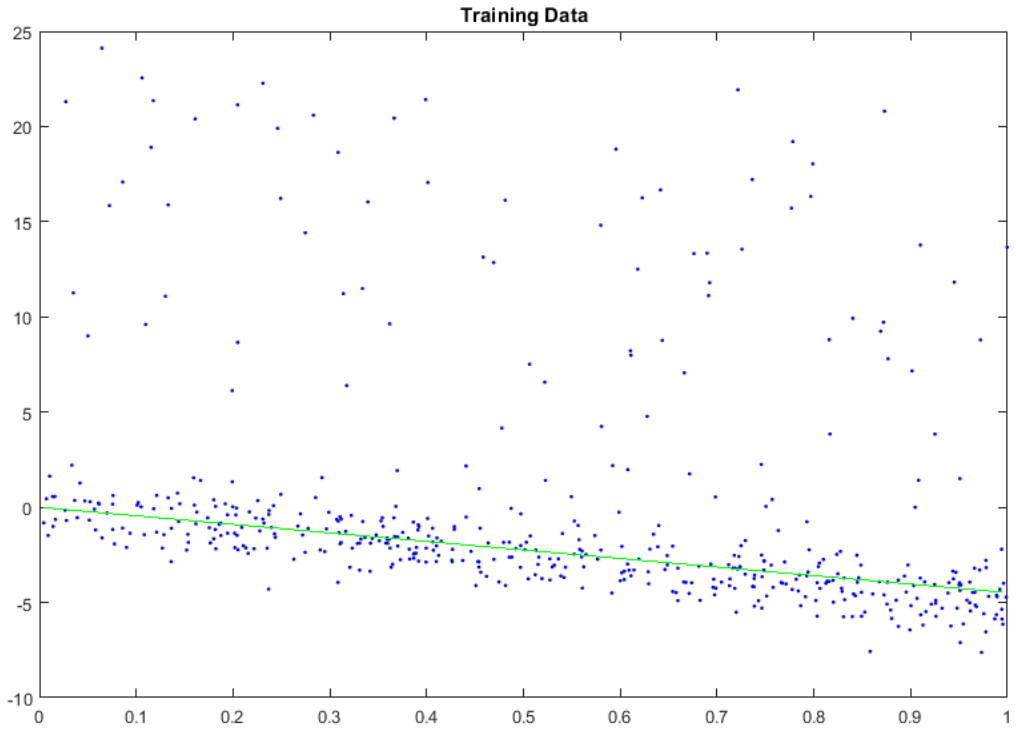

Figure 4 – 2

Here is my MATLAB code:

```
function [f,g] = funObj(w,X,y)

f = 0;
for i = 1:size(X, 1)
    f = f + log(exp(w.*X(i, :)-y(i))+exp(-w.*X(i, :)+y(i)));
end
g = 0;
for i = 1:size(X, 1)
    g = g + (X(i, :)'.*exp(w.*X(i, :)-y(i))-X(i, :)'.*exp(-w.*X(i,
:)+y(i)))/...
        (exp(w.*X(i, :)-y(i))+exp(-w.*X(i, :)+y(i)));
end

end
```

And here is the output:

```
User and numerical derivatives agree.
Backtracking...
Backtracking...
   4    2.00242e-02    1.91178e+03    1.92014e+02
   5    1.37105e-02    1.70700e+03    8.84227e+01
   6    1.49459e-02    1.64481e+03    7.30912e+00
   7    1.19743e-02    1.64451e+03    1.81389e+00
   8    1.18688e-02    1.64449e+03    1.61200e-02
   9    1.18933e-02    1.64449e+03    3.32103e-05
Problem solved up to optimality tolerance
```