UNIVERSITY OF BIRMINGHAM

Machine Learning (Extended)

Computer Based Test 1

Linear Regression by Least Squares

October 21, 2016

Thomas Brereton 1708846

Table of Content

1 Introduction

In this computer based test we are asked to perform linear regression by least squares on a given data set. The dataset analysed is the Olympic men's running time (100m and 400m) for a given range of years.

In Task 1 we determine which polynomial function is the best fit for the 400m data. A common validation method is used to ensure the model is competent. In Task 2 we determine the best value of the regularisation factor, λ, for the polynomial functions of order 1 and 4.

In short, the results of Task 1 illustrate a polynomial function with order 4 best fits the Olympics men's 400m data. In Task 2, the results show a regularisation factor, λ, with value 0, gives the best predictive performance for both polynomial functions on the Olympic men's 100m data.

The analysis was performed in Matlab, and the code listings can be found in Appendices A and B.

2 Task 1

In Task 1, we are asked to analyse the Olympics men's 400m data. We must fit the **n** order polynomial functions to this data, where **n** is 1 to 4, and use 10-fold cross-validation to choose the "best" value of **n**. Refer to Appendix A for the Matlab code used for analysis.

From Figure 1, it is intuitive that a polynomial function with order of 4 is the best fit. However, in order to determine the "best" value of n, "best" must be given a concrete meaning for this task. "Best" is the minimum mean squared loss, where loss is the difference between predicted and observed labels. Ideally, both cross-validation (CV) and training loss are considered but that is not always the case.

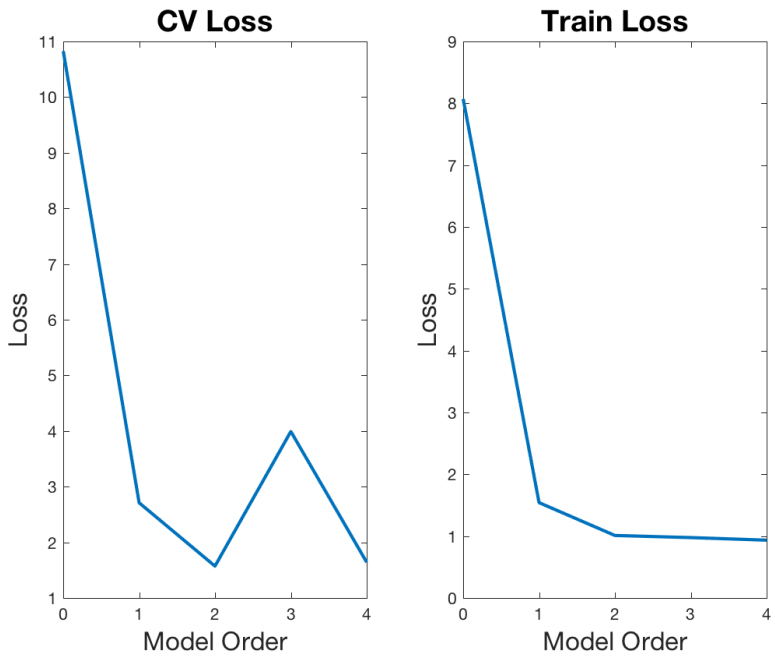
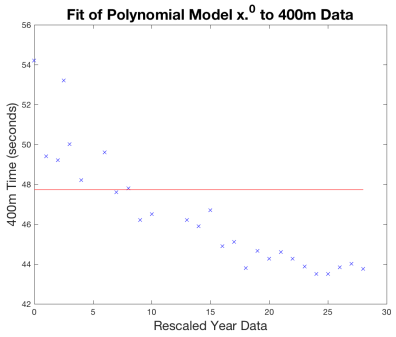


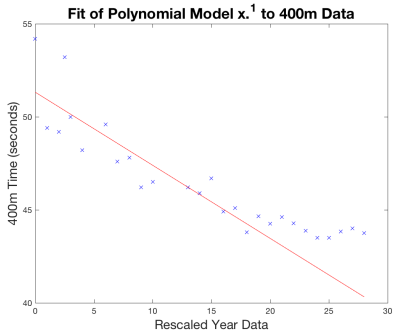
Figure 1: Comparison of the CV and Train loss for polynomial models of order **n**, where **n** = 0 to 4.

Figure 1 shows if only the CV loss is considered, an order of 2 is slightly better, but if only training loss is considered, an order of 4 is best. Moreover, if both CV and training loss are considered, an order of 4

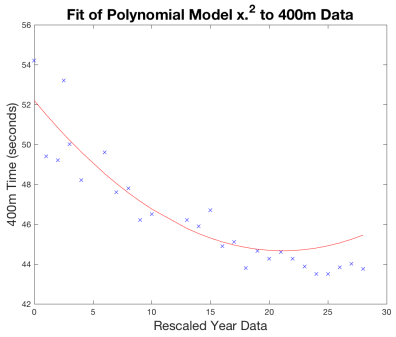
is best. When both are considered, the minimum mean of the CV and training loss determines the best fit.



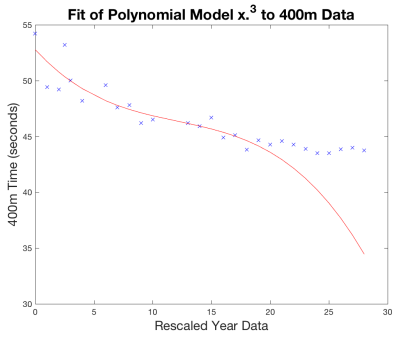
(a) Polynomial model with order $\mathbf{n} = 0$



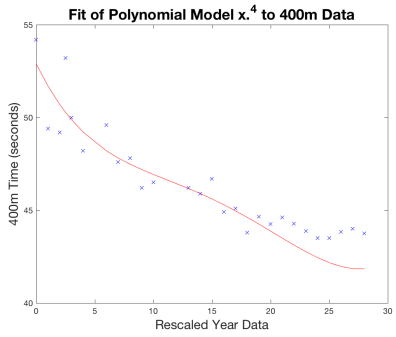
(b) Polynomial model with order $\mathbf{n} = 1$



(c) Polynomial model with order $\mathbf{n} = 2$



(d) Polynomial model with order $\mathbf{n} = 3$



(e) Polynomial model with order $\mathbf{n} = 4$

Figure 2: The original Olympic men's 400m data (blue crosses) with the polynomial models (red) overlaid.

Looking at the Olympics men's 400m data in Figure 2, the trend is slightly parabolic. This indicates a polynomial function of order 2 is the best fit and Figure 2c shows how well this model fits. Interestingly, Figure 2e, shows an order of 4 provides an even closer fit and a more realistic future prediction than an order of 2. In comparison, an order of 2 shows that future predictions would increase in time at an increasing rate. This is highly unlikely given the downward trend of the data.

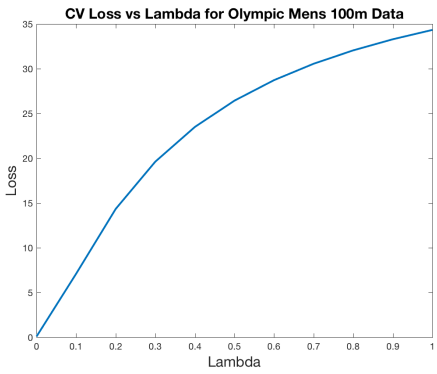
The remaining models in Figure 2 clearly do not model the data well, where orders 1 and 3 show that a time of 0 will be achieved soon. This is not humanly possibly so the models can be discarded.

Also, if we do not standardise data, we run into numerical errors when determining the polynomial function of the data with order n. Therefore, it is good practice to standardise data when the regression model contains polynomial terms [1].

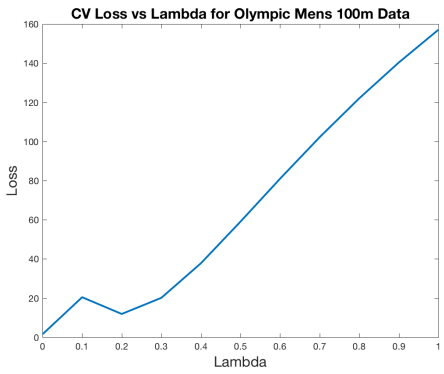
3 Task 2

In task 2 we are given the task to find the value of λ that gives the 'best' predictive performance on the Olympic men's 100m data using 5-fold cross-validation. In this task, "best" is also defined the same as in 2. Refer to Appendix B for the Matlab code used for analysis.

Figure 3b shows that the best value for λ is 0. This is expected for this dataset, as it is linear. Furthermore, a model with a higher order willl fit closer, but increases the risk of over-fitting to the data. This is unlikely to happen with an order of 4 and 27 attributes. If the order was increased to 26 (N - 1), the model would perfectly fit the observed data but this is also an extreme case of over-fitting. To counter this, the lambda value is increased to penalise the over-fitting and generalise the model. A balanced trade-off is often sought between these two variables when dealing with higher order polynomials.



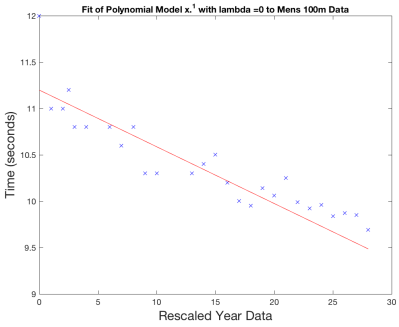
(a) Loss of varying lambda values for order 1



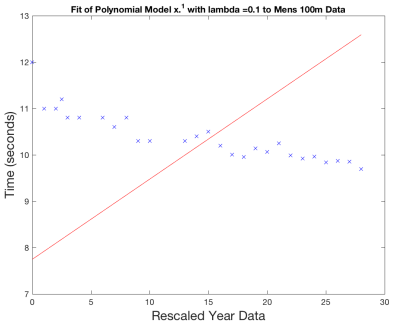
(b) Loss of varying lambda values for order 4

Figure 3: The loss vs lambda values for polynomial models with order 1, and 4.

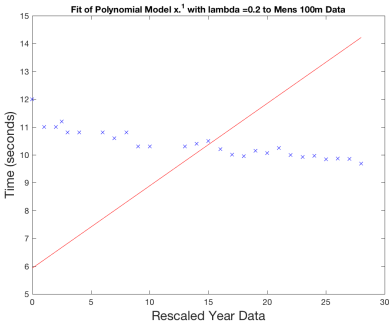
For illustrative purposes, the first 4 values of lambda for both polynomial functions are included in Figures 4 and 5. These show the increase in model loss as the value of lambda increases.



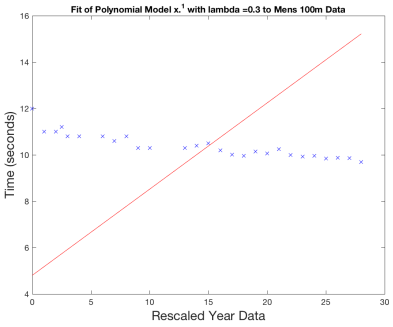
(a) Lambda = 0



(b) Lambda = 0.1

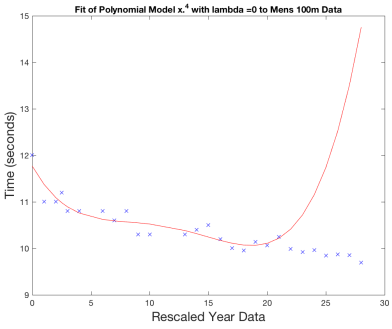


(c) Lambda = 0.2

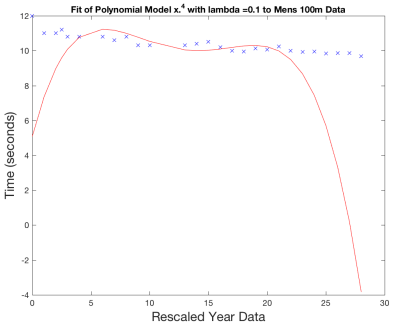


(d) Lambda = 0.3

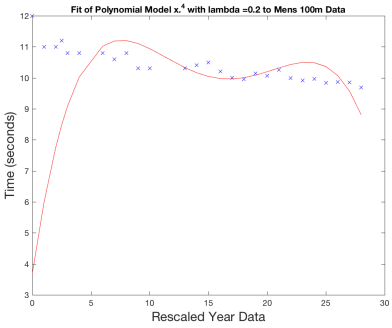
Figure 4: Polynomial model with order 1 and varying lambda values.



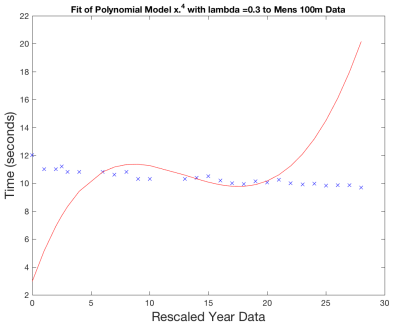
(a) Lambda = 0



(b) Lambda = 0.1



(c) Lambda = 0.2



(d) Lambda = 0.3

Figure 5: Polynomial model with order 4 and varying lambda values.

4 References

[1] Jim Frost. *When Is It Crucial to Standardize the Variables in a Regression Model? | Minitab*. http://blog.minitab.com/blog/adventures-in-statistics/when-is-it-crucial-to-standardize-the-variables-in-a-regression-model. (Accessed on 10/21/2016). Feb. 2016.

[2] Simon Rogers and Mark Girolami. *A First Course in Machine Learning*. 1st. Chapman & Hall/CRC, 2011. isbn: 1439824142, 9781439824146.

Appendix A: Task 1

```
%% CTB1_400metre
% Fit of 1st, 2nd, 3rd, and 4th order polynomial to
% the 400 metre sprint data
% We use 10-fold cross-validation to choose the 'best'
% polynomial order which has minimum loss
clear all;close all;
load olympics.mat
%% Preamble and rescaling data
x = male400(:,1);
x = [x - x(1)];
x = [x]./x(2);
t= male400(:,2); %target labels
%% Folding details
totalFolds = 10; % total number of folds
Npartition = length(x(:,1)); % we find the length of the first column in matrix x
partition = repmat(floor(Npartition/totalFolds),1,totalFolds); % we determine the
    size of the partition (2.7 in this case) and take the floor
partition(end) = Npartition - (totalFolds-1)*partition(1,1);
pcum = [0 cumsum(partition)];
%% Polynomial Order of n
% Change value of 'order' to choose polynomial order n
order = 4; % order of polynomial
X = [];
orderFoldMat = [];
orderTrainMat = [];

 for k = 0:order % we loop to compute CV loss and Train loss for each order of the
    polynomial
    X = [X x.^k]; % we add a column to matrix for every order n where the new
    column is x^k
    for fold = 1:totalFolds % we loop again to change the partition being tested
    and trained
        Xfold = X(pcum(fold)+1 : pcum(fold+1),:); % we slice X to get the rows for
    the CV (test) partition (1)
        Xtrain1 = X(1 : pcum(fold),:); % we slice X to get the training partition
    before the CV partition (2)
        Xtrain2 = X(pcum(fold+1)+1 : end,:); % we slice X to get the training
    partition afer the CV partition (3)
        Xtrain = [Xtrain1; Xtrain2]; % we join the training paritions into one
    matrix (4)

        tfold = t(pcum(fold)+1 : pcum(fold+1),:); % we partition the labels the
    same as (1) to validate the results
        ttrain1 = t(1 : pcum(fold),:); % we partition the labels the same as (2) to
     train the model
        ttrain2 = t(pcum(fold+1)+1 : end,:); % we partition the labels the same as
    (3) train the model
        ttrain = [ttrain1; ttrain2]; % we join the label training data into one
    matrix similar to (4)
```

```
        % we learn the model parameters from the training data and labels(5)
        w = inv(Xtrain'*Xtrain)*Xtrain'*ttrain;
        % we find the predicted results using the params in (5) and (1)
        mpred_fold = Xfold*w;
        % we see how well the model reflects the observed (training) data
        mpred_observed = Xtrain*w;
        % we compare the CV labels to the predicted results
        compare = [tfold mpred_fold tfold-mpred_fold];

        % we store the mean squared difference of CV labels and
        % predicted results, to plot later
        foldLoss(fold,k+1)  = mean((mpred_fold - tfold).^2);
        trainLoss(fold,k+1) = mean((mpred_observed - ttrain).^2);
    end
    orderFoldMat = [orderFoldMat mean(foldLoss(:,k+1))];
    orderTrainMat = [orderTrainMat mean(trainLoss(:,k+1))];

% %     plots and saves to file
% %     comment out to turn off
%     plot(x,t,'bx');
%     plotTitle = strcat('Fit of Polynomial Model x.^',int2str(k),' to 400m Data');
%     title(plotTitle,'fontsize',18);
%     xlabel('Rescaled Year Data','fontsize',16);
%     ylabel('400m Time (seconds)','fontsize',16);
%     hold on;
%     plot(x,w'*X','r');
%     filename = strcat('model',int2str(k),'.png');
%     saveas(gcf,filename);
%     clf;
 end
%% Plot the results
% Reference: A First Course in Machine Learning, Chapter One, cv_demo.m
figure(1);
subplot(121)
plot(0:order,orderFoldMat,'linewidth',2);
xlabel('Model Order','fontsize',16);
ylabel('Loss','fontsize',16);
title('CV Loss','fontsize',18);
subplot(122)
plot(0:order,orderTrainMat,'linewidth',2)
xlabel('Model Order','fontsize',16);
ylabel('Loss','fontsize',16);
title('Train Loss','fontsize',18);
filename = strcat('CVLossANDTrainLoss',int2str(order),'.png');
saveas(gcf,filename);
```

Appendix B: Task 2

```
%% CTB1_100metre
% Fit of 1st and 4th order polynomial to
% the 100 metre sprint data
% We vary the regularisation factor, lambda,
% to determine the best value for the
% polynomial models.
clear all;close all;
load olympics.mat
%% Preamble and rescaling data
x = male100(:,1);
x = [x - x(1)];
x = [x]./x(2);
t= male100(:,2); %target labels
%% Folding details
totalFolds = 5; % total number of folds
Npartition = length(x(:,1)); % we find the length of the first column in matrix x
partition = repmat(floor(Npartition/totalFolds),1,totalFolds); % we determine the
    size of the partition (2.7 in this case) and take the floor
partition(end) = Npartition - (totalFolds-1)*partition(1,1);
pcum = [0 cumsum(partition)];
%% Polynomial Order of n
% Change value of 'order' to choose polynomial order n
order = 1; % order of polynomial
X = [];
orderFoldMat = [];
orderTrainMat = [];
matCVLoss = [];
matLambda = [];

for k = 0:order % we loop to build the attribute set for order n
    X = [X x.^k]; % we add a column to matrix for every order n where the new
    column is x^k
end

for lambda = 0:0.1:1 % we loop through lambda values to vary the regularisation
    for fold = 1:totalFolds % we loop again to change the partition being tested
    and trained
        Xfold = X(pcum(fold)+1 : pcum(fold+1),:); % we slice X to get the rows for
    the CV (test) partition (1)
        Xtrain1 = X(1 : pcum(fold),:); % we slice X to get the training partition
    before the CV partition (2)
        Xtrain2 = X(pcum(fold+1)+1 : end,:); % we slice X to get the training
    partition afer the CV partition (3)
        Xtrain = [Xtrain1; Xtrain2]; % we join the training paritions into one
    matrix (4)

        tfold = t(pcum(fold)+1 : pcum(fold+1),:); % we partition the labels the
    same as (1) to validate the results
        ttrain1 = t(1 : pcum(fold),:); % we partition the labels the same as (2) to
```

```
     train the model
        ttrain2 = t(pcum(fold+1)+1 : end,:); % we partition the labels the same as
    (3) train the model
        ttrain = [ttrain1; ttrain2]; % we join the label training data into one
    matrix similar to (4)


        E = eye(size(Xtrain,2));
        % we learn the model parameters from the training data and labels(5)
        w = inv(Xtrain'*Xtrain + Npartition*lambda*E)*Xtrain'*ttrain;
        % we find the predicted results using the params in (5) and (1)
        mpred_fold = Xfold*w;
        % we see how well the model reflects the observed (training) data
        mpred_observed = Xtrain*w;
        % we compare the CV labels to the predicted results
        compare = [tfold mpred_fold tfold-mpred_fold];

        % we store the mean squared difference of CV labels and
        % predicted results, to plot later
        foldLoss(fold,1)  = mean((mpred_fold - tfold).^2);
    end

    matCVLoss = [matCVLoss mean(foldLoss)];
    matLambda = [matLambda lambda];

    % plots and saves to file
    plot(x,t,'bx');
    plotTitle = strcat('Fit of Polynomial Model x.^',int2str(k),' with lambda = ',
    num2str(lambda),' to Mens 100m Data');
    title(plotTitle);
    xlabel('Rescaled Year Data','fontsize',16);
    ylabel('Time (seconds)','fontsize',16);
    hold on;
    plot(x,w'*X','r');
    filename = strcat('LambdaModel',num2str(lambda),'_',int2str(order),'.png');
    saveas(gcf,filename);
    clf;
end
%% Plot the results
% Reference: A First Course in Machine Learning, Chapter One, cv_demo.m
figure(1);
plot(matLambda,matCVLoss,'linewidth',2);
xlabel('Lambda','fontsize',16);
ylabel('Loss','fontsize',16);
title('CV Loss vs Lambda for Olympic Mens 100m Data','fontsize',16);
filename = strcat('CVLossvsLambda',int2str(order),'.png');
saveas(gcf,filename);
```