Machine Learning (Extended)

Computer Based Test 1
Linear Regression by Least Squares

October 21, 2016

Thomas Brereton 1708846

Table of Content

List of Figures

List of Tables

1 Abstract

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 2 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

2.1 Introduction

In Task 1, we are asked to analyse the Olympics men's 400m data. We must find the polynomial function of order **n** which best fits 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.

To compute the average cross validation loss, data (attributes and labels) are separated into 10 partitions (i.e. 10-fold), where 1 partition is reserved for testing the model. The 9 remaining partitions are used to learn the model with parameters, w_n. This models can use the attributes of the test partition to predict the labels, in this case the time ran for the men's 400m. The predicted values can then be compared to the actual values (labels) from the test partition. The cross validation (CV) loss is calculated by taking the mean squared difference (*msd*) of these two values. This is then repeated 9 more times by rotating the test partition through. The average of these 10 *msd* values is calculated and then plotted against the order to find the minimum value.

2.2 Results

From Figure 1 and Table 1 it is shown that a polynomial function with order 4 is the best fit. However, in order to determine the "best" order, it must be given a concrete meaning for this task. "Best" is considered to be 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. So, three cases are defined as the following.

1. Only CV loss is considered
2. Only Train loss is considered
3. Both CV and Train loss are considered

Consider item 1, Figure 1 and Table 1 show that a polynomial function with order 2 is the best fit. It is visualised clearly on Figure 1 as the minimum point on the line. Looking at Table 1, a minimum value of 1.57 also corresponds with order 2.

Consider item 2, Figure 1 and Table 1 show that a polynomial function with order 4 is the best fit. Figure 1 shows a downward trend to the right, indicating that an order of 4 is indeed the best fit. Looking at Table 1, the minimum value, 3.30, also corresponds with an order of 4.

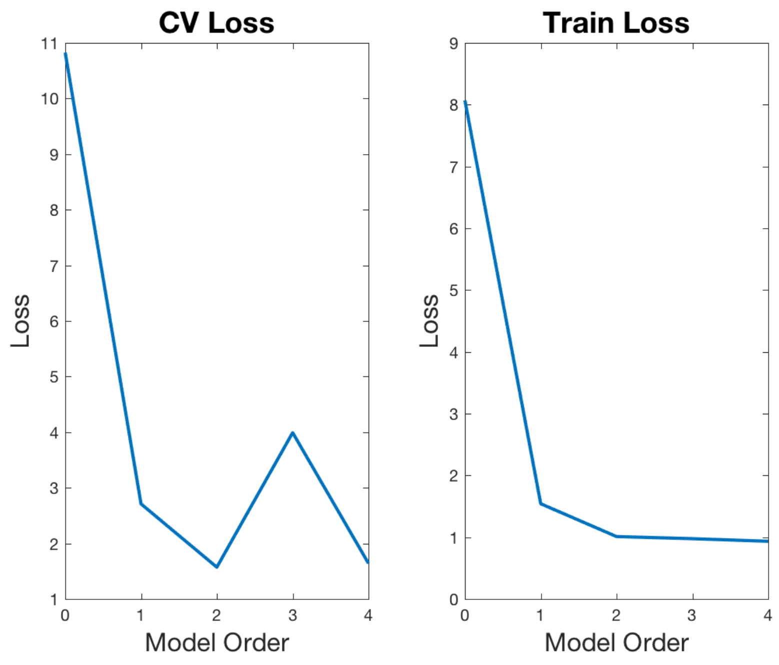


Figure 1: Comparison of the CV and Train loss for polynomial models of order **n**, where **n** = 0 to 4.

Table 1: Summary of Model Losses

Order	CV Loss	Train Loss	Average Squared Loss
0	10.83	8.07	178.61
1	2.71	1.54	9.03
2	1.57	1.01	3.33
3	3.99	0.98	12.35
4	1.64	0.93	3.30

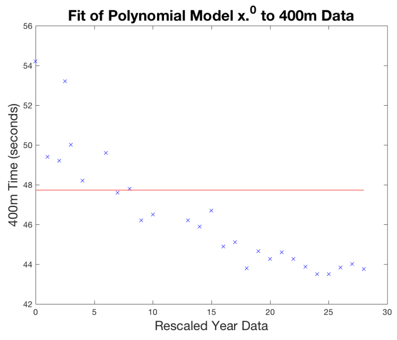
Consider item 3, Figure 1 and Table 1 show that a polynomial function with order 4 is the best fit. It is difficult to see this via the visualisation of Figure 1. However, looking at Table 1, the average squared loss (of CV and Train loss) is lowest when order equals 4.

Figure 2c shows how well each of the models fit the data. Interestingly, Figure 2e, shows an order of 4 provides an accurate model and more realistic future predictions 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.

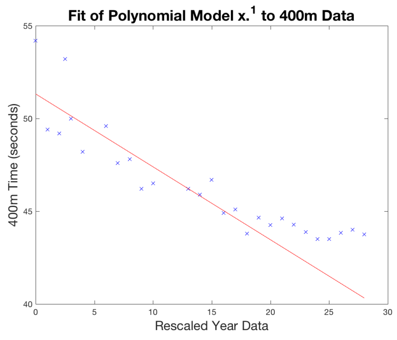
The remaining models in Figure 2 clearly do not model the data well, where orders 1 and 3 show that a time of 0 will be achieved soon. This is not humanly possibly so the models can be discarded.

2.3 Conclusion

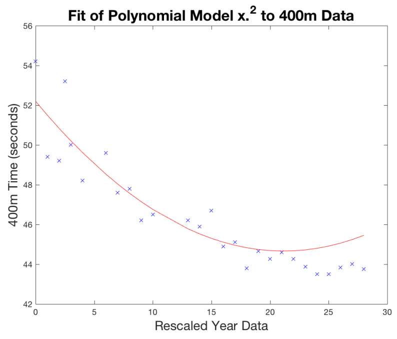
The problem for Task 1 is to find the "best model based on average cross-validation loss." This only considers point 1 from before, therefore, a polynomial function with order 2 best fits the model based on average cross-validation loss.



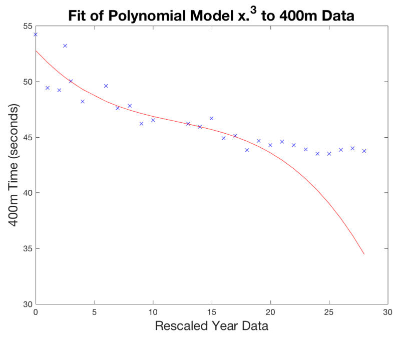
(a) Polynomial model with order **n** $= 0$



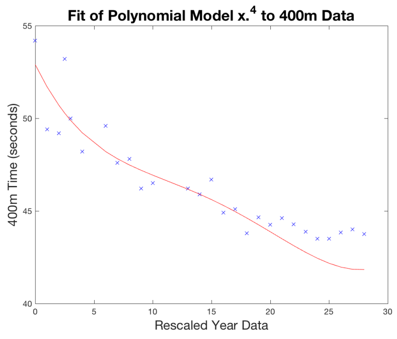
(b) Polynomial model with order **n** $= 1$



(c) Polynomial model with order **n** $= 2$



(d) Polynomial model with order **n** $= 3$



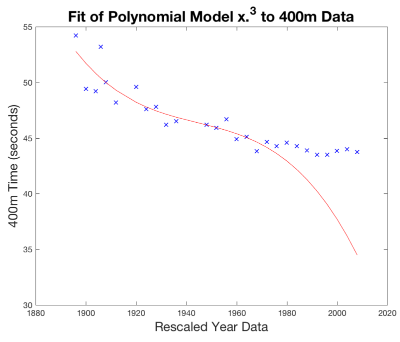
(e) Polynomial model with order **n** $= 4$

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

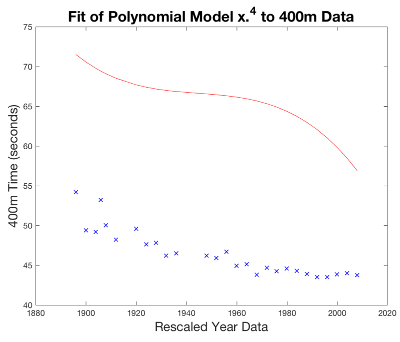
2.4 Further Comments

Further investigation found if the data was not standardised, a systematic error is produced in the model for orders greater than or equal 4. Figure 3 illustrates shows that for and order of 3 there is not error, but there is for an order of 4. Reviewing literature showed this systematic error is common when dealing with high order polynomial function[**WhenIsIt**].Therefore, it is good practice to standardise data when the regression model contains polynomial terms.

Another point worth mentioning is the effect of permuting (randomising) the attributes. For this small dataset the permutations caused some instability in the model and different results were obtained as can be seen in Figure 4. The results of the permuted data agree with the results from before.

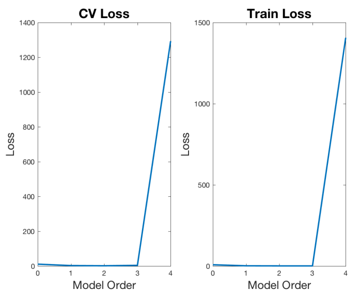


(a) Polynomial model with order $\mathbf{n} = 3$

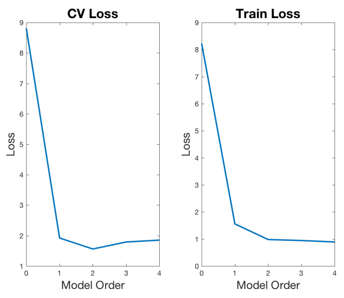


(b) Polynomial model with order $\mathbf{n} = 4$

Figure 3: The original Olympic men's 400m data (blue crosses) with the polynomial models (red) overlaid without data standardisation.



(a) CV and Train loss when the data is not standardised



(b) CV and train loss with permuted data

Figure 4: Comparision of the CV and Train Loss when it is not standardised (4a) and data is permuted (4b).

3 Task 2

3.1 Introduction

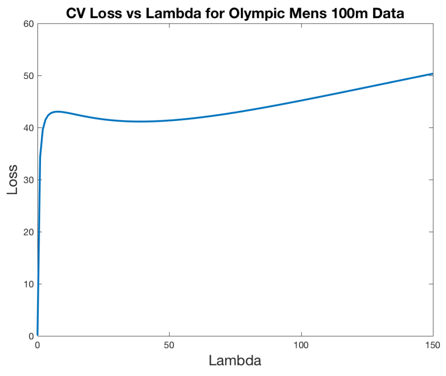
In task 2 we are given the task to find the value of λ that gives the 'best' predictive performance on the Olympic men's 100m data using 5-fold cross-validation. In this task, "best" is also defined the same as in 2. Refer to Appendix B for the Matlab code used for analysis.

Task 2 uses the same method as Task 1, however, an additional parameter, λ, is introduced. The calculation for the CV loss essentially remains the same, except now higher order models (complexity) are penalised by the λ value.

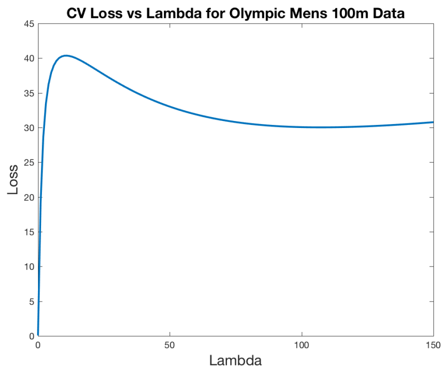
3.2 Results

Figure 5 shows that the best value for λ is 0 for this dataset. However, there is an interesting trend for both orders, where it increases until some value and then there is a local minimum. If this is a common trend, and 0 is not always the minimum, this local minimum could be a desired value to minimise loss.

Although a large range of λ is plotted on 5 , is should be noted that the value is very sensitive. This is evidenced by Figures 6 and 7, where the λ value only varies from 0 to 0.3. The figures indicate there is a model loss even for a λ value of 0.1.

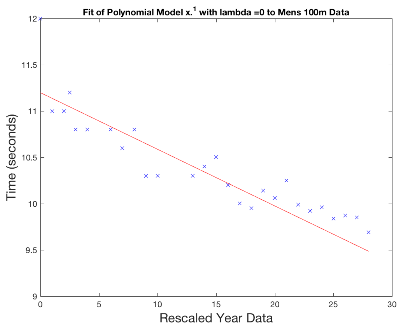


(a) Loss of varying lambda values for order 1

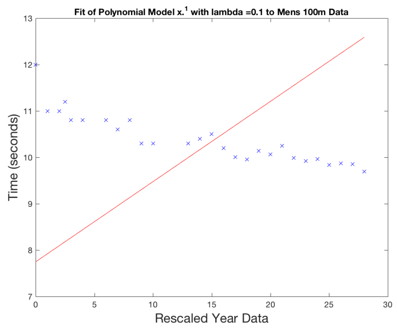


(b) Loss of varying lambda values for order 4

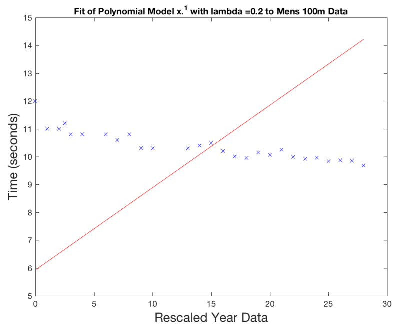
Figure 5: The loss vs lambda values for polynomial models with order 1, and 4.



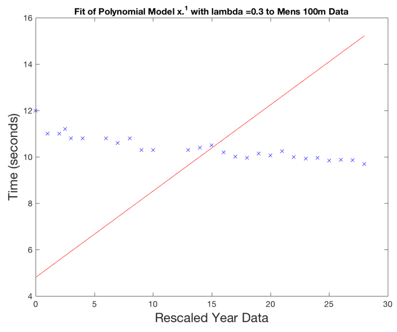
(a) Lambda = 0



(b) Lambda = 0.1



(c) Lambda = 0.2

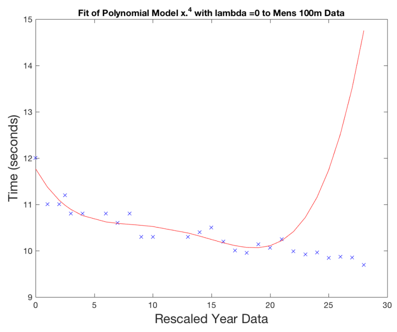


(d) Lambda = 0.3

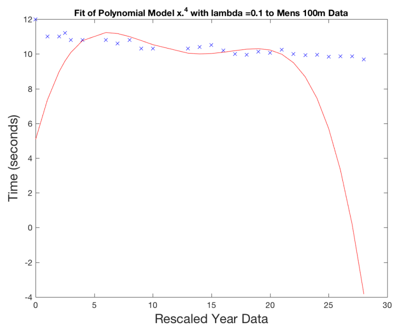
Figure 6: Polynomial model with order 1 and varying lambda values.

3.3 Conclusion

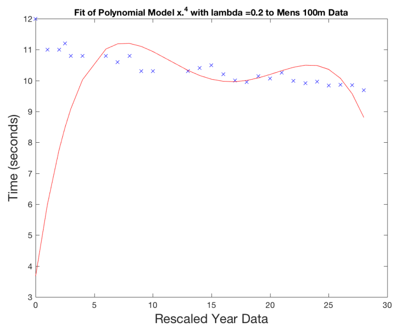
The problem for Task 2 is "using 5-fold cross-validation, find the value of λ that gives the "best" predictive performance on the Olympic men's 100m for 1st and 4th order polynomials." Predictive performance only considers CV loss, as described in point 1 in Section 2.2. Therefore, referring to Figure 5, it is clear λ with value 0, gives the best predictive performance.



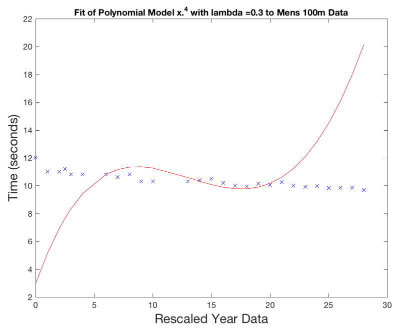
(a) Lambda = 0



(b) Lambda = 0.1



(c) Lambda = 0.2



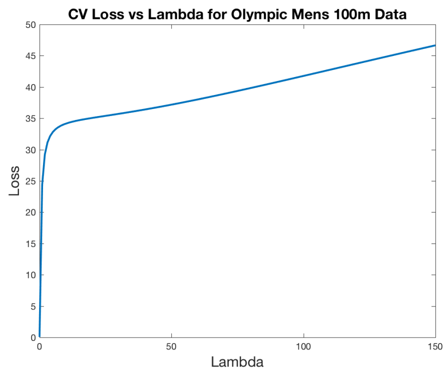
(d) Lambda = 0.3

Figure 7: Polynomial model with order 4 and varying lambda values.

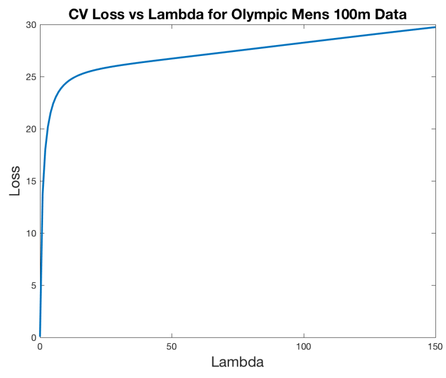
3.4 Further Comments

Further investigation showed that permuting the data also introduced instability for this small dataset as shown in Figure 8. Interestingly, this removes the local minimum found in 5, which may disprove the hypothesis that a λ value for a local minima is desired.

The results of the permuted data supports the hypothesis that a λ value of 0 gives the best predictive performance for the Olympics men's 100m data.



(a) Loss of varying lambda values for order 1



(b) Loss of varying lambda values for order 4

Figure 8: The loss vs lambda values for polynomial models with order 1, and 4 with permuted data.

4 References

Appendix A: Task 1

```matlab
%% 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);
```