Stats 102B - Week 2, Lecture 1

Miles Chen, PhD

Department of Statistics

Week 2 Monday

UCLA

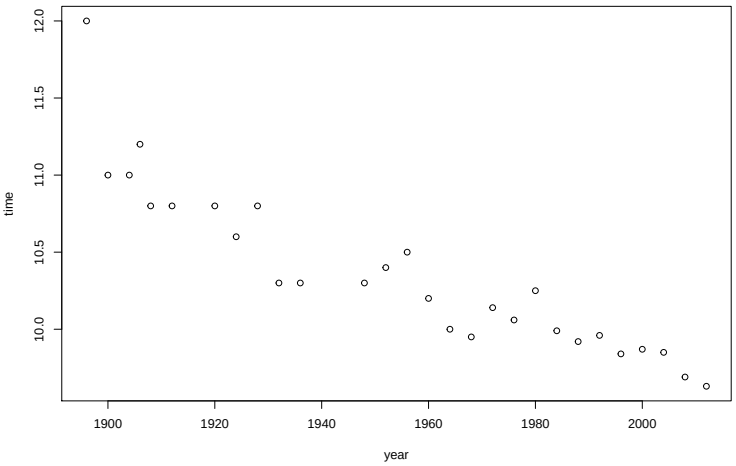
Copyright Miles Chen. For personal use only. Do not distribute.

Section 1

Cross-Validation

Copyright Miles Chen. For personal use only. Do not distribute.

Load the Olympic Data



Copyright Miles Chen. For personal use only. Do not distribute.

Fitting Models of different degrees

```
linear_fit <- lm(olympic$time ~ olympic$year)
poly2model <- lm(olympic$time ~ poly(x, 2))
poly3model <- lm(olympic$time ~ poly(x, 3))
poly4model <- lm(olympic$time ~ poly(x, 4))
poly5model <- lm(olympic$time ~ poly(x, 5))
poly6model <- lm(olympic$time ~ poly(x, 6))
```

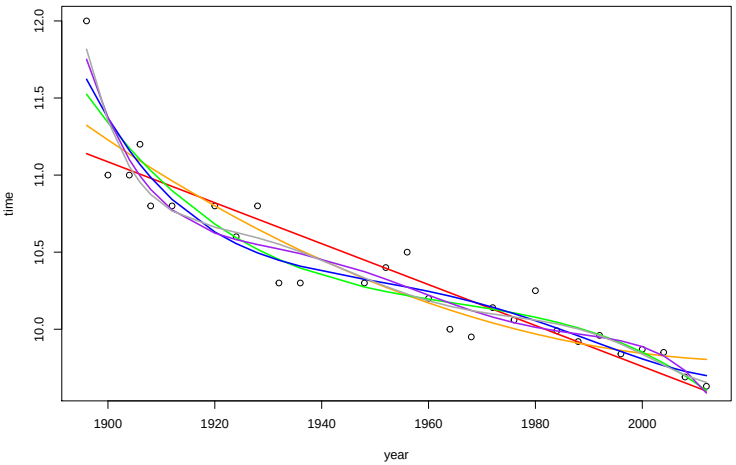
Copyright Miles Chen. For personal use only. Do not distribute.

Plot the fitted models

```
plot(olympic$year, olympic$time, ylab = 'time', xlab = 'year')
lines(olympic$year, linear_fit$fitted.values, col = 'red', lwd = 2)
lines(olympic$year, poly2model$fitted.values, col = 'orange', lwd = 2)
lines(olympic$year, poly3model$fitted.values, col = 'green', lwd = 2)
lines(olympic$year, poly4model$fitted.values, col = 'blue', lwd = 2)
lines(olympic$year, poly5model$fitted.values, col = 'purple', lwd = 2)
lines(olympic$year, poly6model$fitted.values, col = 'darkgray', lwd = 2)
```

Copyright Miles Chen. For personal use only. Do not distribute.

Plot the fitted models - Which model is best?



Copyright Miles Chen. For personal use only. Do not distribute.

Model Selection

The classical approach requires thinking and knowledge about the variables.

Does it make sense for race times to have a quadratic or cubic relationship to time? What about a 6th order relationship to time? These questions are hard to answer, and hard to reason about. The classical approach does have a few tools like AIC or BIC to aid in the model selection procedure.

In contrast, Cross-validation removes this skillful (or arbitrary) thinking from the decision-making process. It replaces it with a validation strategy.

Copyright Miles Chen. For personal use only. Do not distribute.

Cross-validation

Cross-validation splits the observed data into two parts: a training set, and a validation set.

We use the **training set** to fit the model. (For linear regression, fitting the model just means finding $\hat{\mathbf{w}}$)

After fitting the model by using the training set, we use the estimated parameters to make predictions on the **validation set**.

We use a **metric** to measure how well the model performs at making predictions. (MSE is a common choice.)

We do this for each kind of model and by comparing the metrics, we can choose the best performing model.

Copyright Miles Chen. For personal use only. Do not distribute.

Leave-One-Out Cross Validation (LOOCV)

One method of cross-validation is called Leave-one-out cross validation (LOOCV).

In LOOCV, the training data is all of the available data minus one observation. The one observation that is left out serves as the validation set.

Each data point gets to have a turn being the validation set, resulting in a total of N training / validation sets.

Copyright Miles Chen. For personal use only. Do not distribute.

Training = 1,2,3; Validation = 4

For illustration, we will use a toy dataset:

```
x <- c(1, 2, 3, 4)
t <- c(1, 3, 2, 4)
```

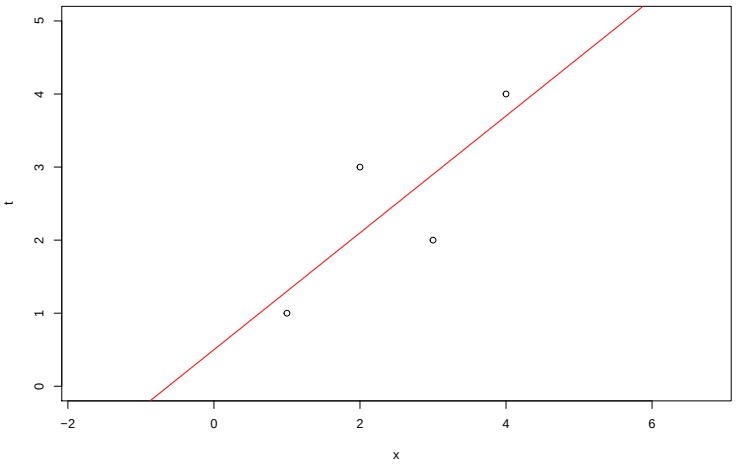
I initialize the total squared error with 0

```
index <- 4          # index of the validation point
total_sqe = 0       # total squared error starts as 0
```

```
trainx <- x[-index]
validx <- x[index]
traint <- t[-index]
validt <- t[index]
train_data <- data.frame(x = trainx, t = traint) # make a dataframe
model_loo <- lm(t ~ x, data = train_data) # fit the model to training data
```

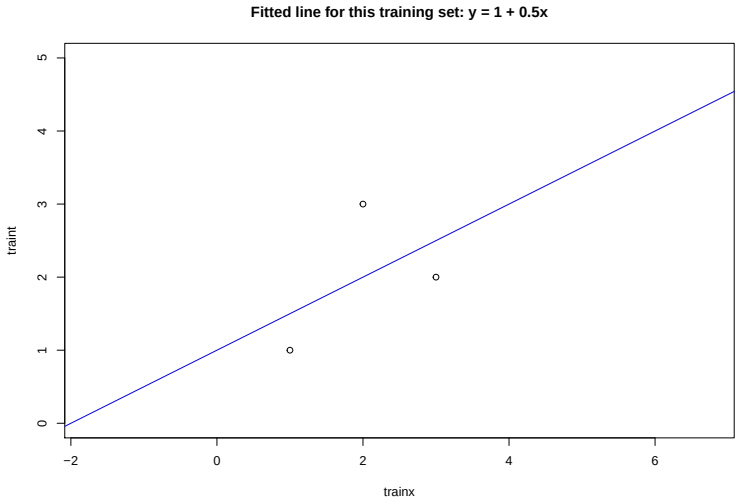
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on all data points



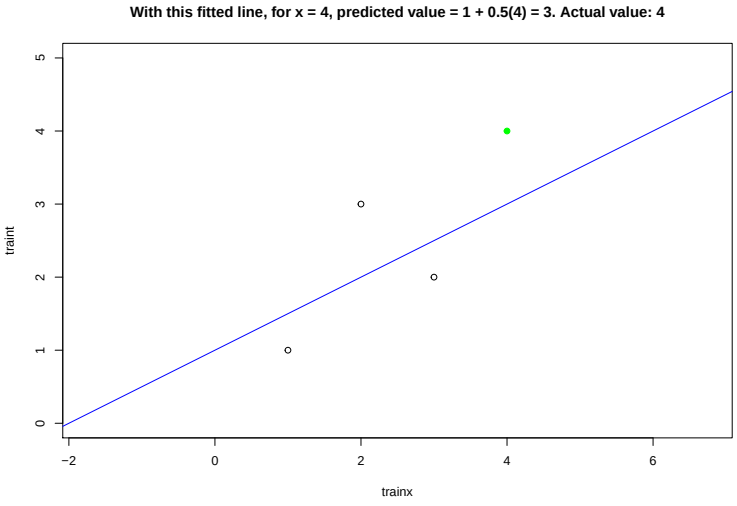
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data (points 1,2,3)



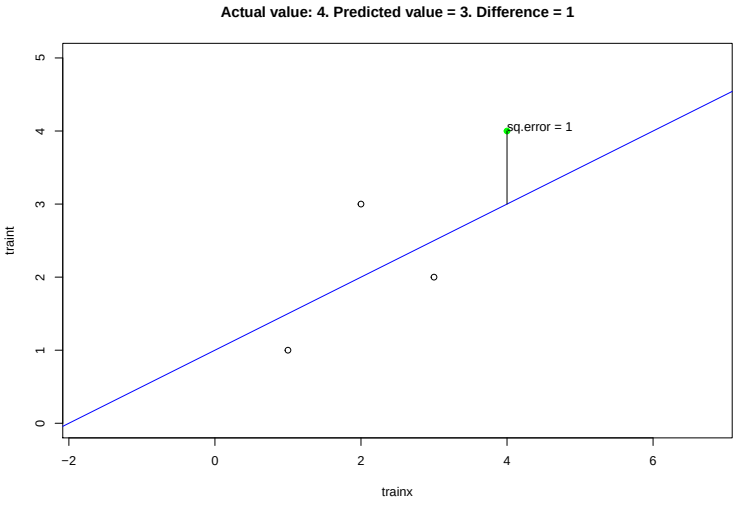
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data vs Validation point



Copyright Miles Chen. For personal use only. Do not distribute.

Validation error



Copyright Miles Chen. For personal use only. Do not distribute.

Add the squared error to the total deviation

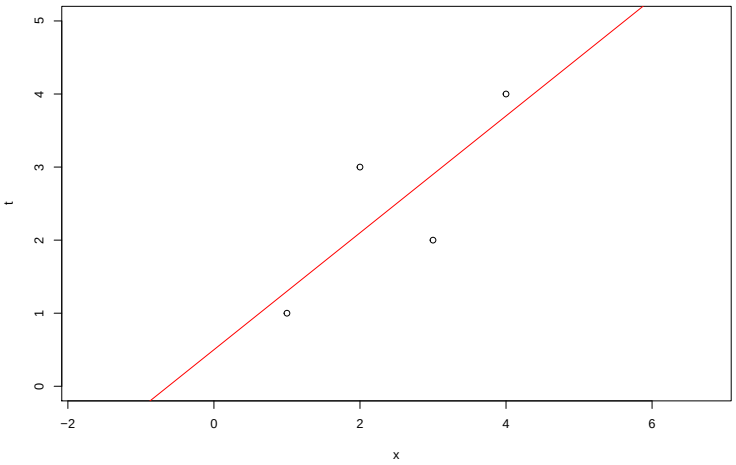
For each training-validation split, we calculate the predicted value(s) for the point(s) in the validation set.

We calculate the squared deviation, and add that to our running total squared deviation.

```
# train the Leave-One-Out model on the training data
model_loo <- lm(t ~ x, data = train_data)
# using the trained model, predict the value for the unseen validation point
t_hat <- predict(model_loo, newdata = data.frame(x = validx))
# calculate the squared error and add it to the total squared error
sqe <- (validt - t_hat) ^ 2
total_sqe <- total_sqe + sqe
```

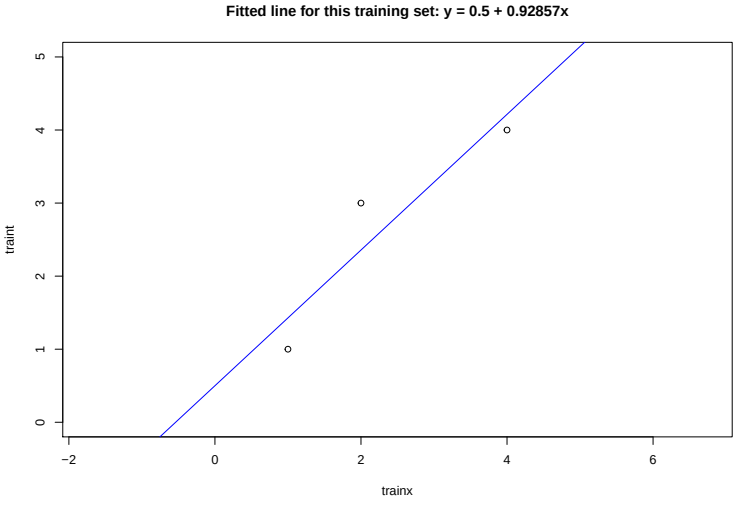
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on all data points



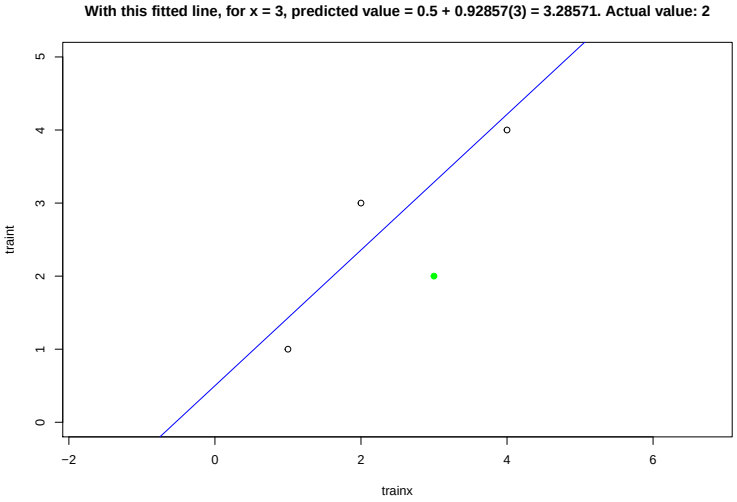
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data (points 1,2,4)



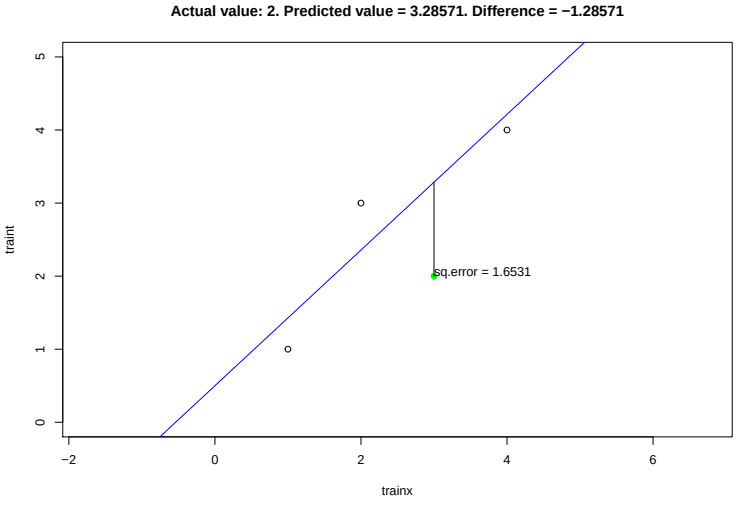
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data vs Validation point



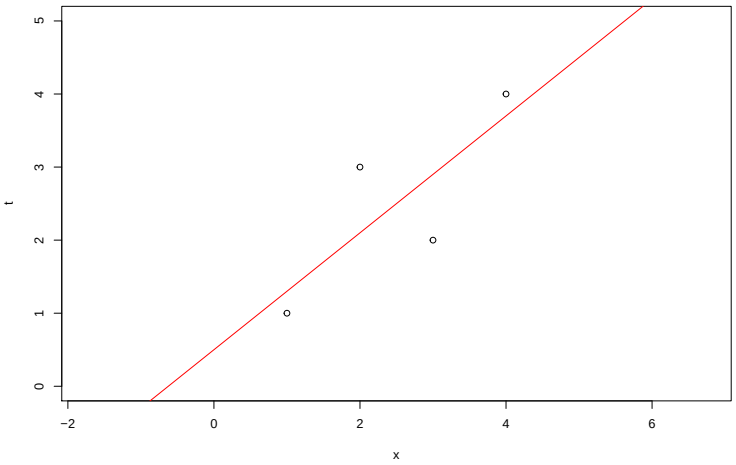
Copyright Miles Chen. For personal use only. Do not distribute.

Validation error



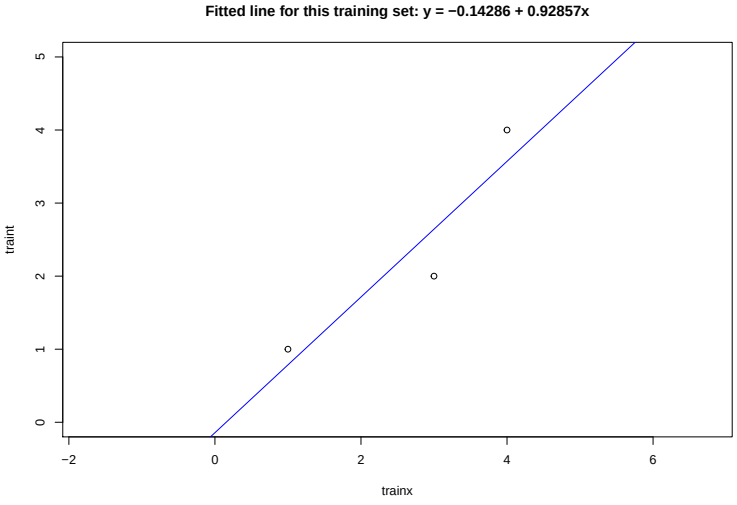
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on all data points



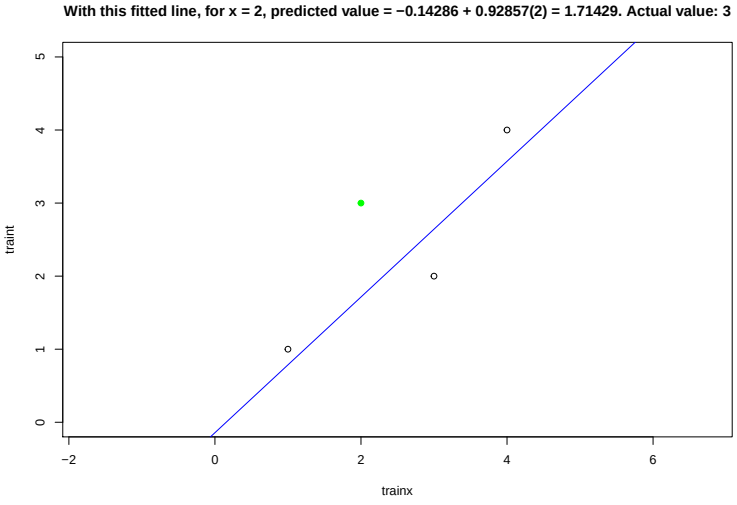
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data (points 1,3,4)



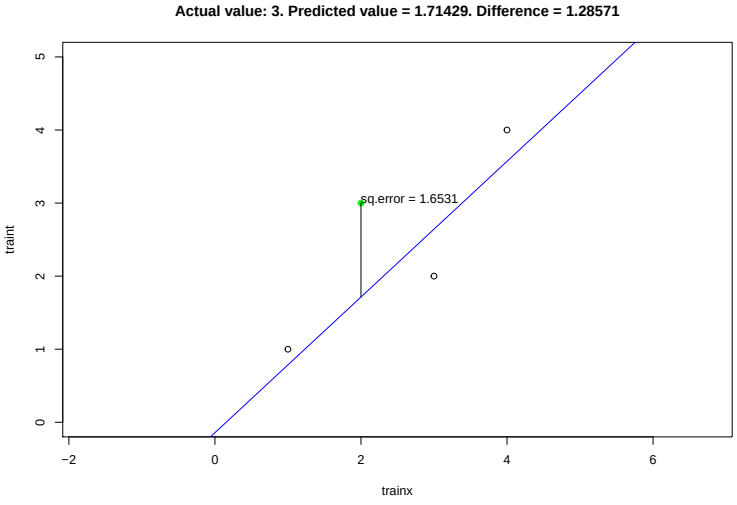
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data vs Validation point



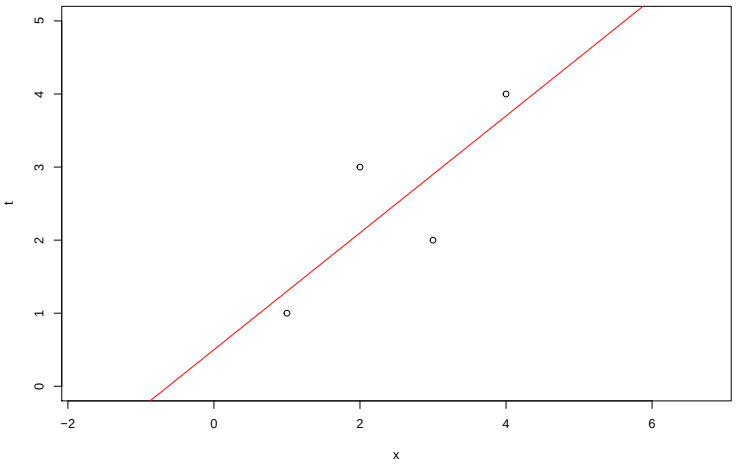
Copyright Miles Chen. For personal use only. Do not distribute.

Validation error



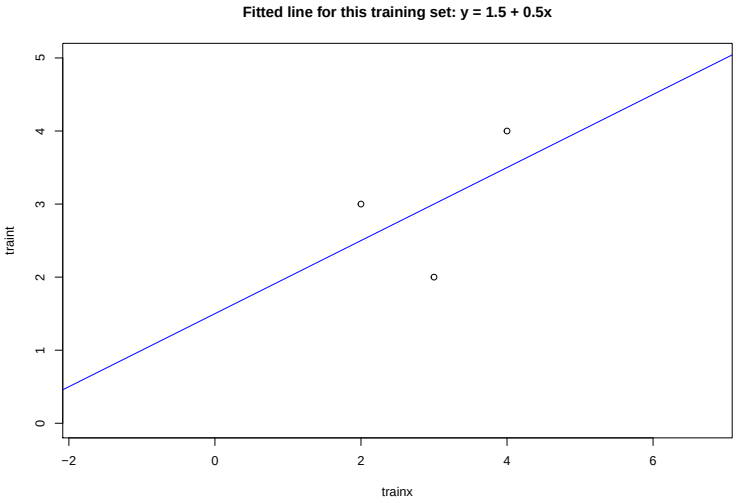
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on all data points



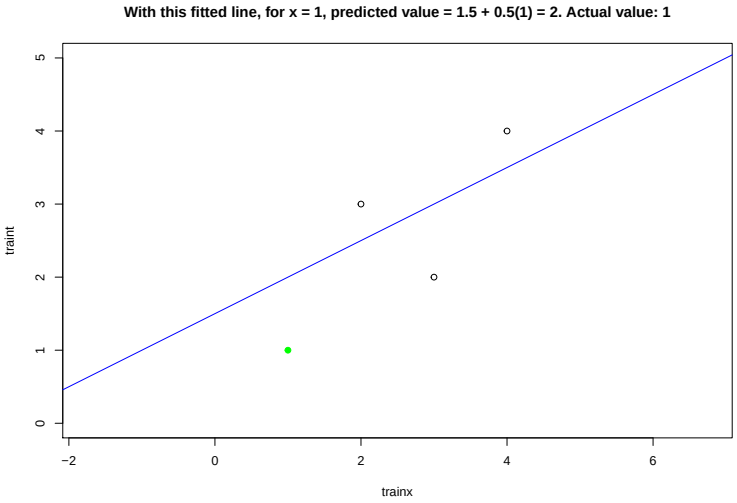
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data (points 2,3,4)



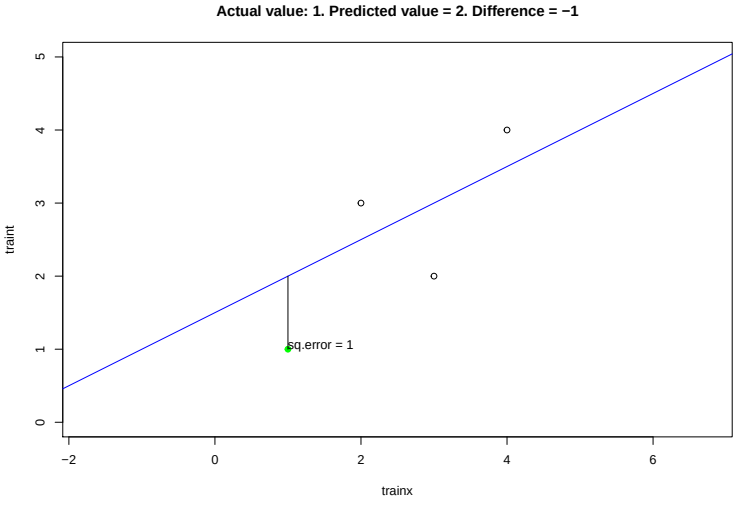
Copyright Miles Chen. For personal use only. Do not distribute.

Linear fit on the training data vs Validation point



Copyright Miles Chen. For personal use only. Do not distribute.

Validation error



Copyright Miles Chen. For personal use only. Do not distribute.

Cross-Validation error

```
total_sqe # total squared error
```

```
##        1
## 5.306122
```

```
total_sqe / 4 # MSE
```

```
##        1
## 1.326531
```

Copyright Miles Chen. For personal use only. Do not distribute.

Cross-Validation error

R has a CV function in package `boot`. Fit a `glm()` model, and use `cv.glm()`. The raw mean squared error of the cross-validation is stored in the results of `cv.glm()` in `$delta[1]`

```
library(boot)
all_data <- data.frame(x, t)
linear_model <- glm(t ~ x, data = all_data)
loocv <- cv.glm(all_data, linear_model)$delta[1]
loocv
```

```
## [1] 1.326531
```

We can see that the value matches what we calculated manually.

Copyright Miles Chen. For personal use only. Do not distribute.

Some Clarification

When using cross-validation to evaluate a model, you will get a single cross-validation score for that model (usually the mean of the mean cross-validation errors.)

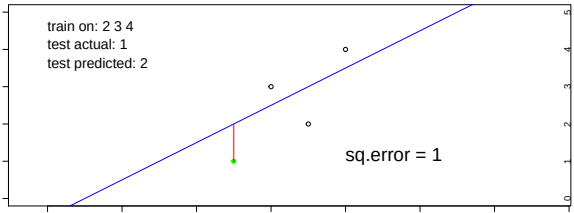
For example, on our toy data set with $N = 4$ points, we performed leave-one-out cross validation for a linear model.

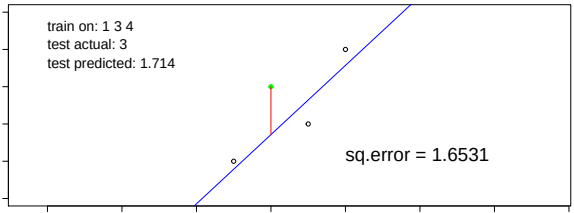
This required us to fit the linear model N times, and thus produced N cross-validation error values.

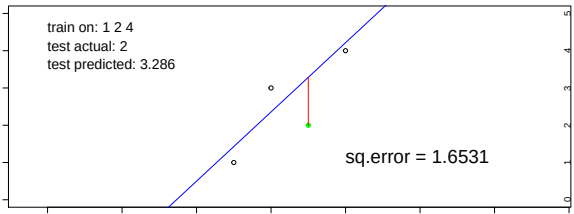
The total cross-validation score for using a linear model on the data is the mean of those four values.

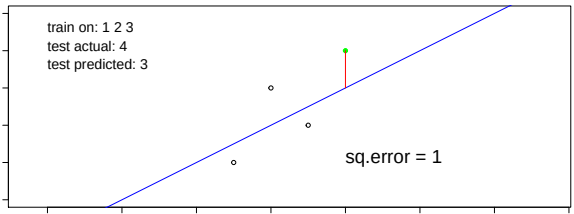
Copyright Miles Chen. For personal use only. Do not distribute.

Fit a linear model N times. Calculate validation error each time.










```
## Total Squared Error:  5.306122 ; Mean Squared Error 1.326531
```

Copyright Miles Chen. For personal use only. Do not distribute.

Clarification continued

After fitting a linear model to the data N times, we take the MSE as the CV score: 1.3265306.

We can then try a different model, maybe a log-log model or a polynomial model.

We fit the model N times, find the validation errors, and take the average. This will serve as the CV score of the alternative model.

By comparing CV scores, we can get an idea of how well the models predict values that were not in the training data.

Copyright Miles Chen. For personal use only. Do not distribute.

Section 2

K-fold Cross Validation

Copyright Miles Chen. For personal use only. Do not distribute.

K-fold Cross Validation

Leave-one-out cross validation has a couple major drawbacks:

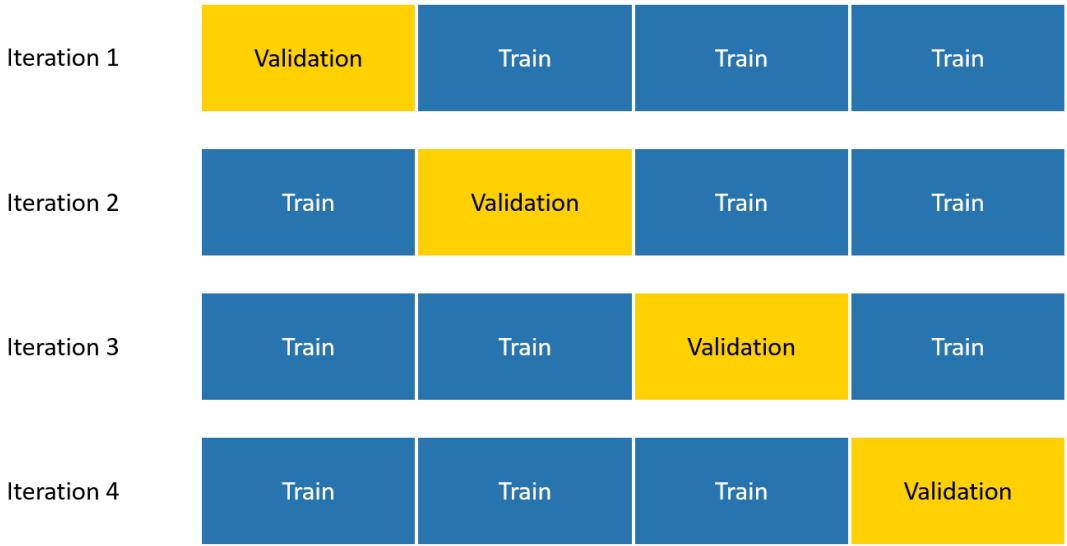
- For a dataset with N data points, you will have to fit N models. This is computationally expensive.
- Because each model is fit on all but one data point, the resulting models often end up looking very very similar to each other. The resulting coefficient estimates end up very highly correlated and are more likely to still overfit the training data.

For a large dataset, an alternative it to use k-fold cross validation. This partitions the data into K equally (or approximately equal) sized blocks.

One block serves as the validation set, while the remaining blocks combine to be the training data.

Copyright Miles Chen. For personal use only. Do not distribute.

Illustration of 4-fold cross-validation



Copyright Miles Chen. For personal use only. Do not distribute.

Revisit Olympic Data

```
dim(olympic)
```

```
## [1] 28  2
```

```
print(olympic)
```

```
##    year  time
## 1  1896 12.00
## 2  1900 11.00
## 3  1904 11.00
## 4  1906 11.20
## 5  1908 10.80
## 6  1912 10.80
## 7  1920 10.80
## 8  1924 10.60
## 9  1928 10.80
## 10 1932 10.30
## 11 1936 10.30
## 12 1948 10.30
## 13 1952 10.40
## 14 1956 10.50
```

Copyright Mike Chen. For personal use only. Do not distribute.

Plot of Olympic Data

```
plot(olympic$year, olympic$time, ylab = 'time', xlab = 'year')
```

Copyright Miles Chen. For personal use only. Do not distribute.

K-fold Cross Validation on the Olympic Data

We'll use the Olympic Data to demonstrate k-fold cross-validation as a model selection tool.

We divide the data into 4 random blocks. Each block will have a turn as serving as the validation set.

For each training/validation split (there's 4 of them), we will fit several proposed models:

- linear model
- quadratic model
- cubic model
- 4th order polynomial model
- 5th order polynomial model
- 8th order polynomial model
- 17th order polynomial model (just for fun)

We'll calculate the total squared error for each model for each validation set. We find the CV score by averaging the squared error for each model, and compare CV scores to select the model.

Copyright Miles Chen. For personal use only. Do not distribute.

4-fold CV: Setting up the train / validation sets

```
set.seed(5)
```

```
index <- sample(1:28)
setA <- index[1:7]
setB <- index[8:14]
setC <- index[15:21]
setD <- index[22:28]
sets <- list(setA, setB, setC, setD)
```

Copyright Miles Chen. For personal use only. Do not distribute.

Where to store results of total squared error

```
cv_line <- rep(NA, 4)
cv_quad <- rep(NA, 4)
cv_ord3 <- rep(NA, 4)
cv_ord4 <- rep(NA, 4)
cv_ord5 <- rep(NA, 4)
cv_ord8 <- rep(NA, 4)
cv_ord17 <- rep(NA, 4)
```

Copyright Miles Chen. For personal use only. Do not distribute.

CV Code

```r
for(i in 1:4) { # run the loop for each validation set
  v_indx <- sets[[i]]  # index of validation data
  trainx <- olympic$year[ -v_indx ]
  traint <- olympic$time[ -v_indx ]
  validx <- olympic$year[ v_indx ]
  validt <- olympic$time[ v_indx ]
  train <- data.frame(x = trainx, t = traint) # data frame for training data
  valid <- data.frame(x = validx, t = validt) # data frame for validation set
  n <- length(v_index)

  model1 <- lm(t ~ x, data = train)           # fit linear model on training
  t_hat1 <- predict(model1, newdata = valid)  # predictions for validation set
  cv_line[i] <- sum((valid$t - t_hat1)^2)/n   # save sum of squared error

  model2 <- lm(t ~ poly(x,2), data = train)   # quadratic fit
  t_hat2 <- predict(model2, newdata = valid)
  cv_quad[i] <- sum((valid$t - t_hat2)^2)/n

  # ...
```

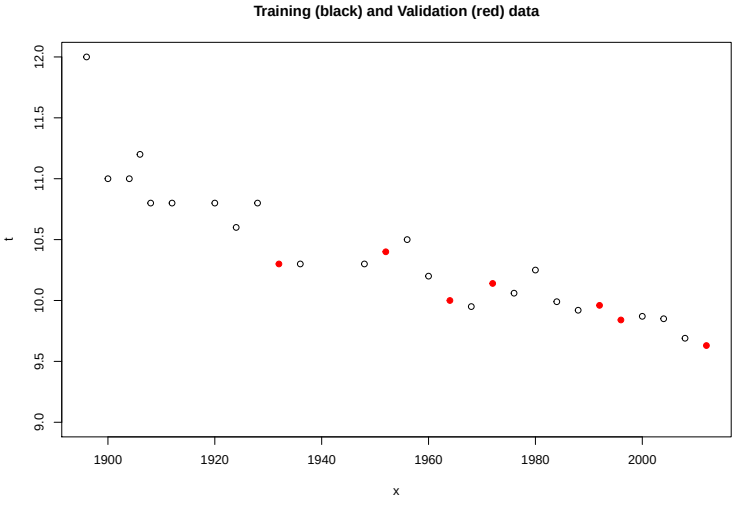
Copyright Miles Chen. For personal use only. Do not distribute.

CV code

```
for(i in 1:4){
  v_indx <- sets[[i]]  # index of validation data
  n <- length(v_indx)
  trainx <- olympic$year[ -v_indx ]; traint <- olympic$time[ -v_indx ]
  validx <- olympic$year[ v_indx ];  validt <- olympic$time[ v_indx ]
  train <- data.frame(x = trainx, t = traint); valid <- data.frame(x = validx, t = validt)

  model1 <- lm(t ~ x, data = train)  # linear fit
  t_hat1 <- predict(model1, newdata = valid)
  cv_line[i] <- sum((valid$t - t_hat1)^2)/n

  model2 <- lm(t ~ poly(x,2), data = train)  # quadratic fit
  t_hat2 <- predict(model2, newdata = valid)
  cv_quad[i] <- sum((valid$t - t_hat2)^2)/n

  model3 <- lm(t ~ poly(x,3), data = train)  # polynomial order 3
  t_hat3 <- predict(model3, newdata = valid)
  cv_ord3[i] <- sum((valid$t - t_hat3)^2)/n

  model4 <- lm(t ~ poly(x,4), data = train)  # polynomial order 4
  t_hat4 <- predict(model4, newdata = valid)
  cv_ord4[i] <- sum((valid$t - t_hat4)^2)/n

  model5 <- lm(t ~ poly(x,5), data = train)  # polynomial order 5
  t_hat5 <- predict(model5, newdata = valid)
  cv_ord5[i] <- sum((valid$t - t_hat5)^2)/n

  model8 <- lm(t ~ poly(x,8), data = train)  # polynomial order 8
  t_hat8 <- predict(model8, newdata = valid)
  cv_ord8[i] <- sum((valid$t - t_hat8)^2)/n

  model17 <- lm(t ~ poly(x,17), data = train)  # polynomial order 8
  t_hat17 <- predict(model17, newdata = valid)
```

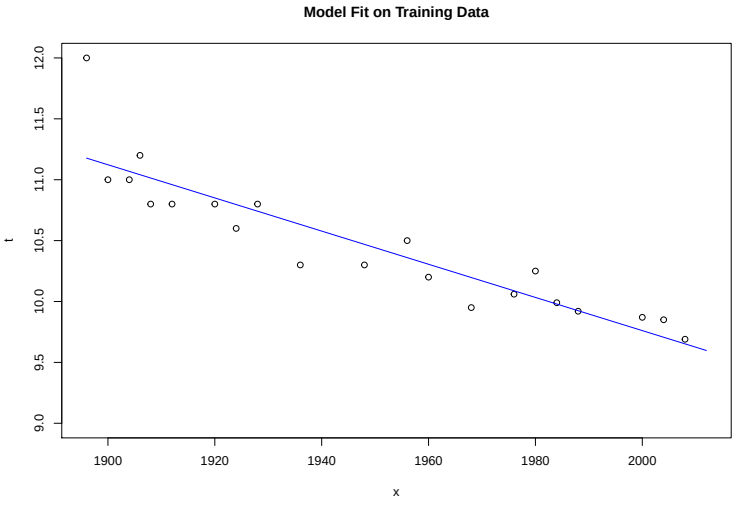
Copyright Miles Chen. For personal use only. Do not distribute.

Plot of the points in one of the four validation sets (Set C)



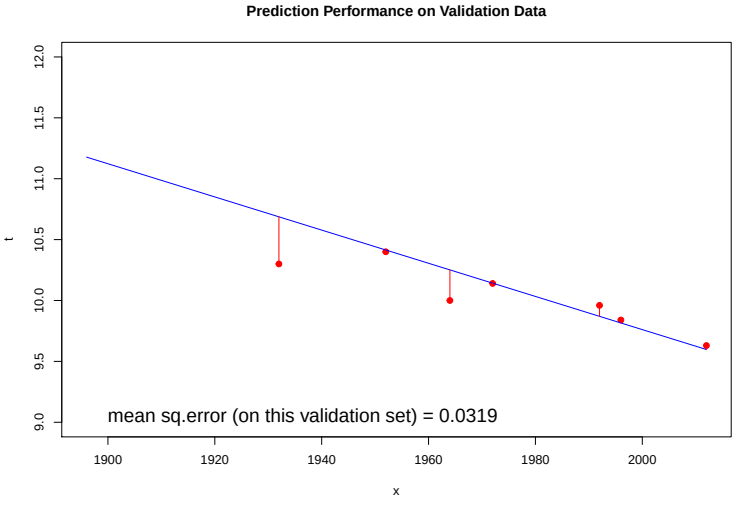
Copyright Miles Chen. For personal use only. Do not distribute.

Linear model fit to training data



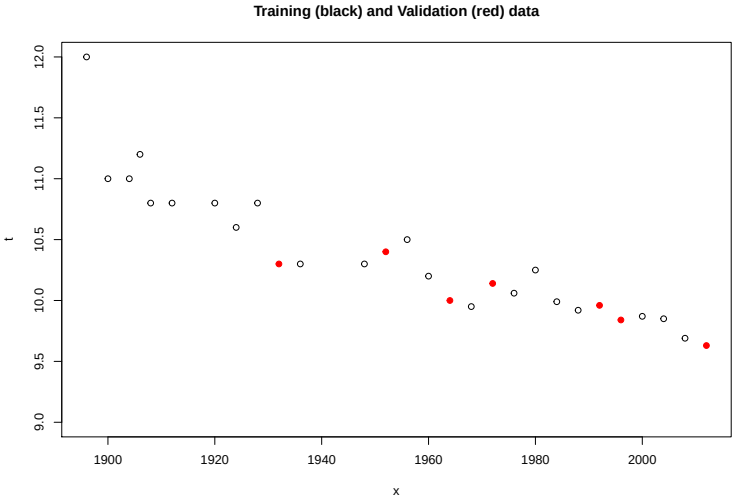
Copyright Miles Chen. For personal use only. Do not distribute.

Checking the predictive performance of the Linear model



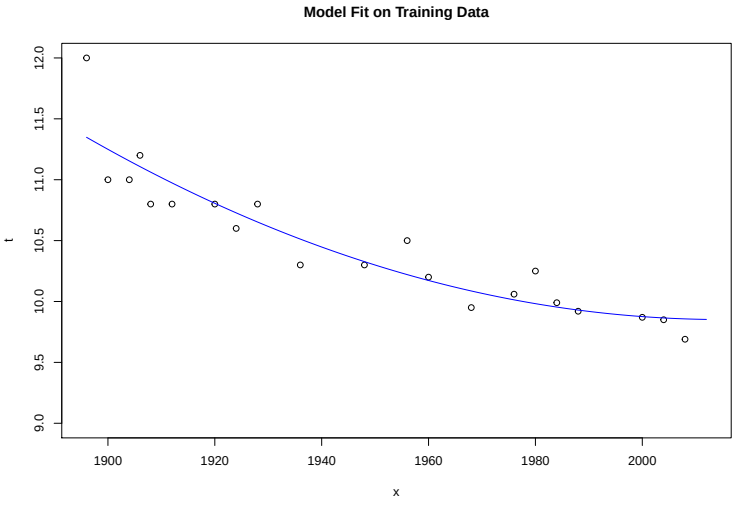
Copyright Miles Chen. For personal use only. Do not distribute.

Validation Set C for reference



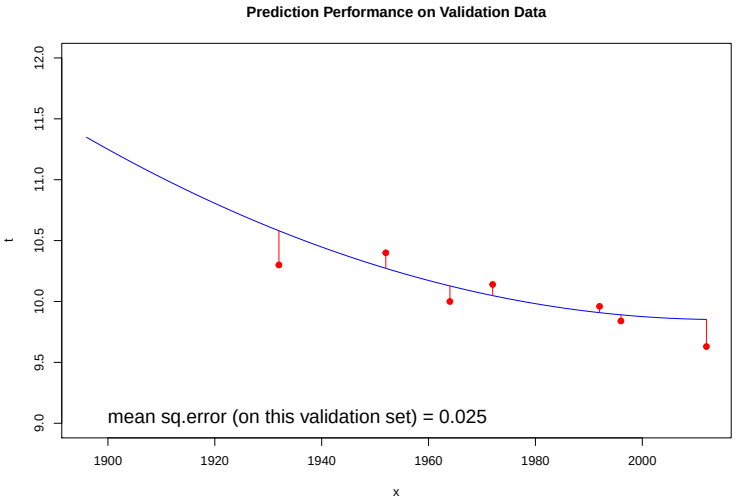
Copyright Miles Chen. For personal use only. Do not distribute.

Quadratic model



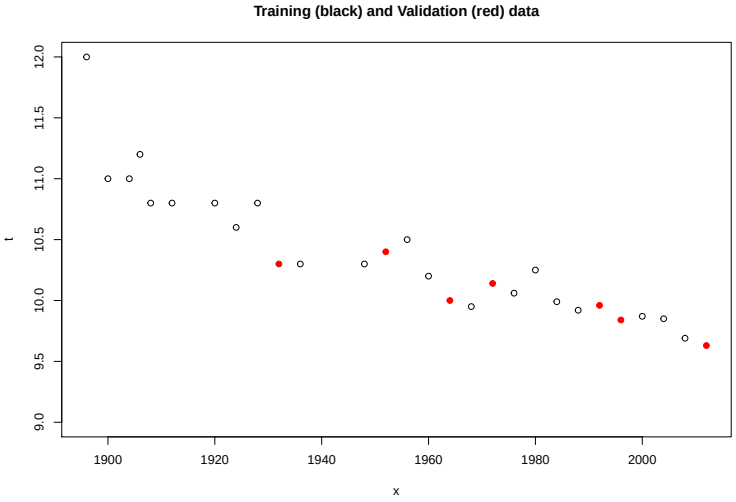
Copyright Miles Chen. For personal use only. Do not distribute.

Quadratic model



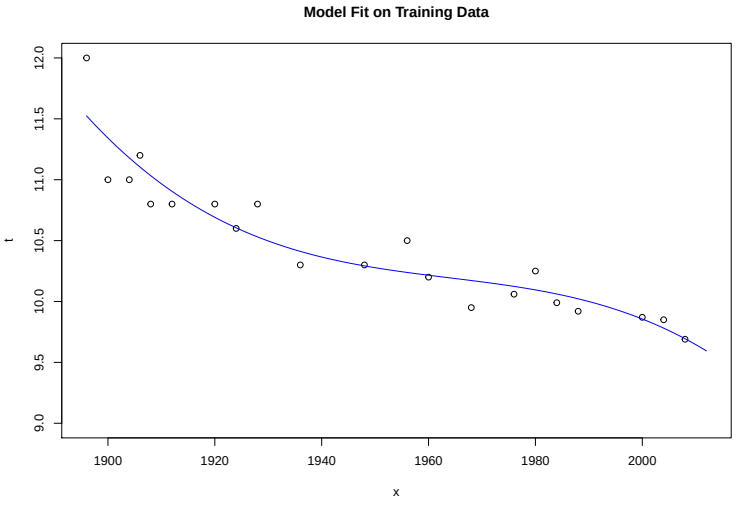
Copyright Miles Chen. For personal use only. Do not distribute.

Validation Set C for reference



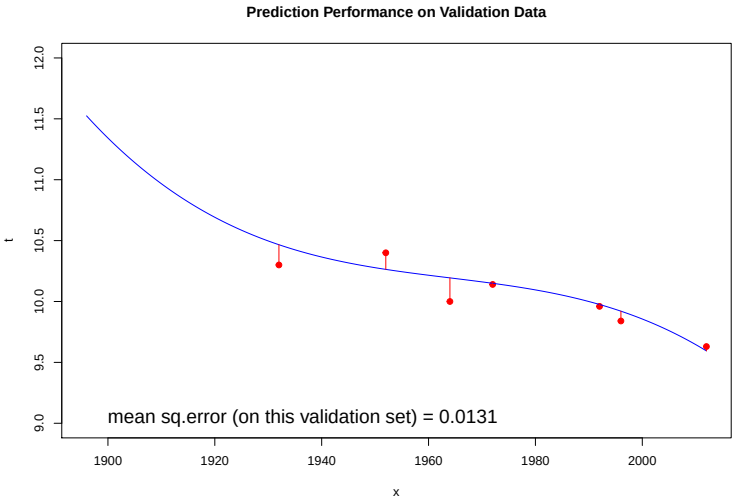
Copyright Miles Chen. For personal use only. Do not distribute.

Cubic model



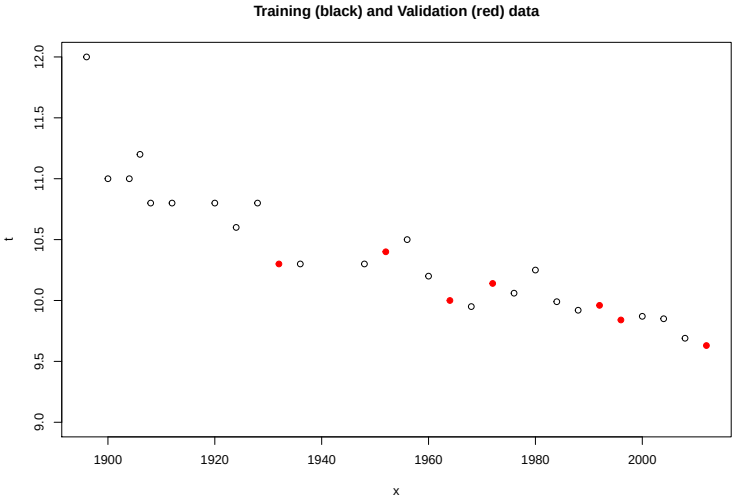
Copyright Miles Chen. For personal use only. Do not distribute.

Cubic model



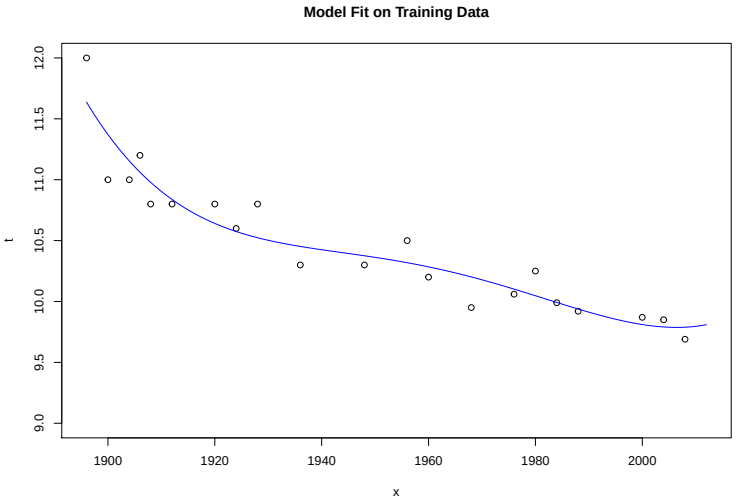
Copyright Miles Chen. For personal use only. Do not distribute.

Validation Set C for reference



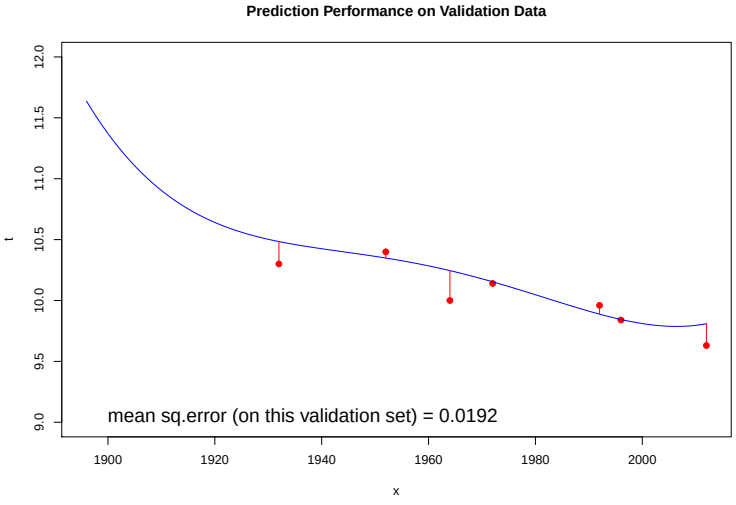
Copyright Miles Chen. For personal use only. Do not distribute.

4-th order polynomial model



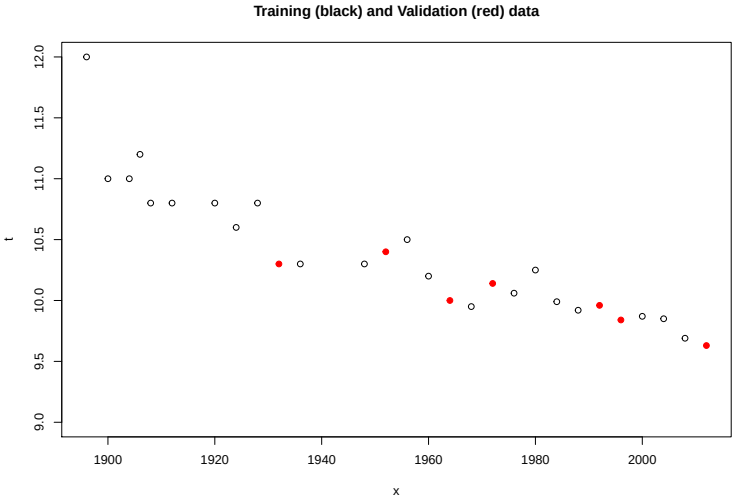
Copyright Miles Chen. For personal use only. Do not distribute.

4-th order polynomial model



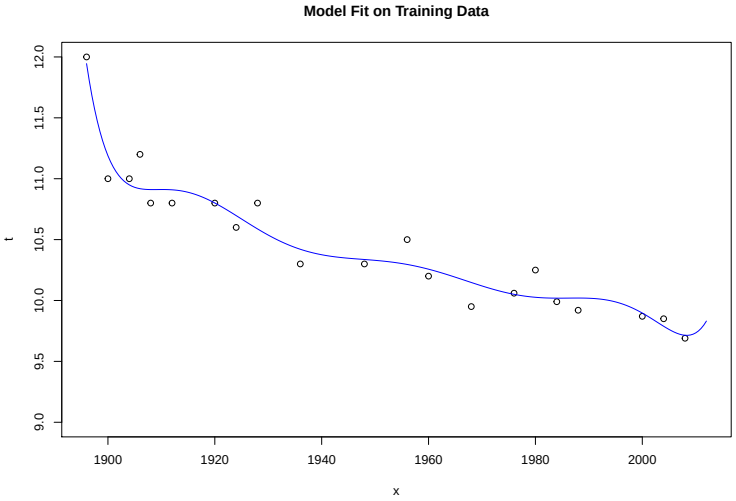
Copyright Miles Chen. For personal use only. Do not distribute.

Validation Set C for reference



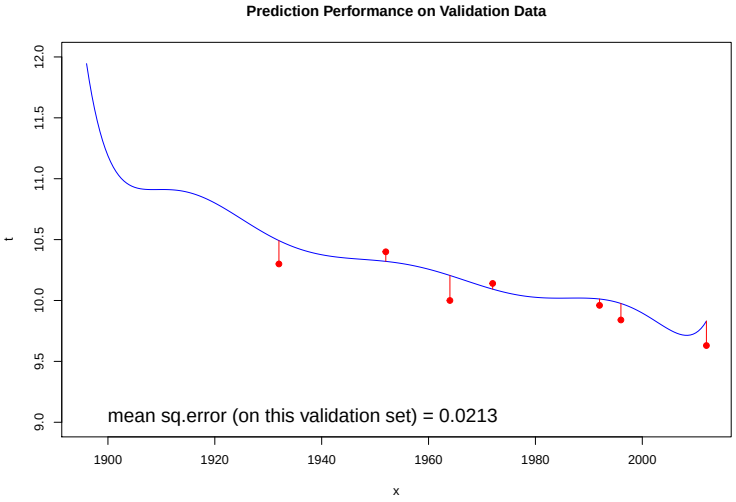
Copyright Miles Chen. For personal use only. Do not distribute.

8-th order polynomial model



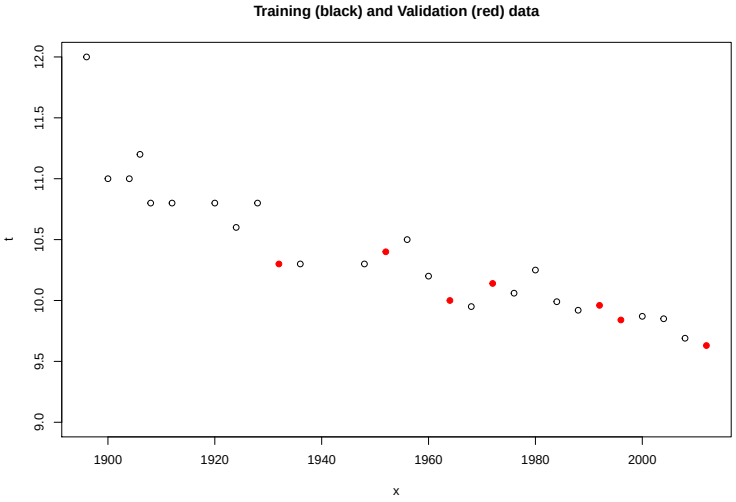
Copyright Miles Chen. For personal use only. Do not distribute.

8-th order polynomial model



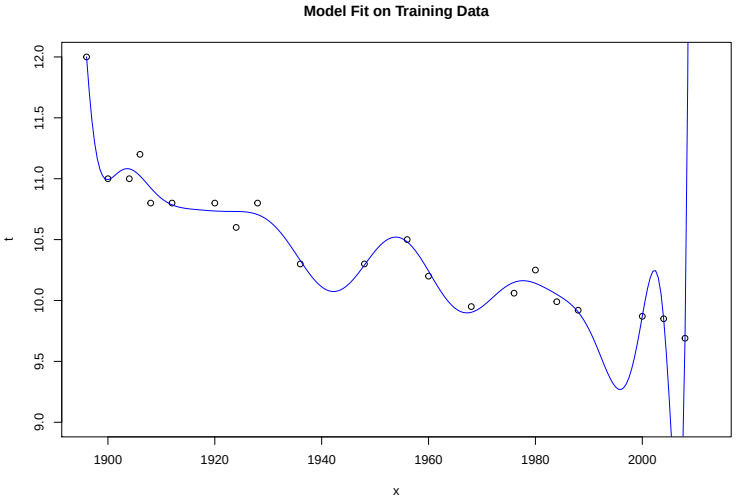
Copyright Miles Chen. For personal use only. Do not distribute.

Validation Set C for reference



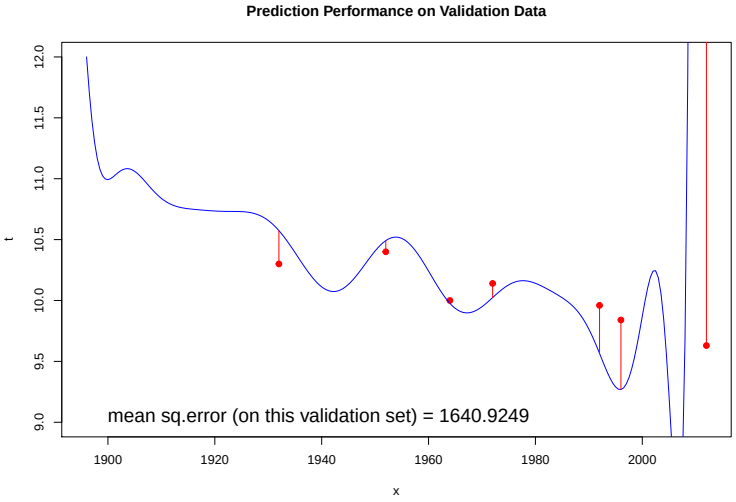
Copyright Miles Chen. For personal use only. Do not distribute.

17-th order polynomial model, just for fun



Copyright Miles Chen. For personal use only. Do not distribute.

17-th order polynomial model



Copyright Miles Chen. For personal use only. Do not distribute.

CV MSE of the different models

```
cv_results <- rbind(cv_line, cv_quad, cv_ord3, cv_ord4, cv_ord5, cv_ord8, cv_ord17)
cv_results <- round(cbind(cv_results, rowMeans(cv_results)), 4)
colnames(cv_results) <- c("Val_Set_A","Val_Set_B","Val_Set_C","Val_Set_D", "CV_Score"
cv_results
```

```
##           Val_Set_A Val_Set_B Val_Set_C Val_Set_D CV_Score
## cv_line      0.0224    0.0258    0.0319    0.1896   0.0674
## cv_quad      0.0253    0.0371    0.0250    0.1625   0.0624
## cv_ord3      0.0460    0.0290    0.0131    0.1403   0.0571
## cv_ord4      0.0567    0.0216    0.0192    0.1669   0.0661
## cv_ord5      0.0518    0.0104    0.0271    0.1684   0.0644
## cv_ord8      0.0334    0.0320    0.0213    0.0809   0.0419
## cv_ord17    91.3581  291.3164 1640.9249   77.8702 525.3674
```

Based on these results, it seems the 8th order polynomial has the lowest average CV SE, and the 3rd order polynomial has the second lowest average CV SE. We would pause and think about which model we choose, and I would argue that we should use the 3rd order polynomial.

Copyright Miles Chen. For personal use only. Do not distribute.

K-fold cross-validation and randomness

K-fold cross validation is sensitive to what values are randomly put into each validation set. Here, I rerun the same code, but with a different starting seed.

```
set.seed(7)
```

```
##           Val_Set_A Val_Set_B Val_Set_C  Val_Set_D   CV_Score
## cv_line      0.0392    0.0201    0.0329     0.1606     0.0632
## cv_quad      0.0346    0.0215    0.0297     0.1376     0.0559
## cv_ord3      0.0352    0.0124    0.0285     0.1095     0.0464
## cv_ord4      0.0452    0.0169    0.0303     0.1478     0.0600
## cv_ord5      0.0473    0.0255    0.0193     0.1827     0.0687
## cv_ord8      0.0354    0.0573    0.0569     0.2342     0.0960
## cv_ord17    10.0741    0.6417   21.5110 53031.3931 13265.9050
```

The third order polynomial still performs the best, but the CV error values vary for each validation set. Here, the 8th order polynomial has the second worst performance.

Copyright Miles Chen. For personal use only. Do not distribute.

K-fold cross-validation and randomness

```
set.seed(10)
```

```
##          Val_Set_A Val_Set_B  Val_Set_C  Val_Set_D   CV_Score
## cv_line     0.0721    0.0205     0.0113     0.1793     0.0708
## cv_quad     0.0428    0.0181     0.0583     0.1451     0.0661
## cv_ord3     0.0217    0.0155     0.0746     0.1206     0.0581
## cv_ord4     0.0310    0.0254     0.0701     0.1595     0.0715
## cv_ord5     0.0606    0.0210     0.0588     0.1631     0.0759
## cv_ord8     0.0435    0.0244     0.0360     0.1370     0.0603
## cv_ord17  142.4144    0.1184 22541.0040 74772.1052 24363.9105
```

The 3rd order polynomial has the best performance again. As we have seen, depending on the seed, it's possible to get a cross-validation result where a sub-optimal model happens to perform best. If you are able to afford the computational expense, it is recommended to run cross-validation more than once.

Copyright Miles Chen. For personal use only. Do not distribute.

CV with `library(boot)`

- Use `glm()` to fit a model
- Use `cv.glm()` to perform cross validation. The `K` argument specifies the number of folds. If left blank, it will use $K = N$, or leave-one-out CV.
- The 'score' provided is the average squared error. You can also provide another function to calculate the score or cost.

On the next few slides, I run `cv.glm` a few times for each model. There is some variation between runs because the random blocks that get created will differ from run to run.

On the other hand, when you do LOOCV, there is no variation between runs because all points serve as the validation set one at a time. So every run will have the same validation sets.

Copyright Miles Chen. For personal use only. Do not distribute.

4-fold cross-validation on a linear model

```
library(boot)
model <- glm(time ~ poly(year,1), data = olympic)
cv.glm(olympic, model, K = 4)$delta[1]
```

```
## [1] 0.06436664
```

```
cv.glm(olympic, model, K = 4)$delta[1]
```

```
## [1] 0.06598626
```

```
cv.glm(olympic, model, K = 4)$delta[1]
```

```
## [1] 0.05766628
```

Note the slight variation in CV scores each time I run it. The variation exists because each time we split the data into K-folds, slightly different values end up in each validation set.

Copyright Miles Chen. For personal use only. Do not distribute.

4-fold cross-validation on a quadratic model

```
model2 <- glm(time ~ poly(year,2), data = olympic)
cv.glm(olympic, model2, K = 4)$delta[1]
```

```
## [1] 0.0620874
```

```
cv.glm(olympic, model2, K = 4)$delta[1]
```

```
## [1] 0.04697625
```

```
cv.glm(olympic, model2, K = 4)$delta[1]
```

```
## [1] 0.06483293
```

Copyright Miles Chen. For personal use only. Do not distribute.

4-fold cross-validation on a cubic model

```
model3 <- glm(time ~ poly(year,3), data = olympic)
cv.glm(olympic, model3, K = 4)$delta[1]
```

```
## [1] 0.04923011
```

```
cv.glm(olympic, model3, K = 4)$delta[1]
```

```
## [1] 0.0644752
```

```
cv.glm(olympic, model3, K = 4)$delta[1]
```

```
## [1] 0.07093559
```

Copyright Miles Chen. For personal use only. Do not distribute.

LOOCV on a cubic model

```
model3 <- glm(time ~ poly(year,3), data = olympic)
cv.glm(olympic, model3)$delta[1]
```

```
## [1] 0.05017418
```

```
cv.glm(olympic, model3)$delta[1]
```

```
## [1] 0.05017418
```

```
cv.glm(olympic, model3)$delta[1]
```

```
## [1] 0.05017418
```

For LOOCV, the validation sets are identical (each point serves as a validation set exactly once). Thus, all three runs produce the same value.

Copyright Miles Chen. For personal use only. Do not distribute.

Cross-Validation Summary (pieces taken from wikipedia)

Cross-validation is a method to see how a particular model will generalize to new data.

The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it in order to flag problems like overfitting.

Multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model's predictive performance.

Copyright Miles Chen. For personal use only. Do not distribute.