DS-6030 Homework Module 4

Matt Scheffel

```
options(tinytex.verbose = TRUE)
```

DS 6030 | Spring 2022 | University of Virginia

3. We now review k-fold cross-validation.

(a) Explain how k-fold cross-validation is implemented.

k-fold cross-validation is a technique that divides a data set into k equally sized partitions/folds. The model is then trained on k-1 of these folds, and the remaining fold is used for testing. The process is repeated k times, with a different group of observations treated as a validation set each time. The average performance of the model is calculated over k iterations.

(b) What are the advantages and disadvantages of k-fold cross-validation relative to:

i. The validation set approach?

The advantages of the k-fold cross-validation relative to the validation set approach include that it provides a more reliable estimate of the model's performance, uses multiple partitions of the data for both training and validation, makes better use of limited data, and reduces the variance of the performance estimate.

The disadvantages of the k-fold cross-validation relative to the validation set approach include that it can be burdensome/expensive for very large data sets or complex models with long training times and it struggles to produce a reliable estimate of performance with small data sets as well.

ii. LOOCV?

The advantages of the k-fold cross-validation relative to LOOCV include that it requires fewer model fits than LOOCV, tends to produce a more reliable estimate of the model's performance if the data set is noisy or includes outliers, and can provide an estimate of the variance of the performance estimate.

The disadvantages of the k-fold cross-validation relative to LOOCV include that it may produce a less biased estimate of the model's performance, does not work well with small data sets, and is liable to be impacted by the choice of k.

5. In Chapter 4, we used logistic regression to predict the probability of `default` using `income` and `balance` on the `Default` data set.

We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

(a) Fit a logistic regression model that uses `income` and `balance` to predict `default`.

```
library(ISLR2)

set.seed(1)
```

```
model1 = glm(default ~ income + balance, data = Default, family = "binomial")
summary(model1)
```

```
#>
#> Call:
#> glm(formula = default ~ income + balance, family = "binomial",
#>     data = Default)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -2.4725  -0.1444  -0.0574  -0.0211   3.7245
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
#> income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
#> balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 2920.6  on 9999  degrees of freedom
#> Residual deviance: 1579.0  on 9997  degrees of freedom
#> AIC: 1585
#>
#> Number of Fisher Scoring iterations: 8
```

(b) Using the validation set approach, estimate the test error of this model. In order to do this, you must perform the following steps:

i. Split the sample set into a training set and a validation set.

```
training_data = sample(dim(Default)[1], dim(Default)[1] / 2)
```

ii. Fit a multiple logistic regression model using only the training observations.

```
model2 = glm(default ~ income + balance, data = Default[train,], family = "binomial")
```

```
#> Error in `[.data.frame`(Default, train, ): object 'train' not found
```

```
model2 = glm(default ~ income + balance, data = Default, family = "binomial", subset = training_data)

summary(fit.glm)
```

```
#> Error in summary(fit.glm): object 'fit.glm' not found
```

iii. Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the `default` category if the posterior probability is greater than 0.5.

```
model.probs = predict(model2, newdata = Default[-training_data, ], type="response")
model.pred=rep("No", 5000)
model.pred[model.probs > 0.5] = "Yes"
```

iv. Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.

```
mean(model.pred != Default[-training_data, ]$default)
```

```
#> [1] 0.0254
```

(c) Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```
training_data2 <- sample(dim(Default)[1], dim(Default)[1] / 2)
model3 <- glm(default ~ income + balance, data = Default, family = "binomial", subset = training_data2)
model.probs2 <- predict(model3, newdata = Default[-training_data2, ], type = "response")
model.pred2 <- rep("No", length(model.probs2))
model.pred2[model.probs2 > 0.5] <- "Yes"
mean(model.pred2 != Default[-training_data2, ]$default)
```

```
#> [1] 0.0274
```

```
training_data2 <- sample(dim(Default)[1], dim(Default)[1] / 2)
model3 <- glm(default ~ income + balance, data = Default, family = "binomial", subset = training_data2)
model.probs2 <- predict(model3, newdata = Default[-training_data2, ], type = "response")
model.pred2 <- rep("No", length(model.probs2))
model.pred2[model.probs2 > 0.5] <- "Yes"
mean(model.pred2 != Default[-training_data2, ]$default)
```

```
#> [1] 0.0244
```

```
training_data2 <- sample(dim(Default)[1], dim(Default)[1] / 2)
model3 <- glm(default ~ income + balance, data = Default, family = "binomial", subset = training_data2)
model.probs2 <- predict(model3, newdata = Default[-training_data2, ], type = "response")
model.pred2 <- rep("No", length(model.probs2))
model.pred2[model.probs2 > 0.5] <- "Yes"
mean(model.pred2 != Default[-training_data2, ]$default)
```

```
#> [1] 0.0244
```

The results obtained demonstrate that the validation estimate of the test error rate can be variable. This depends on which observations are included in the training set as well as which observations are included in the validation set.

(d) Now consider a logistic regression model that predicts the probability of `default` using `income`, `balance`, and a dummy variable for `student`. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for student leads to a reduction in the test error rate.

```
training_data3 <- sample(dim(Default)[1], dim(Default)[1] / 2)
model4 <- glm(default ~ income + balance + student, data = Default, family = "binomial", subset = train
model.probs3 <- predict(model4, newdata = Default[-training_data3, ], type = "response")
model.pred3 <- rep("No", length(model.probs3))
model.pred3[model.probs3 > 0.5] <- "Yes"
mean(model.pred3 != Default[-training_data3, ]$default)
```

```
#> [1] 0.0278
```

Adding a dummy variable does not appear to lead to a reduction in the test error rate.

6. We continue to consider the use of a logistic regression model to predict the probability of `default` using `income` and `balance` on the `Default` data set.

In particular, we will now compute estimates for the standard errors of the `income` and `balance` logistic regression coefficients in two different ways: (1) using the bootstrap, and (2) using the standard formula for computing the standard errors in the `glm()` function. Do not forget to set a random seed before beginning your analysis.

(a) Using the `summary()` and `glm()` functions, determine the estimated standard errors for the coefficients associated with `income` and `balance` in a multiple logistic regression model that uses both predictors.

```
set.seed(1)
new_training_data <- sample(dim(Default)[1], dim(Default)[1] / 2)
new_model <- glm(default ~ income + balance, data = Default, family = "binomial", subset = new_training_
summary(new_model)
```

```
#>
#> Call:
#> glm(formula = default ~ income + balance, family = "binomial",
#>     data = Default, subset = new_training_data)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -2.5830  -0.1428  -0.0573  -0.0213   3.3395
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -1.194e+01  6.178e-01 -19.333  < 2e-16 ***
#> income       3.262e-05  7.024e-06   4.644 3.41e-06 ***
#> balance      5.689e-03  3.158e-04  18.014  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 1523.8  on 4999  degrees of freedom
#> Residual deviance:  803.3  on 4997  degrees of freedom
#> AIC: 809.3
#>
#> Number of Fisher Scoring iterations: 8
```

(b) Write a function, `boot.fn()`, that takes as input the `Default` data set as well as an index of the observations, and that outputs the coefficient estimates for income and balance in the multiple logistic regression model.

```
boot.fn <- function(data, index) {
    fit <- glm(default ~ income + balance, data = data, family = "binomial", subset = index)
    return (coef(fit))
}
```

(c) Use the `boot()` function together with your `boot.fn()` function to estimate the standard errors of the logistic regression coefficients for `income` and `balance`.

```
library(boot)
boot(Default, boot.fn, 1000)
```

```
#>
#> ORDINARY NONPARAMETRIC BOOTSTRAP
#>
#>
#> Call:
#> boot(data = Default, statistic = boot.fn, R = 1000)
#>
#>
#> Bootstrap Statistics :
#>          original        bias     std. error
#> t1* -1.154047e+01 -3.912114e-02 4.347403e-01
#> t2*  2.080898e-05  1.585717e-07 4.858722e-06
#> t3*  5.647103e-03  1.856917e-05 2.300758e-04
```

The bootstrap estimates of the standard errors for the coefficients β_0, β_1, and β_2 are 4.347403e-01, 4.858722e-06 and 2.300758e-04.

(d) Comment on the estimated standard errors obtained using the `glm()` function and using your bootstrap function.

The estimated standard errors obtained by both the `glm()` function method and the bootstrap function method are pretty close in value.

7. In Sections 5.3.2 and 5.3.3, we saw that the `cv.glm()` function can be used in order to compute the LOOCV test error estimate.

Alternatively, one could compute those quantities using just the `glm()` and `predict.glm()` functions, and a for loop. You will now take this approach in order to compute the LOOCV error for a simple logistic regression model on the Weekly data set. Recall that in the context of classification problems, the LOOCV error is given in (5.4).

(a) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2`.

```
model_lr = glm(Direction ~ Lag1 + Lag2, data = Weekly, family = binomial)
summary(model_lr)
```

```
#>
#> Call:
#> glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Weekly)
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -1.623  -1.261   1.001   1.083   1.506
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.22122    0.06147   3.599 0.000319 ***
#> Lag1        -0.03872    0.02622  -1.477 0.139672
#> Lag2         0.06025    0.02655   2.270 0.023232 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 1496.2  on 1088  degrees of freedom
#> Residual deviance: 1488.2  on 1086  degrees of freedom
```

```
#> AIC: 1494.2
#>
#> Number of Fisher Scoring iterations: 4
```

(b) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2` using all but the first observation.

```
model_lr2 = glm(Direction ~ Lag1 + Lag2, data = Weekly[-1,], family = binomial)
summary(model_lr2)
```

```
#>
#> Call:
#> glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Weekly[-1,
#>     ])
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -1.6258  -1.2617   0.9999   1.0819   1.5071
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.22324    0.06150   3.630 0.000283 ***
#> Lag1        -0.03843    0.02622  -1.466 0.142683
#> Lag2         0.06085    0.02656   2.291 0.021971 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 1494.6  on 1087  degrees of freedom
#> Residual deviance: 1486.5  on 1085  degrees of freedom
#> AIC: 1492.5
#>
#> Number of Fisher Scoring iterations: 4
```

(c) Use the model from (b) to predict the direction of the first observation. You can do this by predicting that the first observation will go up if $P(\text{Direction}=\text{“Up”}|\text{Lag1, Lag2}) > 0.5$. Was this observation correctly classified?

```
predict(model_lr2, newdata = Weekly[1,], type = "response") > 0.5
```

```
#>    1
#> TRUE
```

```
Weekly[1,]$Direction
```

```
#> [1] Down
#> Levels: Down Up
```

The prediction for the first observation is Up. However, it was not correctly classified. The correct direction is Down.

(d) Write a `for` loop from $i = 1$ to $i = n$, where n is the number of observations in the data set, that performs each of the following steps:

i. Fit a logistic regression model using all but the i-th observation to predict `Direction` using `Lag1` and `Lag2`.

ii. Compute the posterior probability of the market moving up for the i-th observation.

iii. Use the posterior probability for the i-th observation in order to predict whether or not the market moves up.

iv. Determine whether or not an error was made in predicting the direction for the i-th observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0.

```
error_loop <- rep(0, dim(Weekly)[1])
for (i in 1:dim(Weekly)[1]) {
    new_fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-i, ],  family = "binomial")
    loop.pred <- predict.glm(new_fit, Weekly[i, ], type = "response") > 0.5
    up_true <- Weekly[i, ]$Direction == "Up"
    if (loop.pred != up_true)
        error_loop[i] <- 1
}
error_loop
```

```
#>    [1] 1 1 0 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 0
#>   [38] 0 1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0
#>   [75] 0 1 0 1 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1
#>  [112] 1 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0
#>  [149] 0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0
#>  [186] 0 0 1 1 0 1 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1
#>  [223] 0 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 0
#>  [260] 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0
#>  [297] 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0
#>  [334] 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 0 1
#>  [371] 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 1
#>  [408] 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1
#>  [445] 0 1 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 1 0 0
#>  [482] 0 1 1 1 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0
#>  [519] 1 1 0 0 1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 1 1 1 1
#>  [556] 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 0 1
#>  [593] 0 0 1 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1
#>  [630] 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 0
#>  [667] 0 1 1 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1
#>  [704] 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
#>  [741] 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1
#>  [778] 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 1
#>  [815] 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 0 1 0 1 1 0 0 1
#>  [852] 1 1 0 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1
#>  [889] 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 0 0 1
#>  [926] 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 1 1 1 1 1 0 1
#>  [963] 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 0 0 0
#> [1000] 0 1 0 0 1 0 1 0 0 1 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 0
#> [1037] 1 0 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 1 0 0
#> [1074] 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0
```

(e) Take the average of the n numbers obtained in (d) in order to obtain the LOOCV estimate for the test error. Comment on the results.

```
mean(error_loop)
```

```
#> [1] 0.4499541
```

8. We will now perform cross-validation on a simulated data set.

(a) Generate a simulated data set as follows:

```
set.seed(1)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

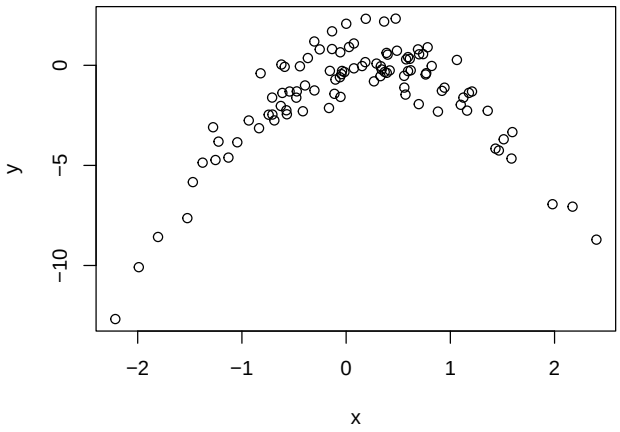
In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

For this data set: $n = 100$ and $p = 2$. The model used to generate the data is:

$Y = X$ - $2X^2 + \epsilon$.

(b) Create a scatterplot of `x` against `y`. Comment on what you find.

```
plot(x,y)
```



The scatter plot demonstrates a downward sloping curve, indicating that our data is non-linear. I do not notice any extreme outliers. The relationship between X and Y appears to be quadratic.

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

i. $Y = \beta_0 + \beta_1 X + \epsilon$

```
set.seed(5)
Data <- data.frame(x, y)
random_model_1 <- glm(y ~ x)
cv.glm(Data, random_model_1)$delta[1]
```

```
#> [1] 7.288162
```

ii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

```
random_model_2 <- glm(y ~ poly(x, 2))
cv.glm(Data, random_model_2)$delta[1]
```

```
#> [1] 0.9374236
```

iii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

```
random_model_3 <- glm(y ~ poly(x, 3))
cv.glm(Data, random_model_3)$delta[1]
```

```
#> [1] 0.9566218
```

iv. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$.

```
random_model_4 <- glm(y ~ poly(x, 4))
cv.glm(Data, random_model_4)$delta[1]
```

```
#> [1] 0.9539049
```

Note you may find it helpful to use the `data.frame()` function to create a single data set containing both `X` and `Y`.

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```
set.seed(20)
random_model_1 <- glm(y ~ x)
cv.glm(Data, random_model_1)$delta[1]
```

```
#> [1] 7.288162
```

```
random_model_2 <- glm(y ~ poly(x, 2))
cv.glm(Data, random_model_2)$delta[1]
```

```
#> [1] 0.9374236
```

```
random_model_3 <- glm(y ~ poly(x, 3))
cv.glm(Data, random_model_3)$delta[1]
```

```
#> [1] 0.9566218
```

```
random_model_4 <- glm(y ~ poly(x, 4))
cv.glm(Data, random_model_4)$delta[1]
```

```
#> [1] 0.9539049
```

Yes, the results are the same as in (c). This is because the LOOCV calculation evaluates n folds of a single observation.

(e) Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

The 2nd model had the smallest LOOCV error. This is to be expected as the scatter plot showed a quadratic relationship between X and Y.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

```
summary(random_model_4)
```

```
#>
#> Call:
#> glm(formula = y ~ poly(x, 4))
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -2.0550  -0.6212  -0.1567   0.5952   2.2267
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
#> poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
#> poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
```

```
#> poly(x, 4)3   0.26411    0.95905   0.275    0.784
#> poly(x, 4)4   1.25710    0.95905   1.311    0.193
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for gaussian family taken to be 0.9197797)
#>
#>     Null deviance: 700.852  on 99  degrees of freedom
#> Residual deviance:  87.379  on 95  degrees of freedom
#> AIC: 282.3
#>
#> Number of Fisher Scoring iterations: 2
```

The resulting p-values show us that the first (linear) and second (quadratic) terms are statistically significant. However, we see that the third (cubic) and fourth terms are not statistically significant. The results do agree with the conclusions drawn based on the cross-validation results.