Ex6

Yelyzaveta Klysa

2023-10-17

Contents

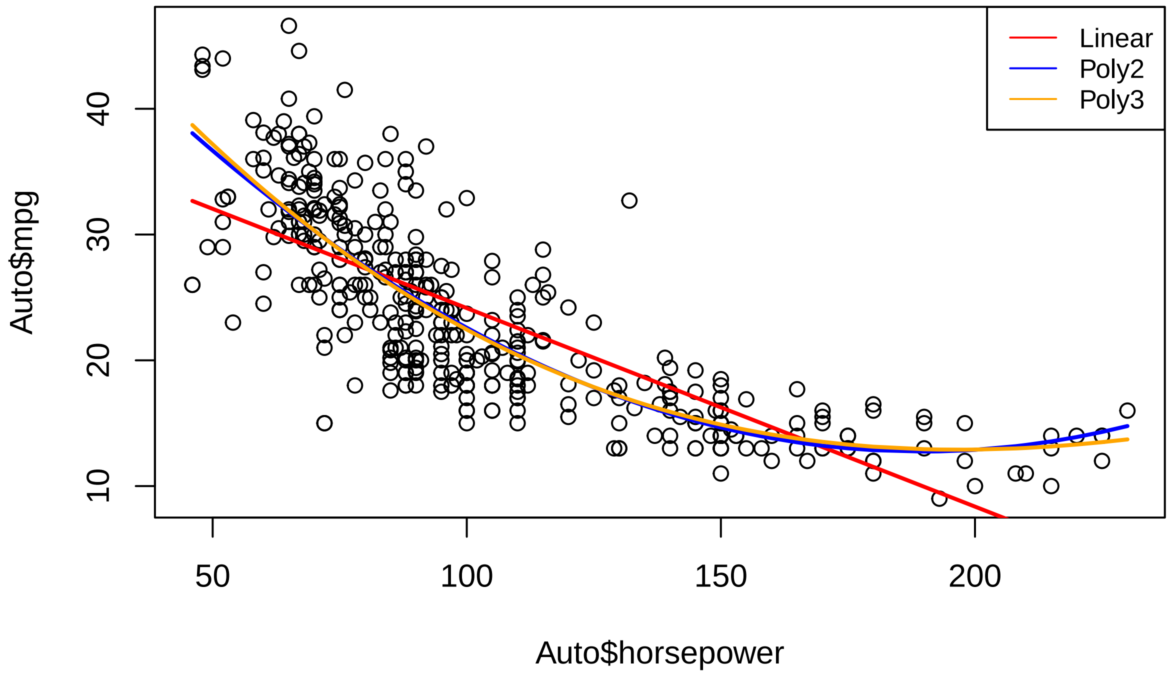
Task 1

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

1.1 Fit the following models

```
Auto <- arrange(Auto, horsepower)
ml1_all <- lm(mpg ~ horsepower, data=Auto)
ml2_all <- lm(mpg ~ poly(horsepower,2), data=Auto)
ml3_all <- lm(mpg ~ poly(horsepower,3), data=Auto)
```

```r
plot(Auto$horsepower, Auto$mpg)
lines(Auto$horsepower, fitted(ml1_all), col="red", lwd=2)
lines(Auto$horsepower, fitted(ml2_all), col="blue", lwd=2)
lines(Auto$horsepower, fitted(ml3_all), col="orange", lwd=2)
legend("topright", legend=c("Linear", "Poly2", "Poly3"), col=c("red", "blue", "orange"),
       lty=1, cex=0.8)
```



1.2 Use the validation set approach to compare the models. Use once a train/test split of 50%/50% and once 70%/30%. Choose the best model based on Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation.

```r
n <- nrow(Auto)
set.seed(12208877)

train_50_inds <- sample(1:n,n*0.5)
train_70_inds <- sample(1:n,n*0.7)

train_50 <- Auto[train_50_inds,]
test_50 <- Auto[-train_50_inds,]
train_70 <- Auto[train_70_inds,]
test_30 <- Auto[-train_70_inds,]
train_sets <- list(train_50, train_70)
test_sets <- list(test_50, test_30)
rmse <- c()
mse <- c()
```

```r
mad <- c()

for (i in 1:2){
  ml1 <- glm(mpg ~ horsepower, data=train_sets[[i]])
  ml2 <- glm(mpg ~ poly(horsepower,2), data=train_sets[[i]])
  ml3 <- glm(mpg ~ poly(horsepower,3), data=train_sets[[i]])

  predicts <- list(predict(ml1, test_sets[[i]]),
                   predict(ml2, test_sets[[i]]),
                   predict(ml3, test_sets[[i]]))
  for (j in 1:3){
    rmse[j+3*(i-1)] <- rmse(test_sets[[i]]$mpg, predicts[[j]])
    mse[j+3*(i-1)] <- mse(test_sets[[i]]$mpg, predicts[[j]])
    mad[j+3*(i-1)] <- mad(test_sets[[i]]$mpg, predicts[[j]])
  }
}

results <- data.frame(
  settings = c("50/50", "50/50", "50/50", "70/30", "70/30", "70/30"),
  models=c("Linear", "Poly2", "Poly3","Linear", "Poly2", "Poly3"),
  rmse=rmse,
  mse=mse,
  mad=mad
)
results
```

```
##   settings models     rmse      mse      mad
## 1    50/50 Linear 5.056621 25.56942 4.893841
## 2    50/50  Poly2 4.330621 18.75428 3.913620
## 3    50/50  Poly3 4.330894 18.75664 3.707499
## 4    70/30 Linear 5.324968 28.35528 5.193883
## 5    70/30  Poly2 4.794296 22.98528 4.260327
## 6    70/30  Poly3 4.848537 23.50831 4.263516
```

Based on the results, the second model seems to have the smallest errors on both options of training/test division. Though taking 50% of data for testing gives better results.

1.3 Use the cv.glm function in the boot package for the following steps.

Use cv.glm for Leave-one-out Cross Validation to compare the models above. Use cv.glm for 5-fold and 10-fold Cross Validation to compare the models above.

```r
mod1 <- glm(mpg ~ horsepower, data=Auto)
mod2 <- glm(mpg ~ poly(horsepower,2), data=Auto)
mod3 <- glm(mpg ~ poly(horsepower,3), data=Auto)
cv_1out_1 <- cv.glm(Auto, mod1)$delta[1]
cv_1out_2 <- cv.glm(Auto, mod2)$delta[1]
cv_1out_3 <- cv.glm(Auto, mod3)$delta[1]

cv_5_1 <- cv.glm(Auto, mod1, K=5)$delta[1]
cv_5_2 <- cv.glm(Auto, mod2, K=5)$delta[1]
cv_5_3 <- cv.glm(Auto, mod3, K=5)$delta[1]

cv_10_1 <- cv.glm(Auto, mod1, K=10)$delta[1]
cv_10_2 <- cv.glm(Auto, mod2, K=10)$delta[1]
```

```
cv_10_3 <- cv.glm(Auto, mod3, K=10)$delta[1]

cv_results <- data.frame(
  models=c("Linear", "Poly2", "Poly3"),
  l1out_mse=c(cv_1out_1, cv_1out_2, cv_1out_3),
  cv5_mse=c(cv_5_1, cv_5_2, cv_5_3),
  cv10_mse=c(cv_10_1, cv_10_2, cv_10_3)
)
cv_results
```

```
##   models l1out_mse  cv5_mse cv10_mse
## 1 Linear  24.23151 24.11093 24.23136
## 2  Poly2  19.24821 19.11870 19.13962
## 3  Poly3  19.33498 19.07725 19.53707
```

1.4 Compare all results from 2 and 3. in a table and draw your conclusions.

The results of cross-validation confirm the results of Task 1.2 that the second model represents the trend of data the the best for `leave-one-out` and `k=10` methods. However, for the case `k=5`, polynomial 3 has slightly smaller error. In the plot, the 2 functions were quite similar, so I would choose the polynomial 2 model to avoid overfitting.

Task 2

Load the data set 'economics' from the package 'ggplot2'.

2.1 - 2.2 Fit the following models to explain the number of unemployed persons 'unemploy' by the median number of days unemployed 'uempmed' and vice versa:

linear model an appropriate exponential or logarithmic model (which one is appropriate depends on which is the dependent or independent variable) polynomial model of 2nd, 3rd and 10th degree

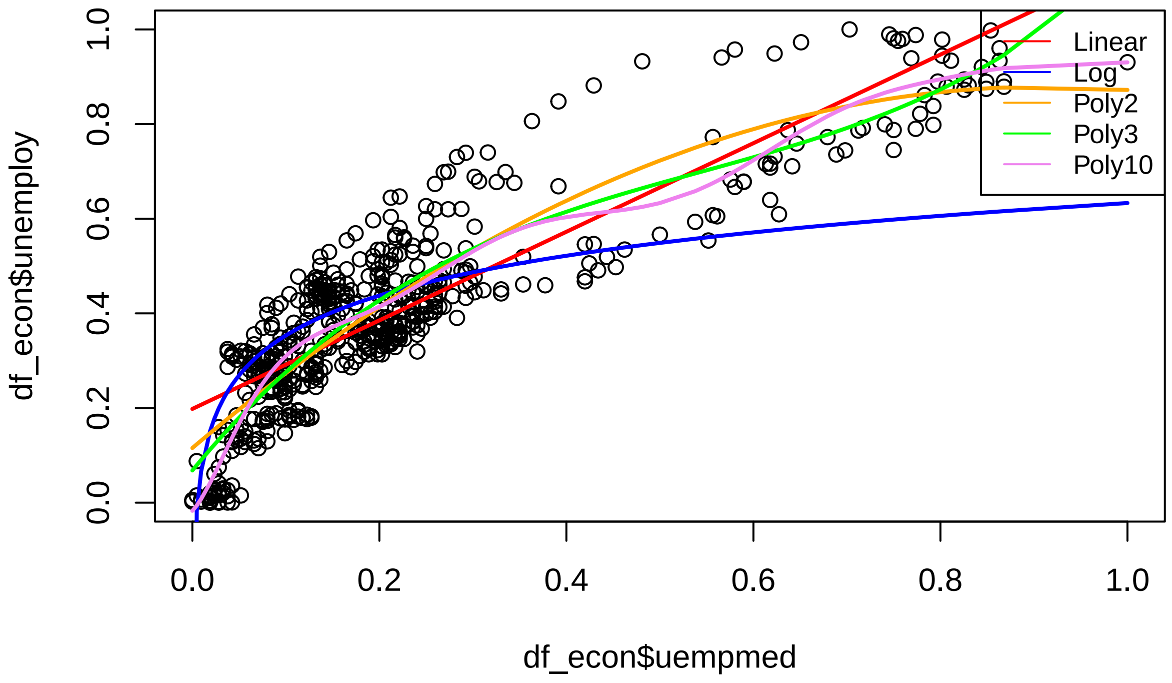
Plot the corresponding data and add all the models for comparison.

In order for exp or log to work on the data, we need to scale the variables with min max scale and introduce a slight error (for exp to work in case of Nan/Inf..).

unemploy ~ uempmed

```
data(economics)
df_econ <- economics
eps <- 0.0000001
df_econ$unemploy <- (df_econ$unemploy - min(df_econ$unemploy)) /
  (max(df_econ$unemploy) - min(df_econ$unemploy)) + eps
df_econ$uempmed <- (df_econ$uempmed - min(df_econ$uempmed)) /
  (max(df_econ$uempmed) - min(df_econ$uempmed)) + eps

df_econ <- arrange(df_econ, uempmed)
plot(df_econ$uempmed, df_econ$unemploy)

ml_linear_unue <- glm(unemploy ~ uempmed, data=df_econ)
ml_log_unue <- glm(unemploy ~ log(uempmed), data = df_econ)
ml_poly2_unue <- glm(unemploy ~ poly(uempmed,2), data=df_econ)
ml_poly3_unue <- glm(unemploy ~ poly(uempmed,3), data=df_econ)
```

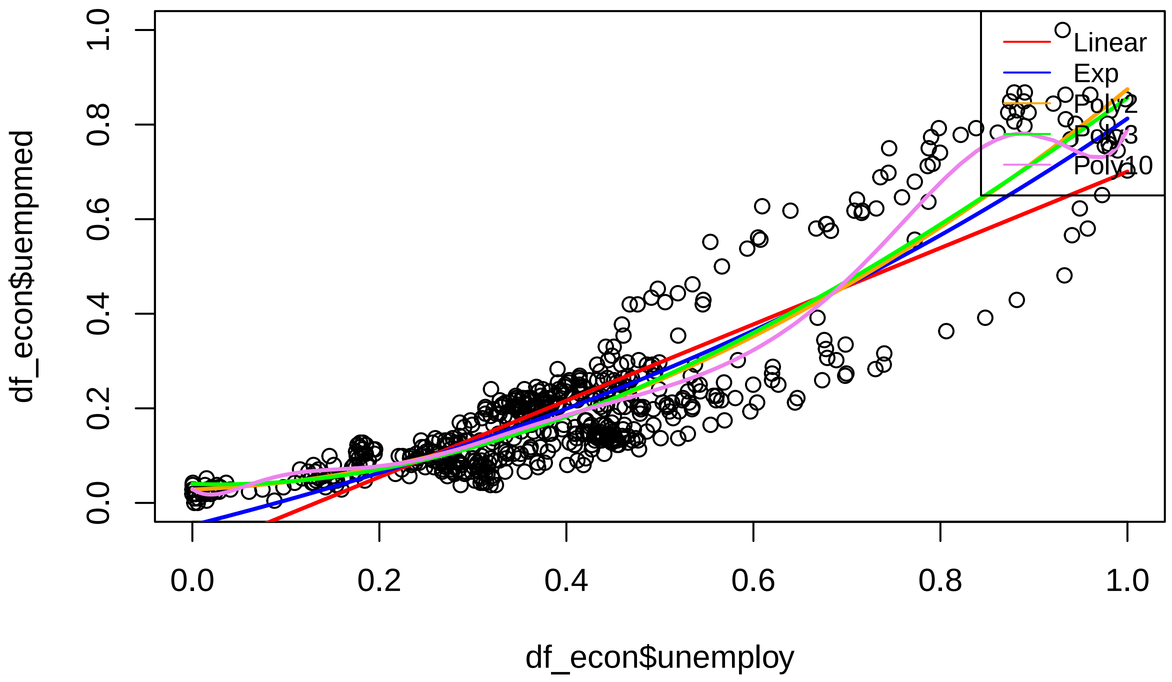
```r
ml_poly10_unue <- glm(unemploy ~ poly(uempmed,10), data=df_econ)

lines(df_econ$uempmed, fitted(ml_linear_unue), col="red", lwd=2)
lines(df_econ$uempmed, fitted(ml_log_unue), col="blue", lwd=2)
lines(df_econ$uempmed, fitted(ml_poly2_unue), col="orange", lwd=2)
lines(df_econ$uempmed, fitted(ml_poly3_unue), col="green", lwd=2)
lines(df_econ$uempmed, fitted(ml_poly10_unue), col="violet", lwd=2)
legend("topright", legend=c("Linear", "Log", "Poly2", "Poly3", "Poly10"),
       col=c("red", "blue", "orange", "green", "violet"), lty=1, cex=0.8)
```



It can be clearly seen that simple linear model is too simple for the problem here, while polynomial 10 is too flexible. Log function seems to not capture the relationship at all.

uempmed ~ unemploy

```r
df_econ <- arrange(df_econ, unemploy)
ml_linear <- glm(uempmed ~ unemploy, data=df_econ)
ml_exp <- glm(uempmed ~ exp(unemploy), data = df_econ)
ml_poly2<- glm(uempmed ~ poly(unemploy,2), data=df_econ)
ml_poly3<- glm(uempmed ~ poly(unemploy,3), data=df_econ)
ml_poly10<- glm(uempmed ~ poly(unemploy,10), data=df_econ)

plot(df_econ$unemploy, df_econ$uempmed)
lines(df_econ$unemploy, fitted(ml_linear), col="red", lwd=2)
lines(df_econ$unemploy, fitted(ml_exp), col="blue", lwd=2)
lines(df_econ$unemploy, fitted(ml_poly2), col="orange", lwd=2)
lines(df_econ$unemploy, fitted(ml_poly3), col="green", lwd=2)
lines(df_econ$unemploy, fitted(ml_poly10), col="violet", lwd=2)
legend("topright", legend=c("Linear", "Exp", "Poly2", "Poly3", "Poly10"),
       col=c("red", "blue", "orange", "green", "violet"), lty=1, cex=0.8)
```



2.3 Use the cv.glm function in the boot package for the following steps. Compare the Root Mean Squared Error and Mean Squared Error.

Use cv.glm for Leave-one-out Cross Validation to compare the models above. Use cv.glm for 5-fold and 10-fold Cross Validation to compare the models above.

unemploy ~ uempmed

```r
cv_1out <- numeric(5)
cv_5 <- numeric(5)
cv_10 <- numeric(5)

cv_1out_rmse <- numeric(5)
cv_5_rmse <- numeric(5)
cv_10_rmse <- numeric(5)

mls_unue <- list(ml_linear_unue, ml_log_unue, ml_poly2_unue, ml_poly3_unue, ml_poly10_unue)

for (i in 1:5){
  cv_1out[i] <- cv.glm(df_econ, mls_unue[[i]])$delta[1]
  cv_5[i] <- cv.glm(df_econ, mls_unue[[i]], K=5)$delta[1]
  cv_10[i] <- cv.glm(df_econ, mls_unue[[i]], K=10)$delta[1]

  cv_1out_rmse[i] <- sqrt(cv.glm(df_econ, mls_unue[[i]])$delta[1])
  cv_5_rmse[i] <- sqrt(cv.glm(df_econ, mls_unue[[i]], K=5)$delta[1])
  cv_10_rmse[i] <- sqrt(cv.glm(df_econ, mls_unue[[i]], K=10)$delta[1])
}

cv_results_unue <- data.frame(
  models=c("Linear", "Log", "Poly2", "Poly3", "Poly10"),
  l1out_mse=cv_1out,
  cv5_mse=cv_5,
  cv10_mse=cv_10,
  l1out_rmse=cv_1out_rmse,
  cv5_rmse=cv_5_rmse,
  cv10_rmse=cv_10_rmse
)
cv_results_unue
```

```
##   models  l1out_mse    cv5_mse   cv10_mse l1out_rmse  cv5_rmse cv10_rmse
## 1 Linear 0.01526423 0.01528324 0.01524838  0.1235485 0.1232570 0.1236179
## 2    Log 0.02965641 0.02818408 0.02853533  0.1722104 0.1687049 0.1693112
## 3  Poly2 0.01392131 0.01385941 0.01397133  0.1179886 0.1180760 0.1181104
## 4  Poly3 0.01355233 0.01366417 0.01356343  0.1164145 0.1163349 0.1163739
## 5 Poly10 0.03203021 0.03880110 0.03156731  0.1789699 0.1657193 0.2188007
```

As was mentioned before poly10 overfits and shows one of the biggest errors, while poly3 give quite nice results with the smallest mse and rmse in all scenarios. An interesting fact is that log function has also bigger error than linear one, sometimes even bigger than poly10.

uempmed ~ unemploy

```r
cv_1out <- numeric(5)
cv_5 <- numeric(5)
cv_10 <- numeric(5)

cv_1out_rmse <- numeric(5)
cv_5_rmse <- numeric(5)
cv_10_rmse <- numeric(5)

mls_ueun <- list(ml_linear, ml_exp, ml_poly2, ml_poly3, ml_poly10)

for (i in 1:5){
  cv_1out[i] <- cv.glm(df_econ, mls_ueun[[i]])$delta[1]
  cv_5[i] <- cv.glm(df_econ, mls_ueun[[i]], K=5)$delta[1]
  cv_10[i] <- cv.glm(df_econ, mls_ueun[[i]], K=10)$delta[1]

  cv_1out_rmse[i] <- sqrt(cv.glm(df_econ, mls_ueun[[i]])$delta[1])
  cv_5_rmse[i] <- sqrt(cv.glm(df_econ, mls_ueun[[i]], K=5)$delta[1])
  cv_10_rmse[i] <- sqrt(cv.glm(df_econ, mls_ueun[[i]], K=10)$delta[1])
}

cv_results_ueun <- data.frame(
  models=c("Linear", "Exp", "Poly2", "Poly3", "Poly10"),
  l1out_mse=cv_1out,
  cv5_mse=cv_5,
  cv10_mse=cv_10,
  l1out_rmse=cv_1out_rmse,
  cv5_rmse=cv_5_rmse,
  cv10_rmse=cv_10_rmse
)
cv_results_ueun
```

```
##   models   l1out_mse     cv5_mse    cv10_mse l1out_rmse   cv5_rmse  cv10_rmse
## 1 Linear 0.009255511 0.009484103 0.009269856 0.09620557 0.09637307 0.09605786
## 2    Exp 0.007273609 0.007238908 0.007326534 0.08528546 0.08508634 0.08522526
## 3  Poly2 0.006686348 0.006764146 0.006735082 0.08177009 0.08223788 0.08149931
## 4  Poly3 0.006697077 0.006838342 0.006663312 0.08183567 0.08216188 0.08198891
## 5 Poly10 0.006301848 0.006201883 0.006358583 0.07938418 0.07914258 0.07941405
```

In this case Poly10 has the smallest error, which is a bit surprising, but the data does follow the curve like was shown in the plot above. The linear model underfits, which was mentioned above in the plot. With the scaling, exponential model fits quite well here, at least it is better than the linear one.

2.4 Explain based on the CV and graphical model fits the concepts of Underfitting, Overfitting and how to apply cross-validation to determine the appropriate model fit. Also, describe the different variants of cross validation in this context.

Underfitting and overfitting were partially explained above.

Underfitting occurs when the model has high bias and fails to represent the data well. For example, it can be observed with the linear models that fail to depict any curves in the relationship between variables.

Overfitting, on the other hand, occurs in the models with low bias and high variance. When the model adjusts to the curves of the data flow so much that it can not generalize it anymore. A clear example is the polynomial 10.

To avoid these 2 extremes, it is crucial to test the models not only on train data but also on unseen data, to see if the model captures the overall trend. It commonly is done either by dividing training data on train and validation sets or using cross validation. Cross validation has its own options: we can choose an appropriate value of folds, in which the dataset will be divided, or use simple leave-1-out method that will perform evaluation on 1 observation for each iteration. Usually the decision, which method to choose and how big the K should be, depends on the time resources, size of the dataset and power resources since CV can be quite time-consuming on big data and complicated models.