Prof. Dr. Steffen Wagner
Angewandte Statistik
Fachbereich II
Berliner Hochschule für Technik

Exercise 05: Bias-Variance Trade Off and Cross Validation

Machine Learning I – SoSe 24

This workshop covers hierarchical clustering and soft clustering. At the end of the worksheet there are a couple of written exercises for you to do at home, which should be good practice for the exam.

1 Preparations

1.1 RStudio Project

1. Open your Machine Learning 1 RStudio Project
2. Create an R Script file to perform this exercise

1.2 Required Packages

For this exercise you require the following additional R packages. Please make sure that you have installed them on your computer before coming to the workshop session for the case that Eduroam is not working.

```
# check if packages can be loaded, i.e. they are already installed
library(FNN)            # KNN regression
library(ISLR2)          # 3D visualisation
library(boot)           # Bootstrap functions
```

If you get an error at this stage, you need to install the packages.

2 Bias-Variance Trade Off

2.1 A simulation study

Download the R file `Bias_Variance_Trade_Off.R`[1] from Moodle. This contains code to investigate the trade off between the squared bias and the variance of an estimator.

- This file simulates data from true values that follow a sine wave, and investigates the squared bias and variance properties of different estimators.
  Because the true values are known we can calculate the squared bias directly.
- The first estimator is just a constant, the mean of all the observed values. This is obviously not going to fit any shape in the true function and so it has a high bias but a low variance.
- The second estimator is "just use the observed value", which corresponds to using k-nearest neighbour regression with $k{=}1$. This will on average have a low bias (fits well) but has a high variance.
- The third part considers different values of k for k-nearest neighbour regression. A plot shows how the squared bias and variance changes with k.

Work through the code. Don't worry about understanding all the fine details, but try to follow the general procedure.

[1] https://lms.bht-berlin.de/mod/resource/view.php?id=1193428

3 Crossvalidation

3.1 Model evaluation using CV

3.1.1 The `Auto` data set

In the complete Section 3.1 you will work through Lab 5.3 in James et al. This uses the dataset called `Auto` in the `ISLR2` package. The dataset was introduced in Section 2 of James. This is a similar but larger data set to `mtcars`, which you used in the *K*-means clustering workshop. First spend a few minutes investigating the data set which is covered in an earlier section in James.

- Load the data set to the workspace running `data("Auto", package = "ISLR2")`.
- Read the help page for the dataset via `?Auto`.
- Have a look at the data set structure via `str(Auto)`.
- Work through the commands in James, section 2.3.5.

(important) Side Remark: Notice that on page 50 you "attached" the data frame `Auto` using `attach(Auto)`. Be careful when using this command! In particular do not attach it a second time (e.g. when you work through the lab on page 191). When you have finished analysing this dataset you really should detach the data from the search path using

```
detach(Auto)
search()      # shows you the elements attached to the search path
```

3.1.2 Quadratic regression model

Consider a quadratic regression model with fuel consumption `mpg` as the outcome variable and `horsepower` as predictor variable.

- Before you start working through Lab 5.3 plot the data and overlay linear and quadratic regression functions using the following code:

```
# scatter plot
plot(mpg ~ horsepower, data = Auto)

# simple linear regression
lm_hp_lin <- lm(mpg ~ horsepower, data=Auto)
lm_hp_quadr <- lm(mpg ~ horsepower + I(horsepower^2), data=Auto)

# model summary
summary(lm_hp_quadr)

# Visualisation: for a simple linear model you can use abline
abline(lm_hp_lin, col = "red")

# for the quadratic model we build our own predictor function
f_q <- function(x){
  coefs <- coef(lm_hp_quadr)
  coefs[1] + coefs[2] * x + coefs[3] * x^2
}
curve(f_q, 40, 230, add = TRUE, col = "blue")
```

`f_q` is a user defined R function, which code the quadratic predictor function

$$f_q(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x + \widehat{\beta}_2 x^2.$$

The `curve` command plots this function between the values $x = 40$ and $x = 230$.

3.1.3 Lab: Cross-validation

Now work through Lab 5.3.1 to 5.3.3 in James et al.

3.2 Model selection using cross-validation

We have decided that a quadratic regression is appropriate for modelling `mpg` dependent on `horesepower`, but there are other variables in the dataset. Is it possible that some of these also have an influence on fuel consumption?

Spoiler: It is quite likely that some but not all variables have some influence.

The traditional statistical approach to choosing a good subset of variable in a regression model is to use AIC (Akaike information criterion) and BIC (Bayesian information criterion). These criteria give a statistic which penalises how many parameters are fitted in the model. The model which gives the lowest statistic corresponds to the best model according to the AIC/BIC criteria. The BIC method gives a much higher penalty to introducing new parameters. An overview of the method is given in section 3.2.2 (*Deciding on Important Variables*, p. 78) in James et al.

An alternative is using Leave-one-out-Cross-Validation (LOOCV) MSE score for model selection.

1. The function `cv.glm()` from the `boot` package will be very useful. Read the help for `cv.glm`:

   ```
   library(boot)
   ?cv.glm
   ```

   Which parameter set is required for leave-one-out CV?
   (Since `cv.glm()` expects a *generalized* linear model object as input, we use the function `glm` for modelling our linear model.)

2. We begin by defining our minimal model with `horsepower` and `horsepower^2`:

   ```
   library(boot)
   glm_01 <- glm(mpg ~ horsepower + I(horsepower^2), data = Auto)
   cv_error <- cv.glm(Auto, glmfit = glm_01)
   cv_error$delta
   ```

   What value(s) are stored in the `delta` component returned by `cv.glm`? How to find out?

3. Now add each of the other variables in turn and for each model calculate the CV error. There is a more elegant way to do this, but the process is easier to understand when fitting each variable explicitly.

   ```
   cv_error <- rep (0, 6)  # Why 6 as 2nd argument?
   cv_error[1] <- cv.glm(Auto, glm_01)$delta[1]
   glm_fit_02 <- glm(mpg ~ horsepower + I(horsepower^2) + year, data = Auto)
   cv_error[2] <- cv.glm(Auto, glm_fit_02)$delta[1]
   ```

   - Continue by fitting `weight`, `acceleration`, `displacement` and `cylinders` instead of `year`, and then plot each of the results.

     ```
     plot(cv_error, type="b")
     ```

   - Which additional variable gives the lowest LOOCV MSE?
   - We now define the *current model* stored `glm_01` to be the minimal model plus this

variable `mpg ~ horsepower + I(horsepower^2) + best_new_variable`.

4. Run a second iteration by adding each other variable in turn to the current model. Does adding a third variable noticeably decreases the LOOCV MSE?
   - Repeat this process by continuing to add variables until the LOOCV MSE increases or flatlines.
   - Output the summary table for your chosen best model.
5. For reference the AIC method suggests all the variables should be in the model:
   `mpg ~ horsepower + I(horsepower^2) + year + weight + acceleration + displacement + cylinders`
   and the BIC method suggests
   `mpg ~ horsepower + I(horsepower^2) + year + weight + acceleration`

4 Programming Cross-Validation

In Section 3 used the command `boot::cv.glm` to find the cross-validation, but it is not difficult to code cross-validation routine yourself. The idea behind this exercise is that you get a better understanding for how cross-validation works.

Look at the example in the Lecture Slides about Cross Validation starting from page 21. You will use leave-one-out cross-validation to find which degree polynomial regression fits these data the best. (Note that in general a polynomial regression is not good at estimating a true function that contains discontinuities/jumps).

Work through the following code, replacing the `???` parts. R code reminders that are useful in this exercise:

- `vecx[7]` returns the 7th element of the vector `vecx`
- `vecx[-7]` removes the 7th element.

1. Generate the underlying outcome function and simulate the observed data

```
set.seed(1234567890)
x <- matrix(1:30, ncol = 1)
y_true <- cut(x, breaks = c(0, 10, 20, 30), labels = c(10, 15, 8)) |>
  as.character() |> as.integer()
epsilon_train <- rnorm(length(x))
y_obs <- y_true + epsilon_train
```

2. Visualise the data

```
plot(x[,1], y_true, type = "p", ylim = range(c(y_obs, y_true)),
     xlab = "x", ylab = "y")
lines(x[,1]-0.5, y_true, type = "s", lty = 2, col = "grey")
points(x[,1], y_obs, lty = 2, col = "red", pch = 20)
legend("topright", legend = c("true values", "observed data"),
       col = c("black", "red"), pch = c(1, 20))
```

3. First run: Start fitting linear regression and leaving just the first observation out

```
# create new data by leaving one observation out
x_loo <- x[-???]
y_loo <- y_obs[-???]

# fit the linear regression model with the loo data
lm_loo <- lm(y_loo ~ x_loo)

# plot the resulting regression to existing plot
abline(lm_loo, col = "grey")

# find the sum of squared error using the missing point
predict(lm_loo, newdata = data.frame(x_loo = x[???])) - y_obs[???])^2
```

4. Now adapt the code so that each observation is left out in turn by looping over the 30 observations:

```r
# initialization of result vector
loo_sse <- NULL

# loop over each of the n observations
for(i in ???){

  #create new data by leaving one observation out
  x_loo <- x[???]
  y_loo <- y_obs[???]

  ## fit the linear regression model with the loo data
  lm_loo <- lm(y_loo ~ x_loo)

  #plot the resulting regression
  abline(loo.fit,col="grey")

  # store current sse to result vector using the missing point
  loo_sse <- c(loo_sse, ???)
}

# calculate mean of calculated sse
mse_loocv <- mean(???)
mse_loocv
```

Note: The diagram shows all 30 different LOOCV linear regression lines.

5. Now fit a polynomial degree k with k from 1 up to 10. The command `poly(x, degree = k, raw = T)` allows us to efficiently specify a polynomial degree k in a regression formula.

```r
# Initialisation of result vector
mse_loocv <- rep(NA, 10)

# outer loop changing the polynomial degree
for(k in ???){

  # inner loop over each of the n observations
    for(i in ???){

      #create new data by leaving one observation out
      x_loo <- x[???]
      y_loo <- y_obs[???]

      ## fit the linear regression model with the loo data
      lm_loo <- lm(y_loo ~ poly(x_loo, degree = ???, raw = T))

      # store current sse to result vector using the missing point
      loo_sse <- c(loo_sse, ???)
    }
```

```
  # store mse fur current polynomial degree in mse_loo
  mse_loocv[k] <-
}
plot(mse_loocv, type="b")
```

Which degree minimizes the mean squared error?
Use this result to obtain the final fitted values using all the data and plot the results.

```
lm_best_mse <- lm(y_obs ~ poly(x, degree = ???))
plot(x, lm_best_mse$fitted.values, col="red", type = "l")
points(x, y_obs)
lines(x, y_true)
```