107.330 Statistische Simulation und computerintensive Methoden

Cross-validation

Alexandra Posekany

WS 2020

Cross-validation vs bootstrap

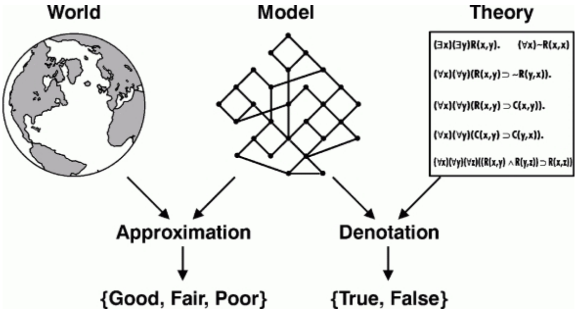
The most popular resampling methods are the bootstrap and cross-validation. The two are mainly separated by their purpose of utilization:

- The **Bootstrap** Method is usually used to measure the **accuracy of parameter estimates**, i.e. constructing distributions and deriving confidence intervals, p-values, standard errors of measurements etc. .
- **Cross-validation** is typically applied in the context of **model assessment** or **model selection**.

Problem of model fitting

Essentially, all models are wrong,
but some are useful.

[George Box]



World
Model
Theory
$(\exists x)(\exists y)R(x,y).$
$(\forall x){\sim}R(x,x)$
$(\forall x)(\forall y)(R(x,y) \supset {\sim}R(y,x)).$
$(\forall x)(\forall y)(R(x,y) \supset C(x,y)).$
$(\forall x)(\forall y)(C(x,y) \supset C(y,x)).$
$(\forall x)(\forall y)(\forall z)((R(x,y) \wedge R(y,z)) \supset R(x,z))$
Approximation
Denotation
{Good, Fair, Poor}
{True, False}

Notion of “Within-sample estimation”

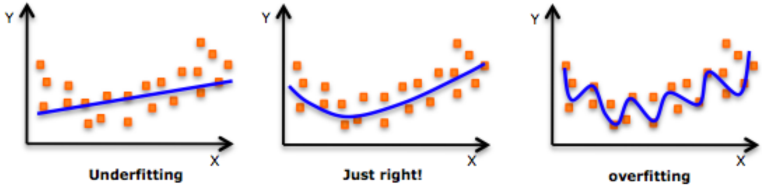
The “problem” of model validation is that based on a model fitted on a finite sample we would like to know how our model performs in general for data coming from the same population.

Hence, we would like to know some error rate for general data from the population based on our model while in practice we can compute only the error rate for the data on which the model was build. This type of approach is referred to as **within-sample validation**, as the sample itself is utilized for fitting the model and estimating the errors.

This is like trusting the cat to keep the cream, as the data for which the model was built to fit is expected to fit more or less well, unless a really inappropriate approach was chosen. The risk here is that using only the same sample for validation will lead to **overfitting**.

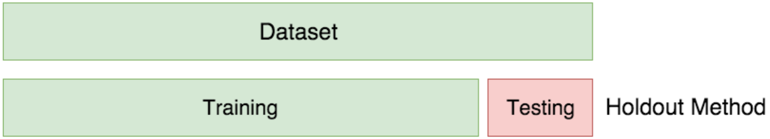
Notion of Overfitting

- **Underfitting**: The chosen model is too simple in its structure to properly fit the data, such as fitting polynomial data with a linear model
- **Overfitting**: the model is too complex to be generalised for new data of the same source, as it was trimmed to best fit the sample data, such as interpolating all points with a high-order polynomial which will lead to basically no error of the fit within the sample.



Notion of “Holding out”

Of course, we would consider our validation as more objective, if it used different data for the validation than the one which was originally used for fitting the model. Therefore, the first step is to (randomly) split the data into two different subsets, the **training data** which is utilized for the model fitting, and the **test data** which is used for testing model performance.



Validation set approach

The most basic holding out method is the validation set approach. It has the steps:

1. Randomly split the data into a training and test set, typically here the data is split in half.

   Watch out that here the basic structure of the data must be contained in the training data set, as otherwise the model will not the be representative of the whole data! E.g. dichotomous variables must have observations of both groups represented.

2. Use the training set for model building.
3. Apply the models of interest to the test data and use some performance measure to evaluate each methods prediction performance.
4. Choose the model with the best prediction performance.

Problems with the Validation set approach

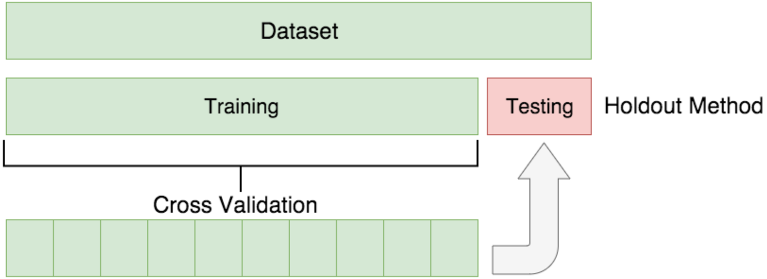
The validation set approach has the following points of critique:

1. Performance might highly depend on the specific used training and testing data division.
2. A smaller set than actually available was used for fitting purposes. As statistical methods tend to perform better with increasing sample size the fitted model likely performs worse than anticipated.

Getting rid of the issues of validation sets

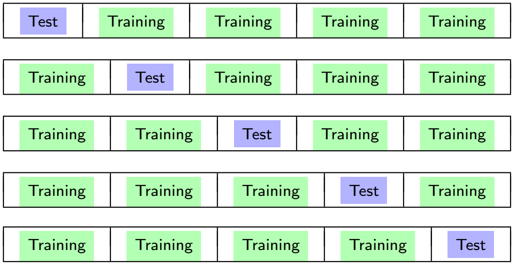
An idea to take care of the first issue is to use methods we already learned about: bootstrapping, jack-knifing. By repeatedly choosing different samples we will get rid of this problem.

An idea take care of the second issue is to use a much large portion than half of the data in each "fitting round".



Leave-one-out cross-validation

Leave-one-out cross-validation (LOOCV) is the first approach to address the issues mentioned above of the validation set approach. In LOOCV the training data set has always size $n - 1$ and only one observation is kept out for testing purpose.



In the following assume a regression type context for a sample of size n where $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$ are the explanatory and response variables, respectively. We denote $(\mathbf{x}_{-i}, y_{-i})$ as the training data and kept the single oberservation $(\mathbf{x}_i, y_i)$ as the test data.

Leave-one-out cross-validation - Error estimation

The prediction error when measured using the mean squared error (MSE) is then

$$MSE_i = (y_i - \hat{y}_i)^2.$$

Clearly MSE_i is an unbiased estimate for the prediction error - but quite a poor one. Hence the procedure is repeated n times to obtain

$$MSE_1, \ldots, MSE_n$$

the different MSEs of each observation compared to the model fitted by the rest of the data.

The LOOCV estimate for the general test MSE is then the average of all MSEs.

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i.$$

Advantages of LOOCV over validation set approach

LOOCV has several advantages over the validation set approach which specifically address the two issues mentioned above:

1. It is reproducible as no random splitting is involved. We do not depend on a single random selection of training or test data. All n different selections of training and test data are predefined.
2. As repeatedly n-1 observations are used in the model estimation it has usually less bias than the validation set approach. For large enough n the performance of each cross validation sample is indiscernible from the one of the whole data.

Disadvantages of LOOCV

Along with the advantages come the disadvantages.

One major problem of LOOCV is that it is potentially **expensive to compute**, especially if n is large and if each individual model is slow to fit.

1. We always have to fit as many models as we have data points and
2. each model is almost of the same complexity to fit as the original one.

Hence, while it can be used to many statistical methods if it is feasible depends a lot on the data set and methods under considerations.

LOOCV and OLS

For some methods however getting $CV_{(n)}$ is surprisingly cheap.

For example for OLS the following relationship holds:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2,$$

where h_i is the leverage of the ith observation. As the leverage measures the influence of every single data point on the whole model fit and can be obtained in a matrix notation from the OLS estimation procedure, this results is obtained almost for free.

Usually however such formulas do not exist and the models have to be fitted n times.

k-fold cross-validation

An alternative to leave-one-out cross-validation is **k-fold cross-validation ($CV_{(k)}$)**.

In $CV_{(k)}$ the data is randomly divided into k approximately equal sized subsets (folds). Then for $CV_{(k)}$ we use the k-th fold as test data while the remaining $k-1$ folds are the training data. Hence there will be k separate models fit, each with a mean squared errors $MSE_1, \dots, MSE_k$.

From those, we obtain

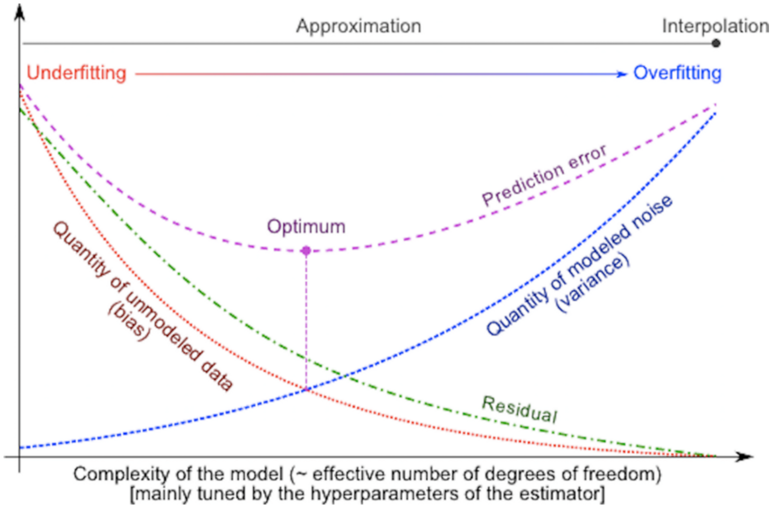
$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} MSE_i.$$

LOOCV is a special case of $CV_{(k)}$ with k=n.

Practical considerations

- LOOCV is better when one is interested in an unbiased estimate of the test MSE, as similarly to jack-knifing it can evaluate the bias cmoning from single observations.
- $CV_{(k)}$ is biased, but has a lower variance. This is due to the fact that in $CV_{(n)}$ the model fits are all highly dependent, whereas for $CV_{(k)}$ there is less overlap and therefore less correlation between different MSEs of different training sets.
- When performing k-fold CV the computational burden depends heavily on the value k which specifies how many models have to be fitted.

Variance-Bias Trade-Off



Practical considerations

When not the estimation of the testing MSE is of main focus, but the comparison of competing models, only the relative performances matter and one is less concerned about bias in the estimation of the MSE.

From extended experimental studies, it was concluded that $k = 5$ or $k = 10$ are best in the sense of avoiding excessive bias while keeping the variance from getting too high. This refers to the optimum of the two in the graphical representation above.

CV for classification

CV is a useful tool for many practical problems with iid data. And MSE is not the only possible choice as performance measure. The median absolute daviation is a more robust measure with respect to outlying values.

In addition for a whole set of problems, such as classification, measures such a accuracy, missclassification, sensitivity and specificity measure the rates of correctly or incorrectly classified observations.

Measures for evaluating Prediction Performance

Metric Outcome	Mean Squared Error	$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$
	Root Mean Squared Error	$\mathrm{RMSD} = \sqrt{\mathrm{MSE}(\hat{\theta})}$ $\mathrm{RMSD} = \sqrt{\mathrm{E}((\hat{\theta}-\theta)^2)}$ $\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}$
	Median absolute Deviation	$\mathrm{MAD} = \mathrm{median}(\lvert X_i - median(X)\rvert)$

Classification

Categorial Dichotomous Outcome	Sensitivity True Positive Rate	$\frac{\sum \text{True Positive}}{\sum \text{Condition Positive}}$
	Specificity True Negative Rate	$\frac{\sum \text{True Negative}}{\sum \text{Condition Negative}}$
Classification	Accuracy	$\frac{\#\text{correctly classifed units}}{\#\text{all units}}$
	Missclassification Rate	$\frac{\#\text{incorrectly classifed units}}{\#\text{all units}}$

Monte Carlo Variants of CV

- An alternative idea to LOOCV for dealing with the issues of the validation set approach would be to combine it with Monte Carlo simulation. In this **Monte Carlo validation set approach**, the validation set is randomly drawn repeated many times and the overall error measure is calculated as the average over all such error measures on the different test and training sets.
- Combining this idea with $CV_{(k)}$ leads to **Monte Carlo corss validation**. $CV_{(k)}$ is repeated many times over different versions of splitting the data into folds. As in all Monte Carlo, we then average over all these $CV_{(k)}$ error measures.

CV example I

Consider we generate data following the polynomial model:

$$y_i = 5 + 0.1 \cdot x_i + 0.004 \cdot x_i^2 - 0.0003 \cdot x_i^3 + \varepsilon_i, i = 1, \dots, n$$

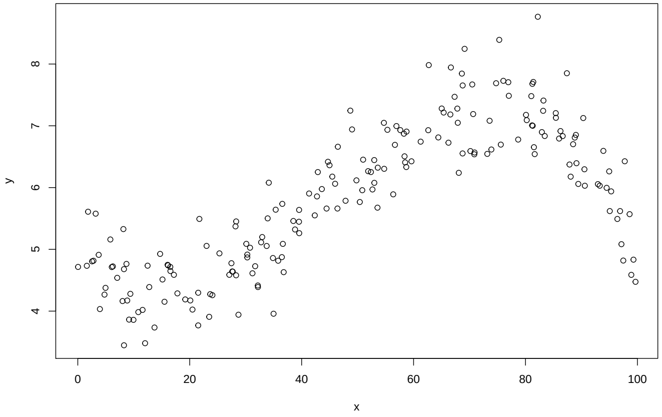
with residual errors $\varepsilon_i \sim N(0, 0.25)$.

Consider a sample of size n=200.

```
> set.seed(123)
> n <- 200
> eps <- rnorm(n, 0, 0.5)
> x <- sort(runif(n, 0, 100))
> X <- cbind(1, poly(x, degree = 3, raw = TRUE))
> y <- as.numeric(X %*% c(5,-0.1,0.004,-0.00003) + eps)
> DATA <- data.frame(x = x, y = y)
```

Visulaisation of the data

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



Fitting Cubic Splines

The problem at hand is then to fit a cubic spline model and to choose the appropriate degrees of freedom of the spline estimation function. We will consider the values ranging from 1 to 20.

As a brief revision, a natural cubic spline has the form $f(u) = a_0 + u^1 \cdot a_1 + u^2 \cdot a_2 + u^3 \cdot a_3$ at all control points u and fullfills the following 4 side conditions required for estimating the coefficients a_0, a_1, a_2, a_3 with all intervals:

- position of start point:
  $p_0 = f(0) = a_0 + 0^1 \cdot a_1 + 0^2 \cdot a_2 + 0^3 \cdot a_3$
- first derivative of the starting point:
  $p_1 = f'(0) = a_1 + 2 \cdot 0^1 \cdot a_2 + 3 \cdot 0^2 \cdot a_3$
- second derivative of the starting point:
  $p_2 = f''(0) = 2 \cdot a_2 + 6 \cdot 0^1 \cdot a_3$
- position of end point:
  $p_3 = f(1) = a_0 + 1^1 \cdot a_1 + 1^2 \cdot a_2 + 1^3 \cdot a_3$

Cubic Splines in R

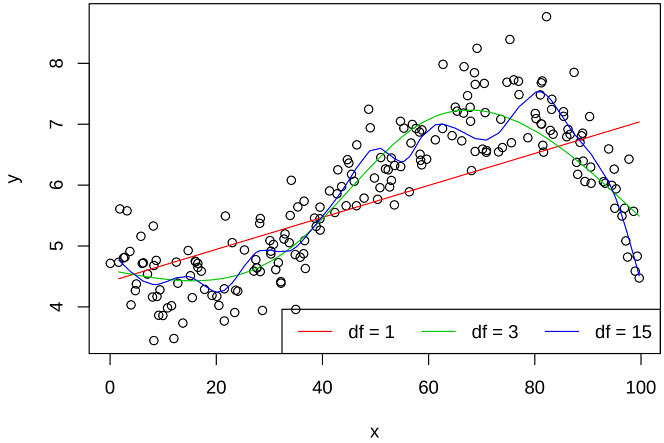
The degrees of freedom allow the algorithm to choose df-1 knots which are situated at suitably chosen quantiles of x. If an intercept is fitted as well, only df-2 knots are available for spline estimation.

df=1 corresponds to not knots which means that a straight line rather than a polynomial is fitted.

```
> library(splines)
> dfs <- 1:20
> set.seed(12334)
> ind <- sort(sample(nrow(DATA), ceiling(nrow(DATA)/2)))
> TRAIN <- DATA[ind,]
> TEST <- DATA[-ind,]
> FITS <- apply(t(dfs), 2, function(df) lm(y ~ ns(x, df=df), data=TRAIN))
```

Visualisation of Cubic splines

```
> par(mar=c(4,4,0,0)+0.1); plot(x,y);
> legend("bottomright", paste("df =",c(1,3,15)), col=2:4, lty=1, ncol=3)
> lines(TRAIN$x, fitted(FITS[[1]]), col=2)
> lines(TRAIN$x, fitted(FITS[[3]]), col=3)
> lines(TRAIN$x, fitted(FITS[[15]]), col=4)
```



Calculation of MSE

```
> PREDS <- lapply(FITS, predict, TEST)
> MSE <- function(yhat, y) mean((yhat-y)^2)
> MSES <- lapply(PREDS, MSE, y=TEST$y)
> unlist(MSES)
 [1] 0.6569235 0.5944813 0.2644427 0.2556883 0.2437177 0.2619221 0.2871260
 [8] 0.2751665 0.2715856 0.2806758 0.2893099 0.2761064 0.2739354 0.2934503
[15] 0.3082299 0.2899985 0.3090916 0.2990688 0.3114684 0.2934960
> which.min(unlist(MSES))
[1] 5
```

We would therefore conclude from this validation set that df=5 is the optimal value for fitting the splines.

Validation set - influence of the seed

We repeat the same procedure with another seed.

```
> set.seed(44)
> ind2 <- sort(sample(nrow(DATA), ceiling(nrow(DATA)/2)))
> TRAIN2 <- DATA[ind2,]
> TEST2 <- DATA[-ind2,]
> FITS2 <- apply(t(dfs), 2, function(df) lm(y ~ ns(x, df=df), data=TRAIN2))
> PREDS2 <- lapply(FITS2, predict, TEST2)
> MSES2 <- lapply(PREDS2, MSE, y=TEST2$y)
> which.min(unlist(MSES2))
[1] 10
```

Now we would conclude that df=10 is the optimal value for fitting the splines.

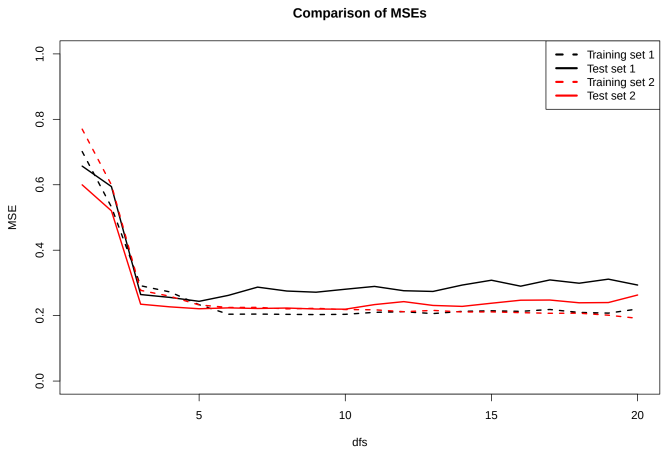
However, this is contradictory to this first validation sets result.

Comparing the 2 validation sets

```
> FITTED <- lapply(FITS, fitted)
> FITTED2 <- lapply(FITS2, fitted)
> MSEtrain <- lapply(FITTED2, MSE, y=TRAIN$y)
> MSEtrain2 <- lapply(FITTED2, MSE, y=TRAIN2$y)
```

```
> plot(dfs, unlist(MSEtrain), lty = 2, lwd = 2, type = "l",
+     ylim = c(0, 1), ylab = "MSE", main = "Comparison of MSEs")
> lines(dfs, unlist(MSES), lty = 1, lwd = 2)
> lines(dfs, unlist(MSEtrain2), lty = 2, lwd = 2, col = 2)
> lines(dfs, unlist(MSES2), lty = 1, lwd = 2, col = 2)
> legend("topright", legend = c("Training set 1", "Test set 1",
+     "Training set 2", "Test set 2"), col = c(1, 1,
+     2, 2), lwd = 3, lty = c(2, 1, 2, 1))
```

Comparing the 2 validation sets



Auxiliary Calculation of MSE for all dfs

```
> MSEind <- function(ind, dfs=1:20, DATA=DATA)
+   {
+   TRAIN <- DATA[-ind,]
+   TEST <- DATA[ind,]
+   FITS <- apply(t(dfs), 2,
+        function(df) lm(y ~ ns(x, df=df), data=TRAIN))
+   PREDS <- lapply(FITS, predict, TEST)
+   MSE <- function(yhat, y) mean((yhat-y)^2)
+   unlist(lapply(PREDS, MSE, y=TEST$y))
+   }
```

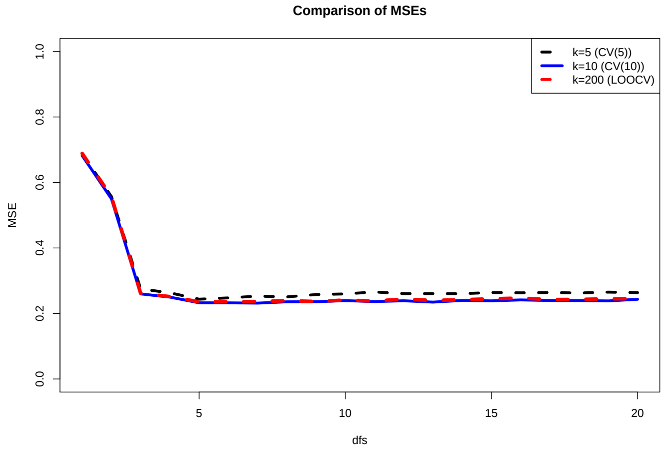
k-fold Cross Validation

```
> kCV <- function(k=5, DATA=DATA, dfs=1:20)
+   {
+   N <- nrow(DATA)
+   SIZES <- floor(N/k)
+   GROUPS <- rep(1:k, c(rep(SIZES, k-1),N-(k-1)*SIZES))
+   inds <- sample(GROUPS)
+   INDs <- split(1:N, inds)
+   MSEs <- lapply(INDs, MSEind, DATA=DATA, dfs=dfs)
+   MSEs <- do.call(rbind,MSEs)
+   colMeans(MSEs)
+   }
```

Comparing CV approach with different k

```
> set.seed(9876)
> CV5 <- kCV(k=5, DATA=DATA)
> CV10 <- kCV(k=10, DATA=DATA)
> CV200 <- kCV(k=200, DATA=DATA)
>
> results<-c(which.min(CV5),which.min(CV10),which.min(CV20
> names(results)<-c("k=5","k=10","k=200")
> results
  k=5  k=10 k=200
    5     7     5
```

Comparing the 2 validation sets



Task 1

We will work with the dataset Auto in the ISLR package. Obtain information on the data set, its structure and the real world meaning of its variables from the help page.

1. Fit the following models

```
mpg ~ horsepower
mpg ~ poly(horsepower,2)
mpg ~ poly(horsepower,3)
```

Visualise all 3 models in comparison added to a scatterplot of the input data.

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.
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.

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

Task 2

Load the data set ‘economics’ from the package ‘ggplot2’.

1. 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

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

Compare the Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation.

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.