Cross Validation

S.Tonini, F. Chiaromonte (special thanks to J. Di Iorio and L. Insolia)

February 21th 2024

Contents

Introduction **1**
Libraries . . . 1
Data . . . 1

Cross validation **1**
LOOCV . . . 1
k-fold CV . . . 4

Using caret for CV **7**
Data . . . 7

Temporal Block "Cross" Validation **12**

Introduction

Libraries

We are going to use:

- **tidyverse**: *Easily Install and Load the 'Tidyverse'*
- **caret**: *Classification and Regression Training*
- **ggplot2**: *Create Elegant Data Visualisations Using the Grammar of Graphics*
- **dslabs**: *Data Science lab*

```
library(tidyverse)  # data manipulation and visualization
library(caret)      # statistical learning techniques
library(ggplot2)    # plots
library(dslabs)     # digit recognition data
library(datasets)
```

Data

We will use the **polls_2008** dataset, which is part of the **dslabs** package. It contains data from 131 pollsters for the popular vote between Obama and McCain in the 2008 presidential election. The variables are:

- **day.** Days until election day. Negative numbers are reported so that days can increase up to 0, which is election day.
- **margin.** Average difference between Obama and McCain for that day

Cross validation

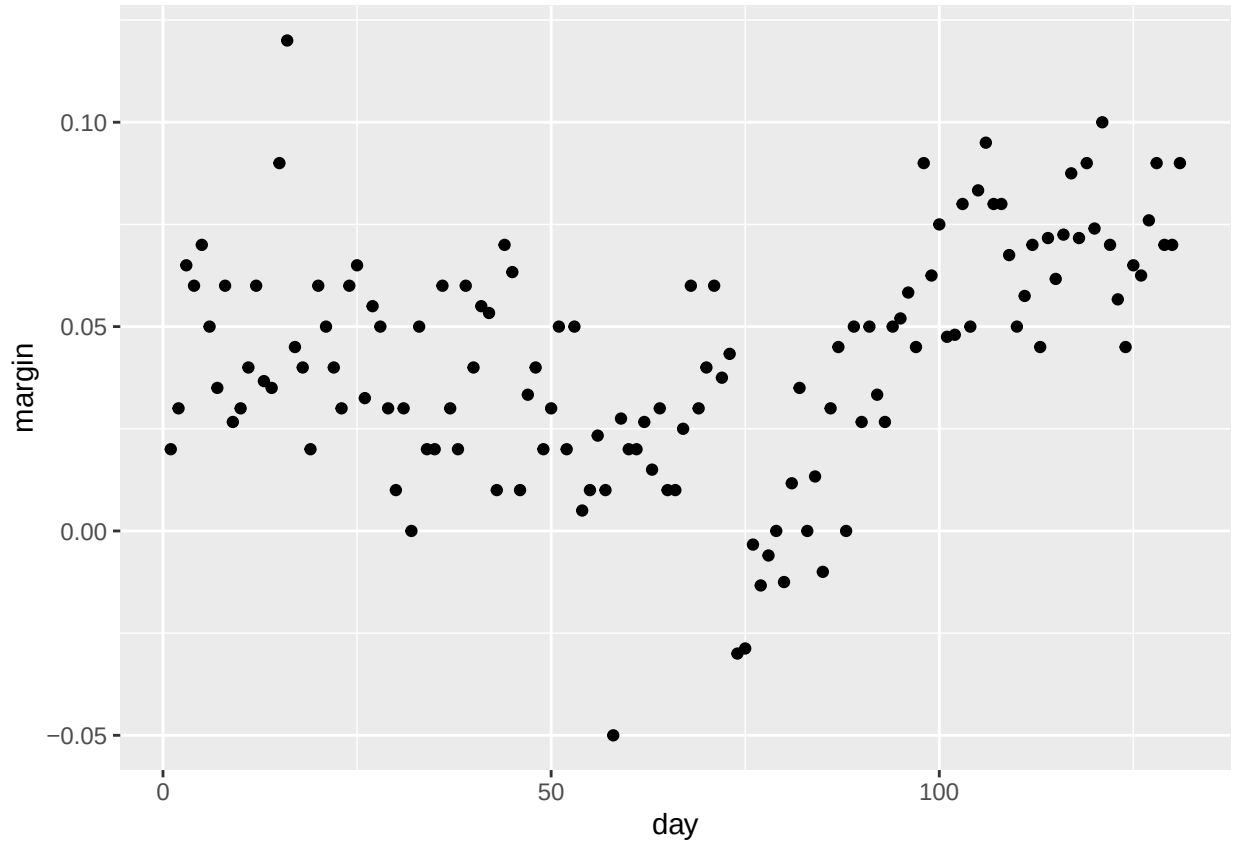
We are going to perform CV by hand. Precisely we are going to perform:

1. Leave-one-out cross validation (LOOCV)
2. k-folds cross validation

LOOCV

In this approach, we reserve only one data point from the available dataset, and train the model on the rest of the data. This process iterates for each data point.

```
data("polls_2008")
polls_2008 <- data.frame(polls_2008)
polls_2008$day <- 1:nrow(polls_2008)
qplot(day, margin, data = polls_2008)
```

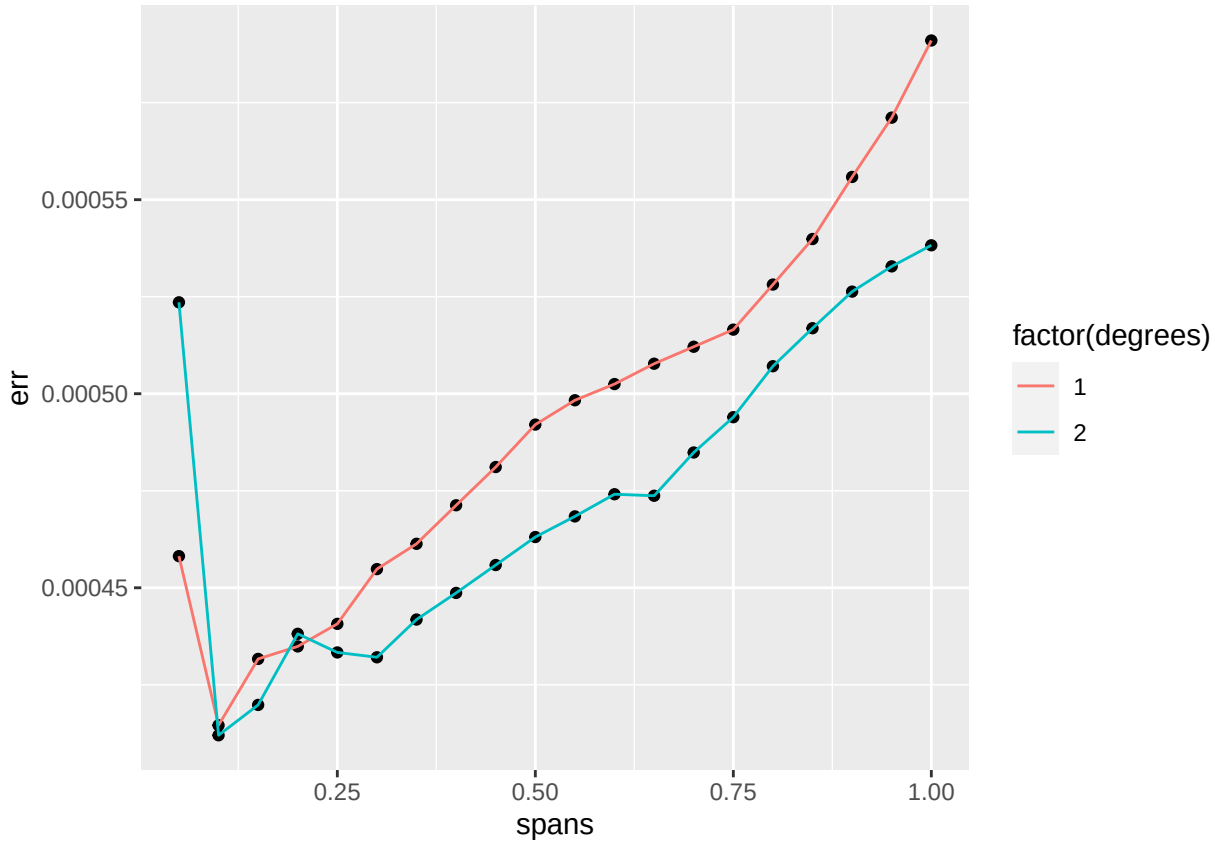



```
degree_list <- list()
span_values <- seq(0.05,1,length=20)
for(deg in 1:2){ #polynomial degree
  err <- list()
  for(k in 1:length(span_values)){ #smoothness
    score <- list()
    for(i in 1:(nrow(polls_2008))){
      training = polls_2008[-i,]
      model = loess(margin ~ day, data = training, span = span_values[k],
                    degree=deg)
```

```
      validation = polls_2008[i,]
      pred = na.omit(predict(model, validation))
      # error of ith fold
      score[[i]] = (validation$margin - pred)^2
    }
  # returns a vector with the average error for a given degree & span
  err[[k]] <- mean(unlist(score),na.rm=TRUE)
  }
  degree_list[[deg]] <- err
}

# prepare dataframe for ggplot
spans <- rep(span_values,2)
degrees <- rep(c(1,2), each = length(span_values))
err <- unlist(degree_list)
df_toplot <- as.data.frame(cbind(spans,degrees,err))

# plot
plt <- ggplot(df_toplot, aes(x=spans, y=err, group=factor(degrees))) +
  geom_point() + geom_line(aes(col=factor(degrees)))
plt
```



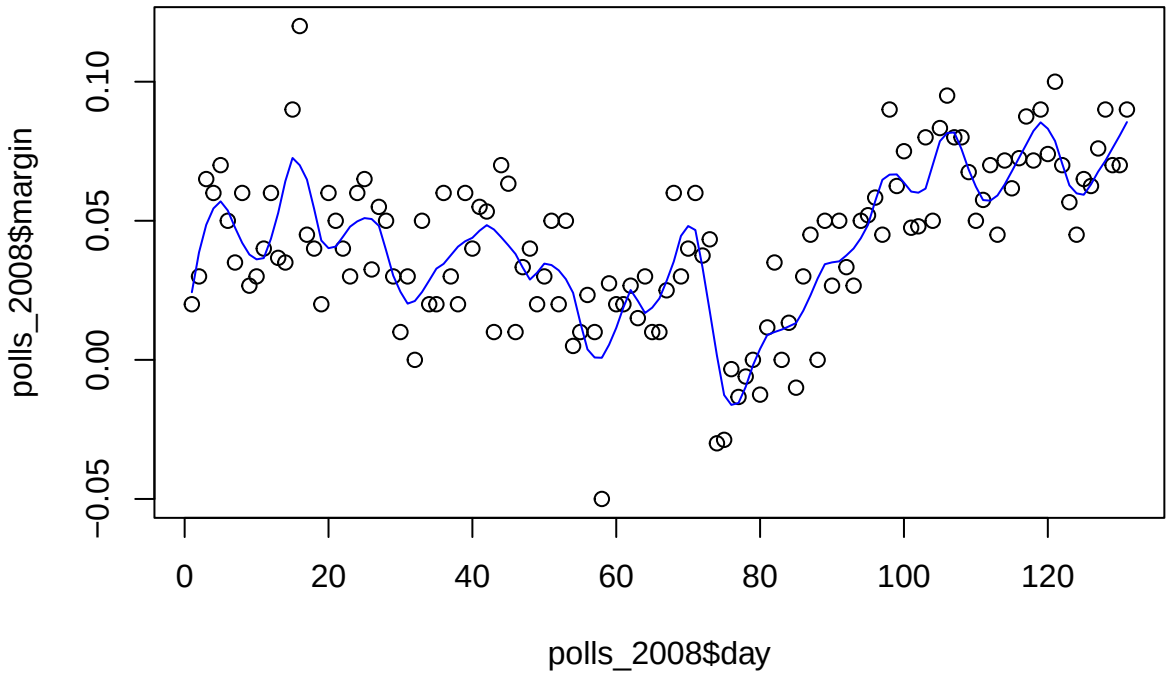
Let's find the parameters corresponding to the minimum error.

```
best <- df_toplot[which(df_toplot$err==min(df_toplot$err)),]
best
```

```
##    spans degrees         err
## 22   0.1       2 0.000411992
```

Let's plot the resulting smoothed curve.

```
res <- loess(margin ~ day, data = polls_2008, span =best$spans, degree=best$degrees)
plot(polls_2008$day, polls_2008$margin)
lines(predict(res), col='blue')
```



k-fold CV

Let's validate the parameter using the k-fold cross validation.

These are the steps we need to implement:

1. Randomly split your entire dataset into k folds;
2. Iterate across each kth fold, which serves as a testing set, and train your model only on the remaining k˘1 folds;
3. Test model accuracy/effectiveness on the kth fold, and record the "error" you see on each of the k predictions;
4. Repeat this until each of the k-folds has served as the test set;
5. The average of your k recorded errors is called the **cross validation error** and will serve as a performance metric for the model.

Create the folds:

```r
set.seed(123)
flds <- caret::createFolds(1:nrow(polls_2008),
                           k = 3, list = TRUE,
                           returnTrain = FALSE)
flds
```

```
## $Fold1
##  [1]   5   6  10  14  17  21  26  28  29  31  32  34  36  39  41  51  52  53  56
## [20]  57  60  65  67  69  73  78  80  81  85  86  87  91  93 100 101 102 103 111
## [39] 116 117 121 127 129 131
##
## $Fold2
##  [1]   2   3   4   8   9  13  15  19  22  24  30  37  42  43  44  49  50  54  58
## [20]  61  62  64  74  75  76  84  88  89  90  94  95  98  99 104 105 106 107 108
## [39] 110 113 123 125 126
##
## $Fold3
##  [1]   1   7  11  12  16  18  20  23  25  27  33  35  38  40  45  46  47  48  55
## [20]  59  63  66  68  70  71  72  77  79  82  83  92  96  97 109 112 114 115 118
## [39] 119 120 122 124 128 130
```

```r
class(flds)
```

```
## [1] "list"
```

```r
# yopu can use [[k]] or [k] to access the k-th element
flds[1]     # you need to "unlist" afterwards (see below)
```

```
## $Fold1
##  [1]   5   6  10  14  17  21  26  28  29  31  32  34  36  39  41  51  52  53  56
## [20]  57  60  65  67  69  73  78  80  81  85  86  87  91  93 100 101 102 103 111
## [39] 116 117 121 127 129 131
```

```r
flds[[1]]   # you do not
```

```
##  [1]   5   6  10  14  17  21  26  28  29  31  32  34  36  39  41  51  52  53  56
## [20]  57  60  65  67  69  73  78  80  81  85  86  87  91  93 100 101 102 103 111
## [39] 116 117 121 127 129 131
```

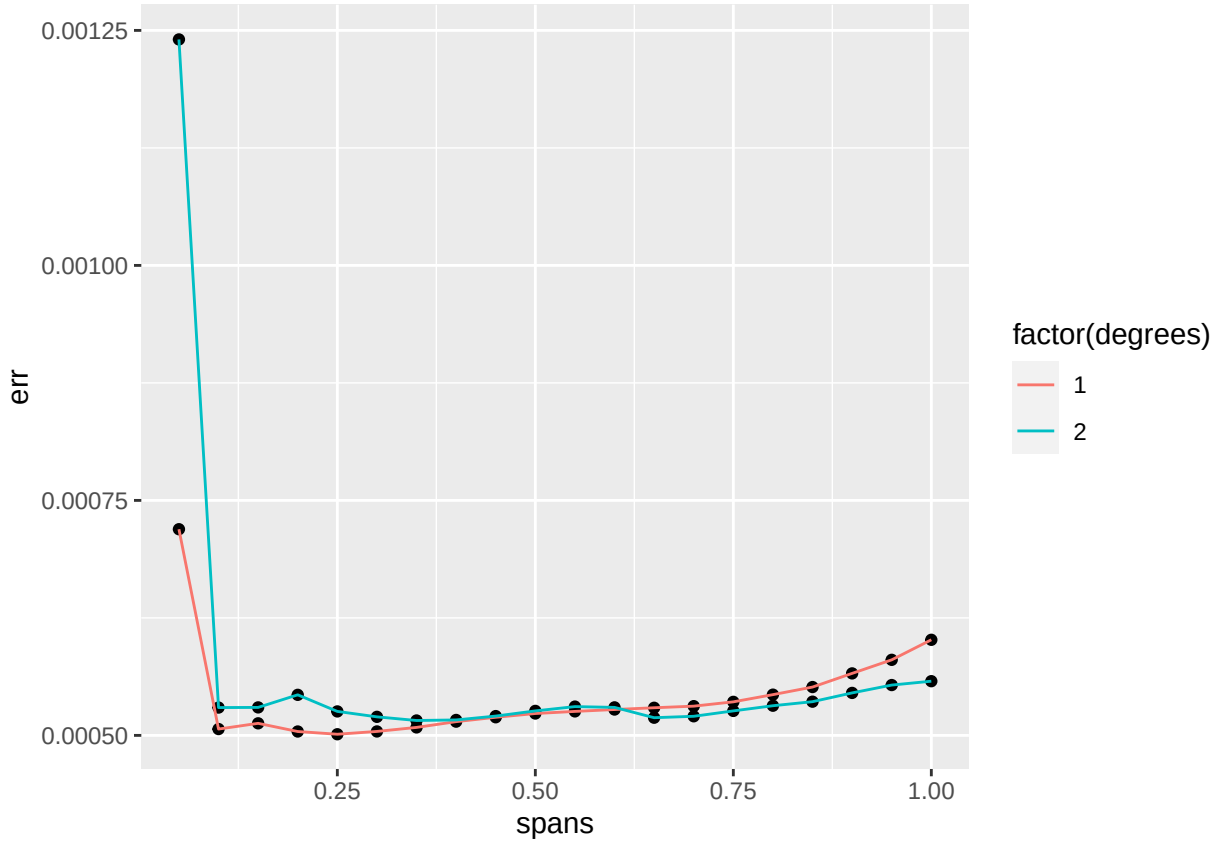
Perform an iteration similarly to the one for LOOCV:

```r
degree_list <- list()
for(deg in 1:2){ #polynomial degree
  err <- list()
  for(k in 1:length(span_values)){ #smoothness
    score <- list()
    for(i in 1:length(flds)){
      validation <- polls_2008[unlist(flds[i]),]
      training <- polls_2008[unlist(flds[-i]),]
      model = loess(margin ~ day, data = training, span = span_values[k],
                    degree=deg)
      pred = na.omit(predict(model, validation))
      # error of ith fold
      score[[i]] = (validation$margin - pred)^2
    }
    err[[k]] <- mean(unlist(score))
  }
```

```
  degree_list[[deg]] <- unlist(err)
}


spans <- rep(span_values,2)
degrees <- rep(c(1,2), each = length(span_values))
err <- unlist(degree_list)
df_toplot <- as.data.frame(cbind(spans,degrees,err))


plt <- ggplot(df_toplot, aes(x=spans, y=err, group=factor(degrees))) +
  geom_point() + geom_line(aes(col=factor(degrees)))
plt
```



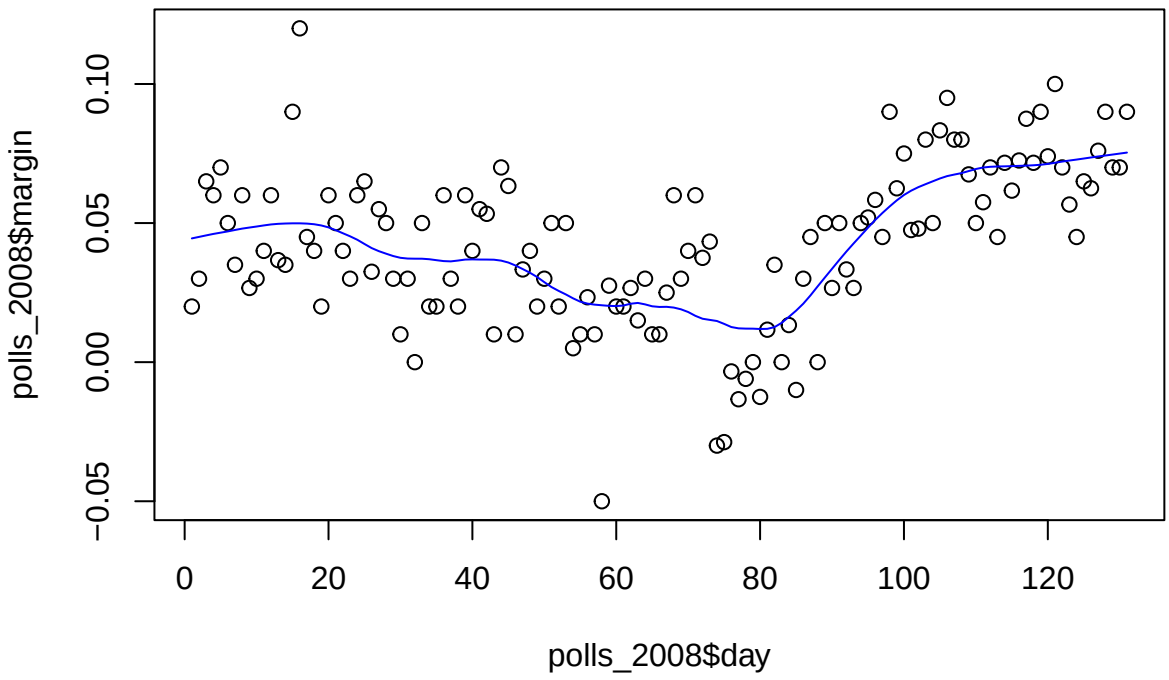
Let us find the parameters corresponding to the minimum error.

```
df_toplot[which(df_toplot$err==min(df_toplot$err)),]
```

```
##   spans degrees          err
## 5  0.25       1 0.0005012194
```

Let us plot the resulting regression line.

```
best <- df_toplot[which(df_toplot$err==min(df_toplot$err)),]
res <- loess(margin ~ day, data = polls_2008, span =best$spans, degree=best$degrees)
plot(polls_2008$day, polls_2008$margin)
lines(predict(res), col='blue')
```



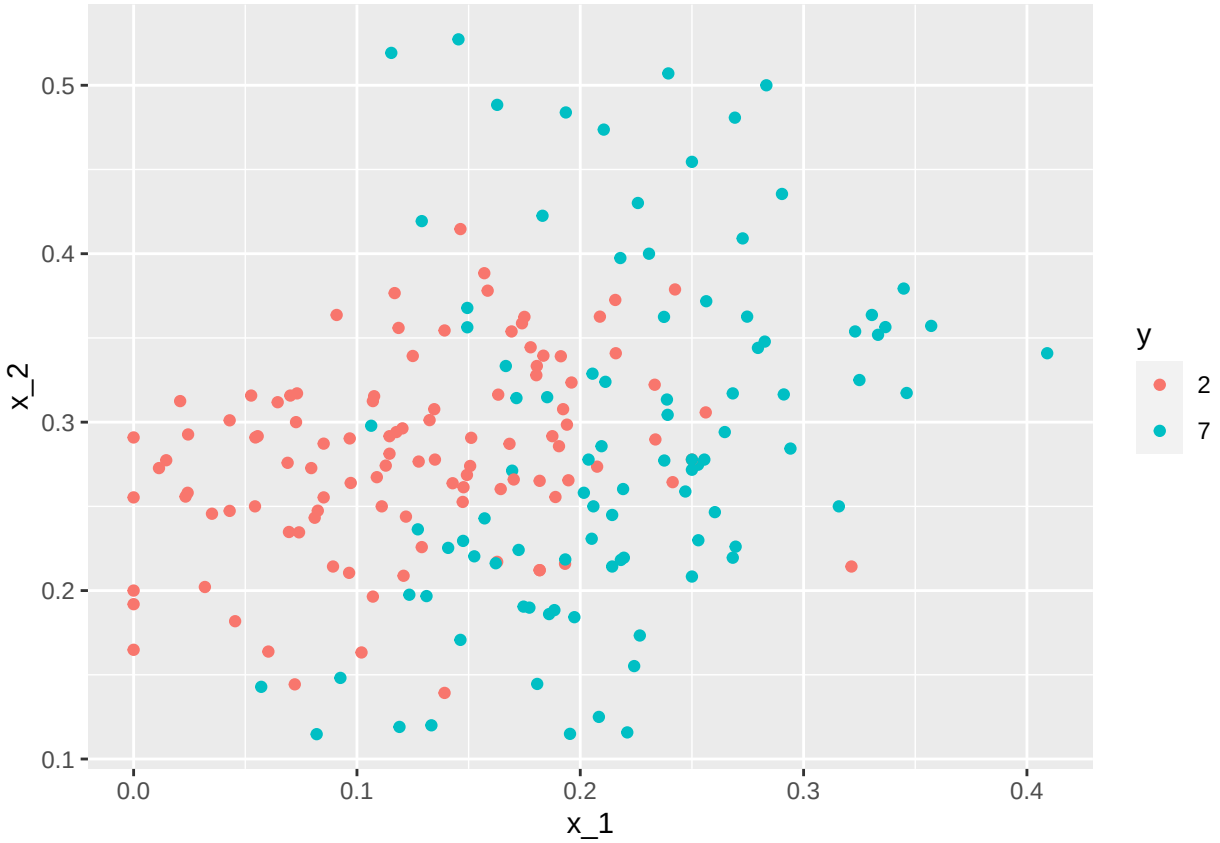
Using caret for CV

Data

We will use a dataset already presented in a previous lesson (the **economics** in **ggplot2** package) as well as the `mnist_27` dataset which is part of the **dslabs** package.

The **mnist** dataset is a very famous and large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. In the `mnist_27` data set, we only include a randomly selected set of 2's and 7's (the label/class that we try to predict) along with the two predictors based on the proportion of dark pixels in the upper left and lower right quadrants respectively. The dataset is divided into training and test sets.

```
data("mnist_27")
mnist_27$test %>% ggplot(aes(x_1, x_2, color = y)) + geom_point()
```



Cross validation is also implemented in the train function of the caret package. Here caret train function allows one to train different algorithms using the same syntax.

So, for example, we can type:

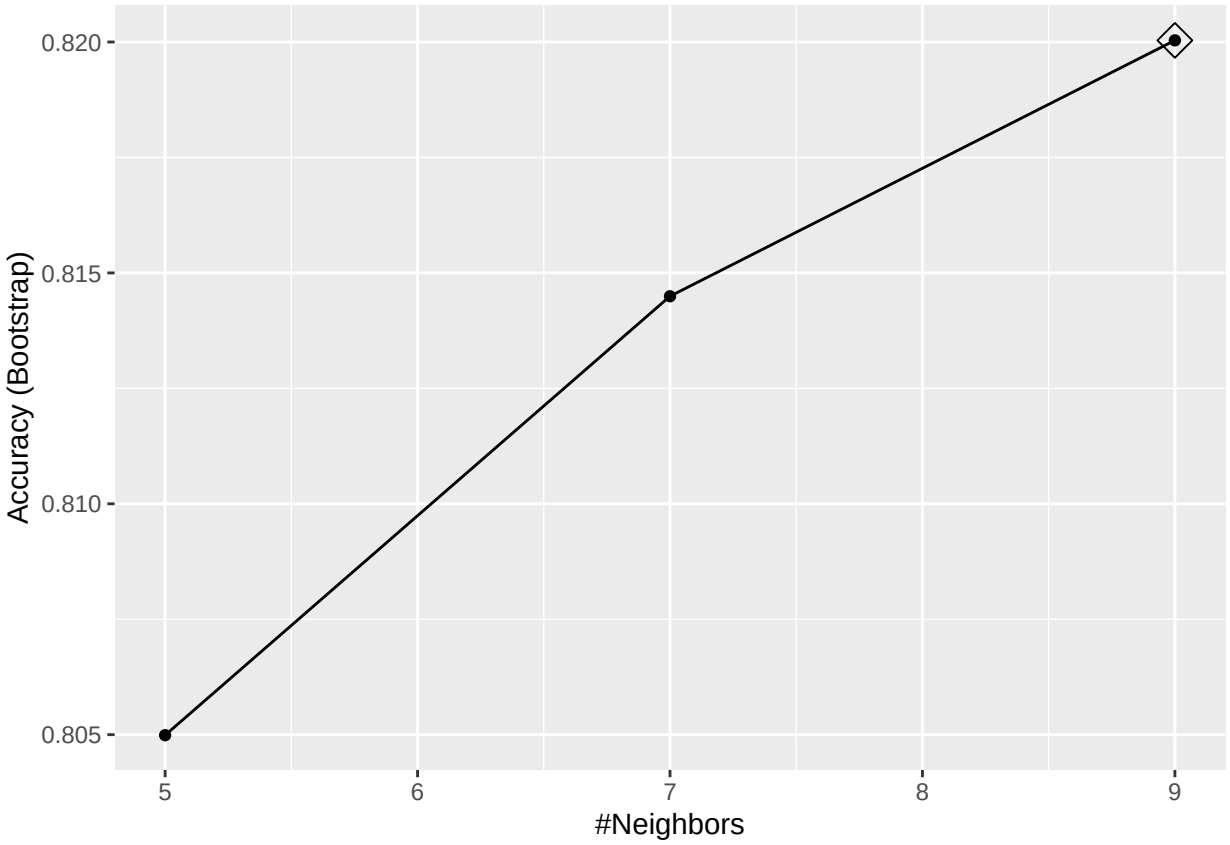
```
train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
confusionMatrix(y_hat_knn, mnist_27$test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  2  7
##          2 91 17
##          7 15 77
##
##                Accuracy : 0.84
##                  95% CI : (0.7817, 0.8879)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6785
##
##  Mcnemar's Test P-Value : 0.8597
##
##             Sensitivity : 0.8585
##             Specificity : 0.8191
```

```
##          Pos Pred Value : 0.8426
##          Neg Pred Value : 0.8370
##              Prevalence : 0.5300
##          Detection Rate : 0.4550
##    Detection Prevalence : 0.5400
##       Balanced Accuracy : 0.8388
##
##        'Positive' Class : 2
##
```

In the presence of a a tuning parameter, it automatically uses cross validation to decide among a few default values. You can quickly see the results of the cross validation using the **ggplot** function. The argument **highlight** highlights the max:

```
ggplot(train_knn, highlight = TRUE)
```



By default, caret used bootstrap resampling with 25 repetitions – this is the default resampling approach in caret. The process caret used was:

- Randomly sample the data with replacement. This means that a single observation could be chosen more than once. The total size of the modeling dataset will be the same as the original dataset but some observations will not be included and some will be included more than once.
- Develop a model based on the randomly sampled observations only.
- Use the withheld observations (also known as "out-of-bag" (OOB) observations) to compute the Accuracy based on the predictions.
- Repeat the process (25 models are run) and average the Accuracy values.

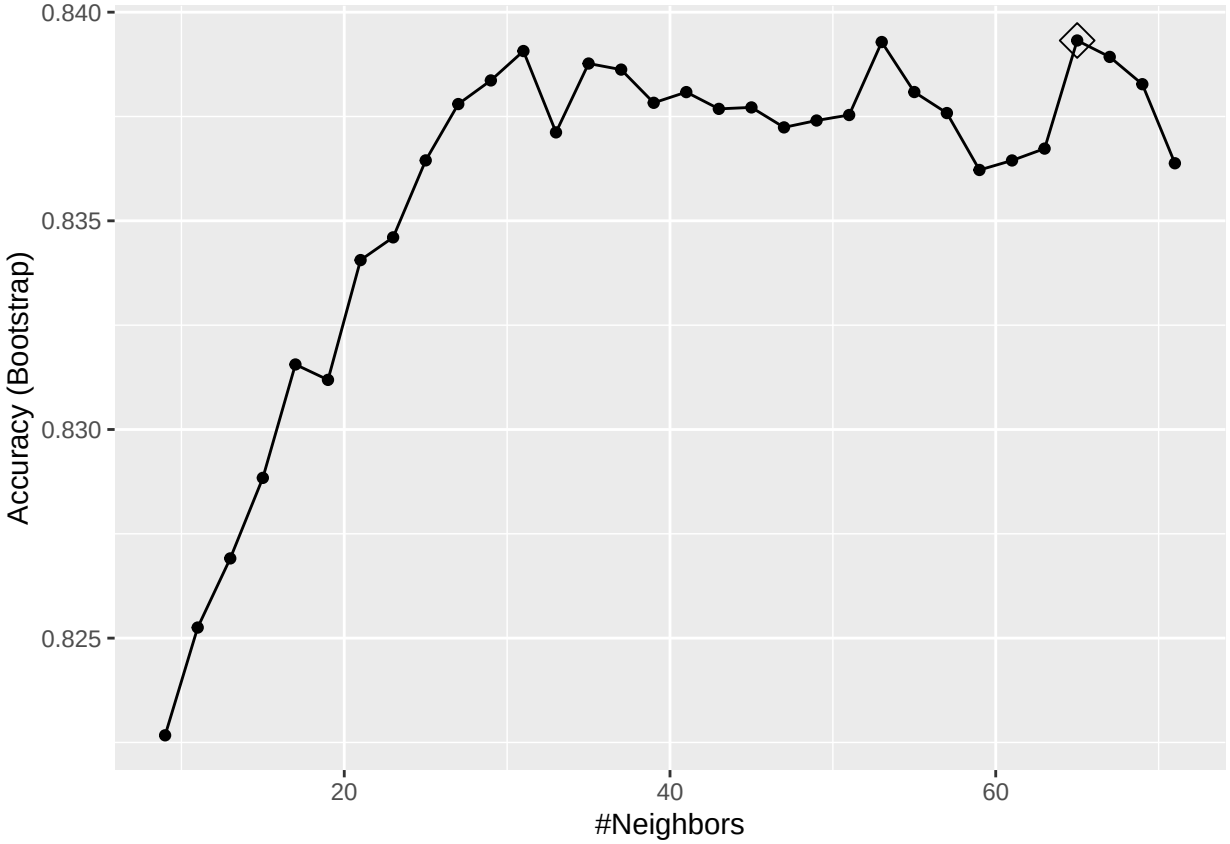
- When the resampling is done, caret then runs a final model on the full dataset and stores this in finalModel. So, in this example, 25 + 1 models are run, 25 bootstraps and one model run using full dataset.

Note: the Accuracy is the percentage of correctly classified OOB observations.

For the kNN method, the default is to try $k = (5, 7, 9)$.

We can change this using the **tuneGrid** parameter.

```
set.seed(2024)
train_knn <- train(y ~ ., method = "knn",
                   data = mnist_27$train,
                   tuneGrid = data.frame(k = seq(9, 71, 2)))
ggplot(train_knn, highlight = TRUE)
```



The best k shown in the plot and the corresponding training set outcome distribution is accessible as follows:

```
train_knn$bestTune
```

```
##    k
## 29 65
```

```
train_knn$finalModel
```

```
## 65-nearest neighbor model
## Training set outcome distribution:
##
##   2   7
## 379 421
```

The overall accuracy on the training set is:

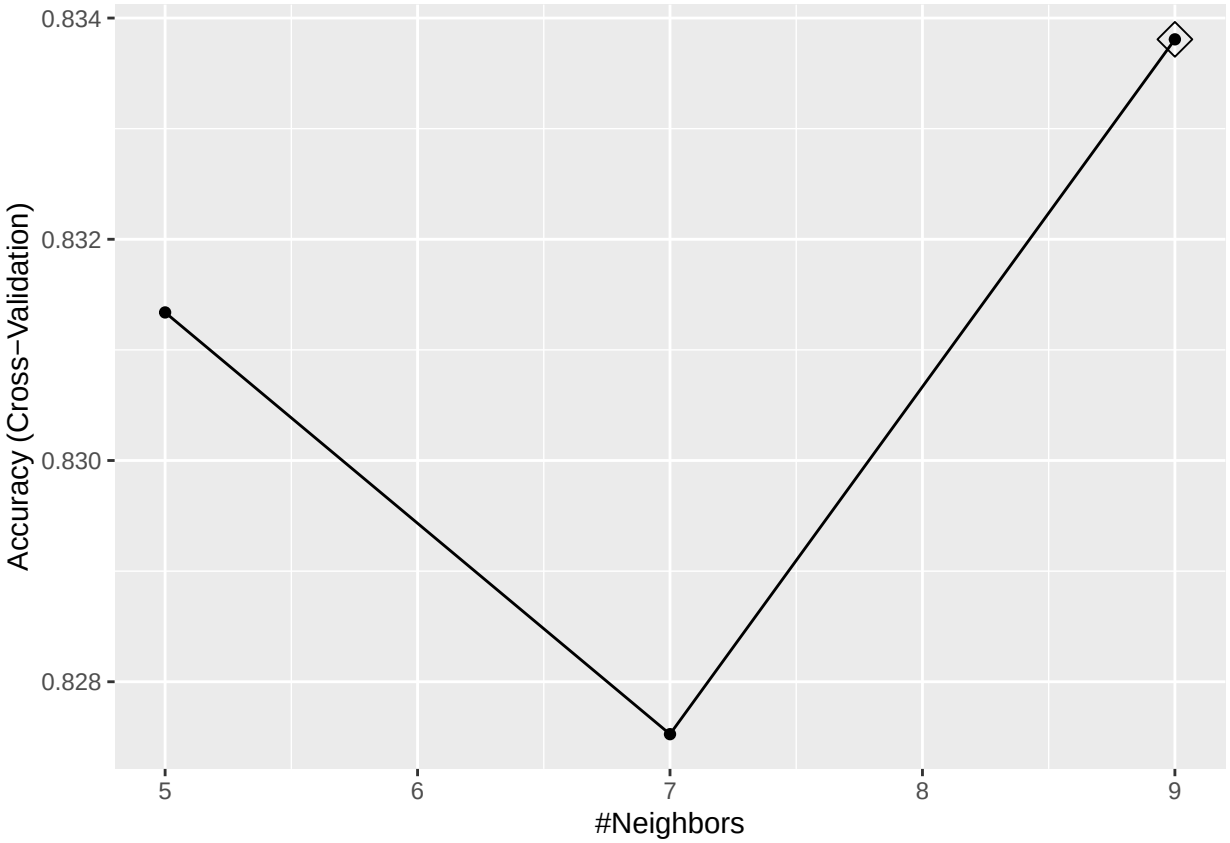
```
confusionMatrix(predict(train_knn, mnist_27$test, type = "raw"),mnist_27$test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  2  7
##          2 91 16
##          7 15 78
##
##                Accuracy : 0.845
##                  95% CI : (0.7873, 0.8922)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6887
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8585
##             Specificity : 0.8298
##          Pos Pred Value : 0.8505
##          Neg Pred Value : 0.8387
##              Prevalence : 0.5300
##          Detection Rate : 0.4550
##    Detection Prevalence : 0.5350
##       Balanced Accuracy : 0.8441
##
##        'Positive' Class : 2
##
```

we can use k-folds cross validation instead of bagging by setting

```
set.seed(2024)
tc <- trainControl(method = "cv", number = 10)

train_kCV <- train(y ~ ., method = "knn",
                   data = mnist_27$train,
                   trControl = tc)
ggplot(train_kCV, highlight = TRUE)
```

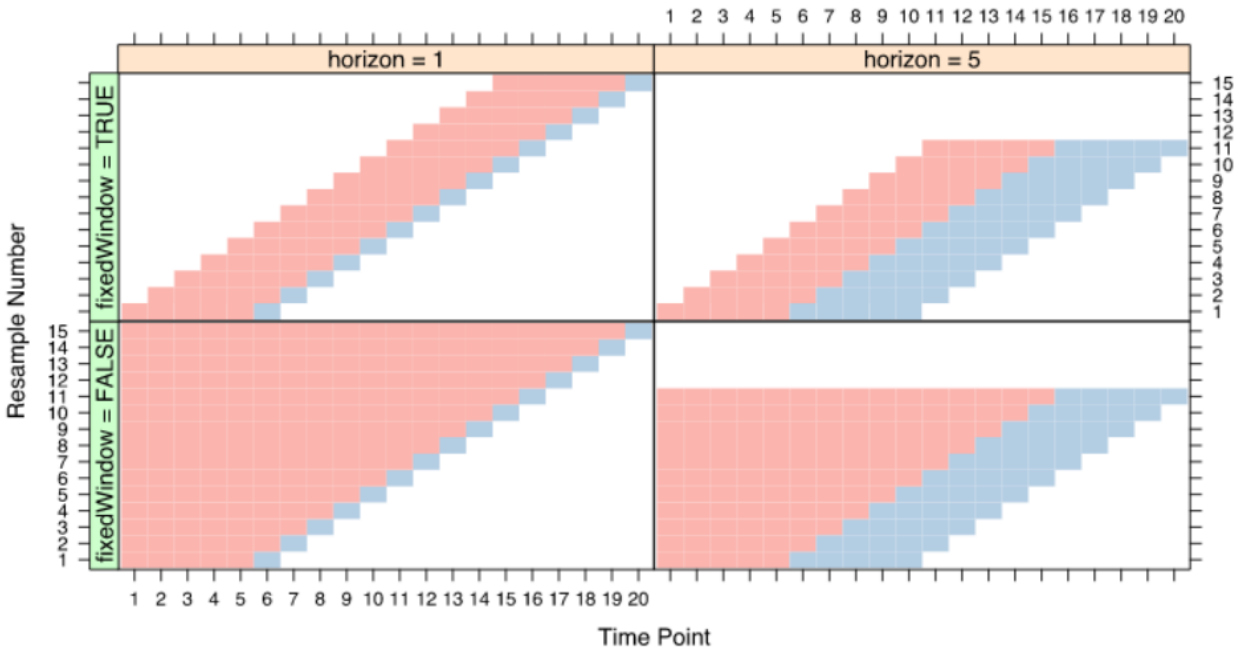


Temporal Block "Cross" Validation

Simple random sampling is probably not the best way to resample time series data. **caret** contains a function called **createTimeSlices** that can create the indices for this type of splitting.

The function takes as input a vector and three parameters:

- y: a vector of outcomes. These should be in chronological order
- $initialWindow$: the initial number of consecutive values in each training set sample
- $horizon$: The number of consecutive values in test set sample
- $fixedWindow$: A logical: if FALSE, the training set always start at the first sample and the training set size will vary over data splits.



Example 1: one training set and one test set

```
p <- 0.75

createTimeSlices(y = 1:nrow(polls_2008),
                 initialWindow = round(p*nrow(polls_2008),0),
                 horizon = (nrow(polls_2008)-round(p*nrow(polls_2008),0)),
                 fixedWindow = TRUE)           # in this example fixedWindow doesn't matter!!
```

```
## $train
## $train$Training98
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
## [76] 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
##
##
## $test
## $test$Testing98
##  [1]  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
## [20] 118 119 120 121 122 123 124 125 126 127 128 129 130 131
```

Example 2: rolling window

```
createTimeSlices(y = 1:nrow(polls_2008),
                 initialWindow = round(p*nrow(polls_2008),0),
                 horizon = ((nrow(polls_2008)-round(p*nrow(polls_2008),0))-2),
                 fixedWindow = TRUE)
```

```
## $train
## $train$Training098
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
## [76] 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
```

```
##
## $train$Training099
##  [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
## [26] 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
## [51] 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
## [76] 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
##
## $train$Training100
##  [1]   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21
## [20]  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
## [39]  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
## [58]  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
## [77]  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
## [96]  98  99 100
##
##
## $test
## $test$Testing098
##  [1]  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
## [20] 118 119 120 121 122 123 124 125 126 127 128 129
##
## $test$Testing099
##  [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
## [20] 119 120 121 122 123 124 125 126 127 128 129 130
##
## $test$Testing100
##  [1] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [20] 120 121 122 123 124 125 126 127 128 129 130 131
```

Example 3: recursive window

```
createTimeSlices(y = 1:nrow(polls_2008),
                 initialWindow = round(p*nrow(polls_2008),0),
                 horizon = ((nrow(polls_2008)-round(p*nrow(polls_2008),0))-2),
                 fixedWindow = FALSE)
```

```
## $train
## $train$Training098
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
## [76] 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
##
## $train$Training099
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
## [76] 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
##
## $train$Training100
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
```

```
##  [91]  91  92  93  94  95  96  97  98  99 100
##
##
## $test
## $test$Testing098
##  [1]  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
## [20] 118 119 120 121 122 123 124 125 126 127 128 129
##
## $test$Testing099
##  [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
## [20] 119 120 121 122 123 124 125 126 127 128 129 130
##
## $test$Testing100
##  [1] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [20] 120 121 122 123 124 125 126 127 128 129 130 131
```

Tip for Mod 2!!!

The function **trainControl** (in **caret**) controls the computational setup of the **train** function.

The created object can be placed in the **trControl** argument.

```
#trainControl(method = "timeslice",
#              initialWindow = 90,
#              horizon = 25,
#              fixedWindow = FALSE)
```

We run a rolling window in the case of $h = 1$ and a rolling window of size 100

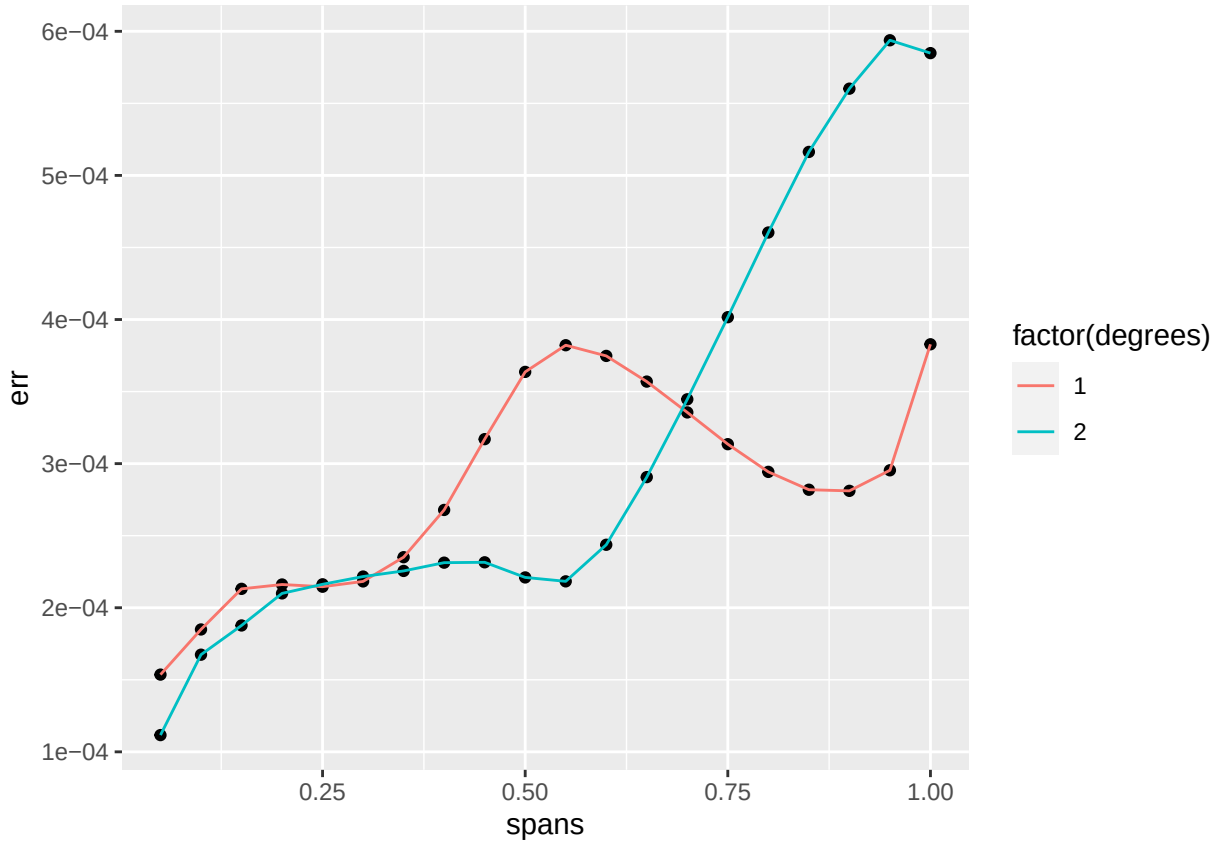
```
flds <- createTimeSlices(y = 1:nrow(polls_2008),
                 initialWindow = 100,
                 horizon = 1,
                 fixedWindow = TRUE)
```

```
degree_list <- list()
for(deg in 1:2){ #polynomial degree
  err <- list()
  for(k in 1:length(span_values)){ #smoothness
    score <- list()
    for(i in 1:length(flds$test)){
      validation <- polls_2008[unlist(flds$test[i]),]
      training <- polls_2008[unlist(flds$train[-i]),]
      model = loess(margin ~ day, data = training, span = span_values[k],
                    degree=deg)
      pred = na.omit(predict(model, validation))
      # error of ith fold
      score[[i]] = (validation$margin - pred)^2
    }
    err[[k]] <- mean(unlist(score))
  }
  degree_list[[deg]] <- unlist(err)
}


spans <- rep(span_values,2)
degrees <- rep(c(1,2), each = length(span_values))
err <- unlist(degree_list)
```

```
df_toplot <- as.data.frame(cbind(spans,degrees,err))


plt <- ggplot(df_toplot, aes(x=spans, y=err, group=factor(degrees))) +
  geom_point() + geom_line(aes(col=factor(degrees)))
plt
```



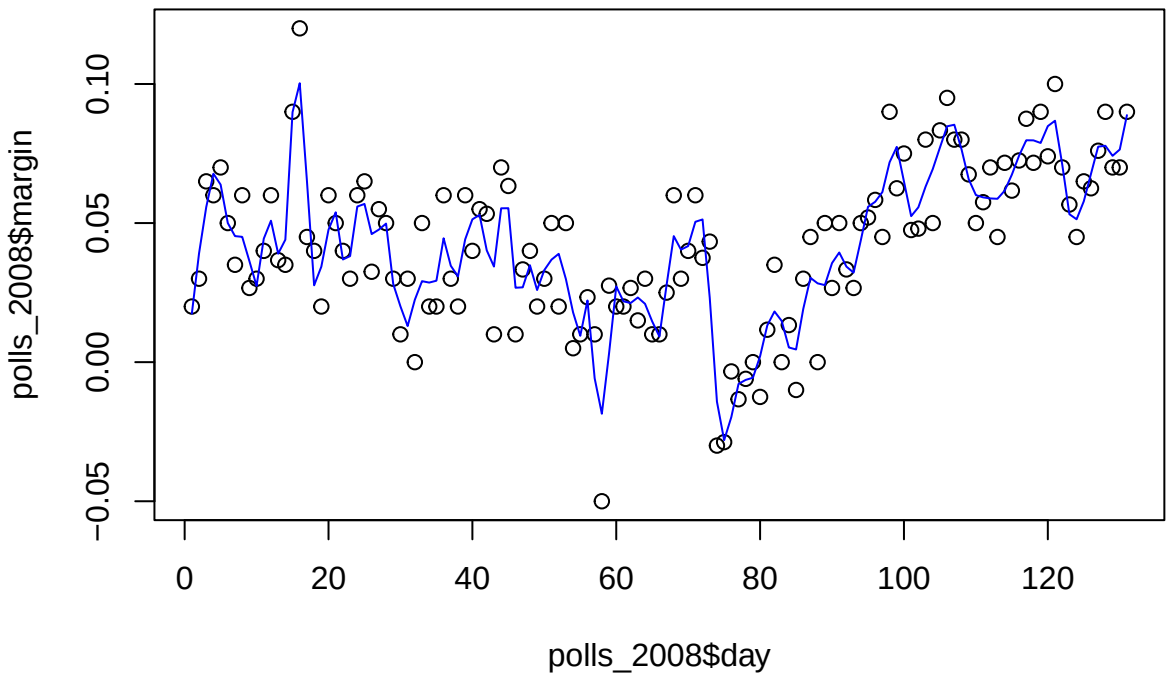
Let us find the parameters corresponding to the minimum error.

```
df_toplot[which(df_toplot$err==min(df_toplot$err)),]
```

```
##    spans degrees          err
## 21  0.05       2 0.0001116394
```

Let us plot the resulting regression line.

```
best <- df_toplot[which(df_toplot$err==min(df_toplot$err)),]
res <- loess(margin ~ day, data = polls_2008, span =best$spans, degree=best$degrees)
plot(polls_2008$day, polls_2008$margin)
lines(predict(res), col='blue')
```


polls_2008$margin
−0.05
0.00
0.05
0.10
0
20
40
60
80
100
120
polls_2008$day