Cross Validation

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

February 28th 2023

Contents

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*

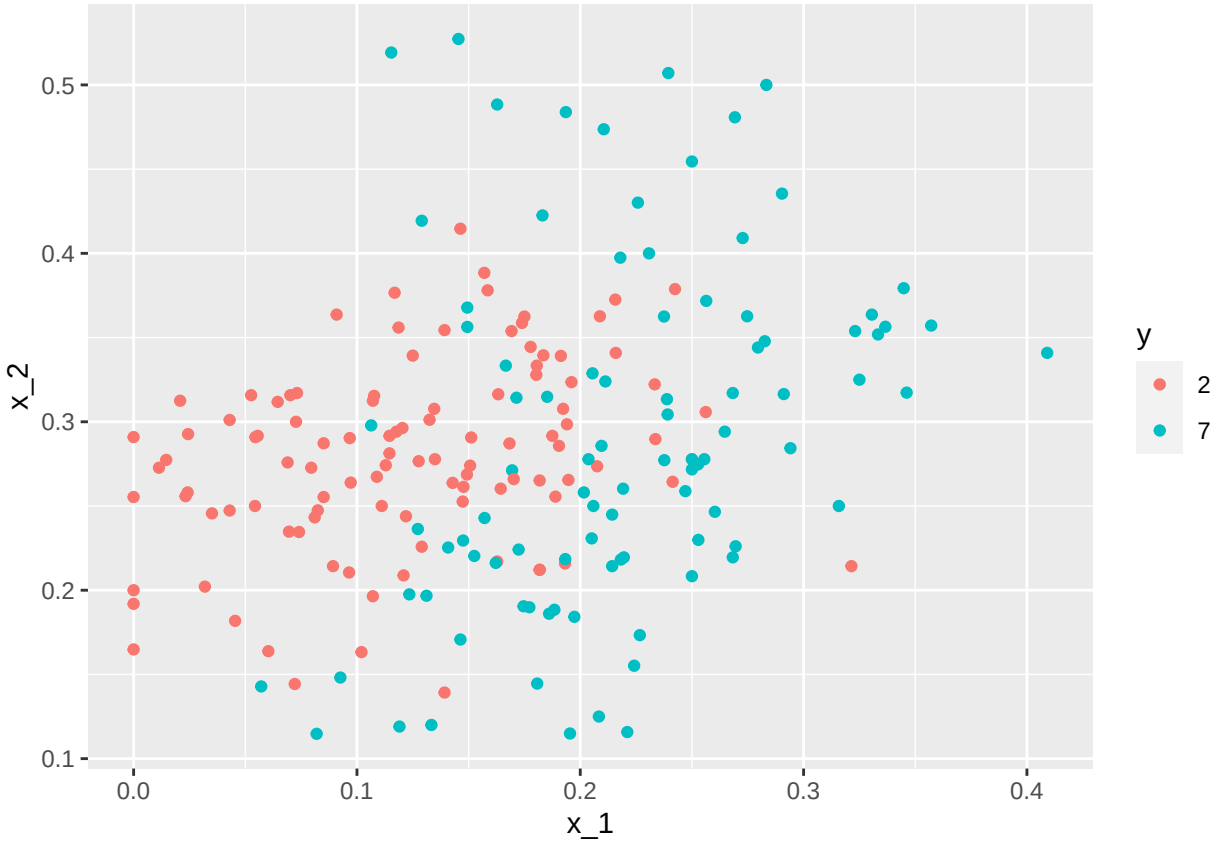
```
library(tidyverse)  # data manipulation and visualization
library(caret)      # statistical learning techniques
library(ggplot2)    # plots
library(dslabs)     # digit recognition data
```

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()
```



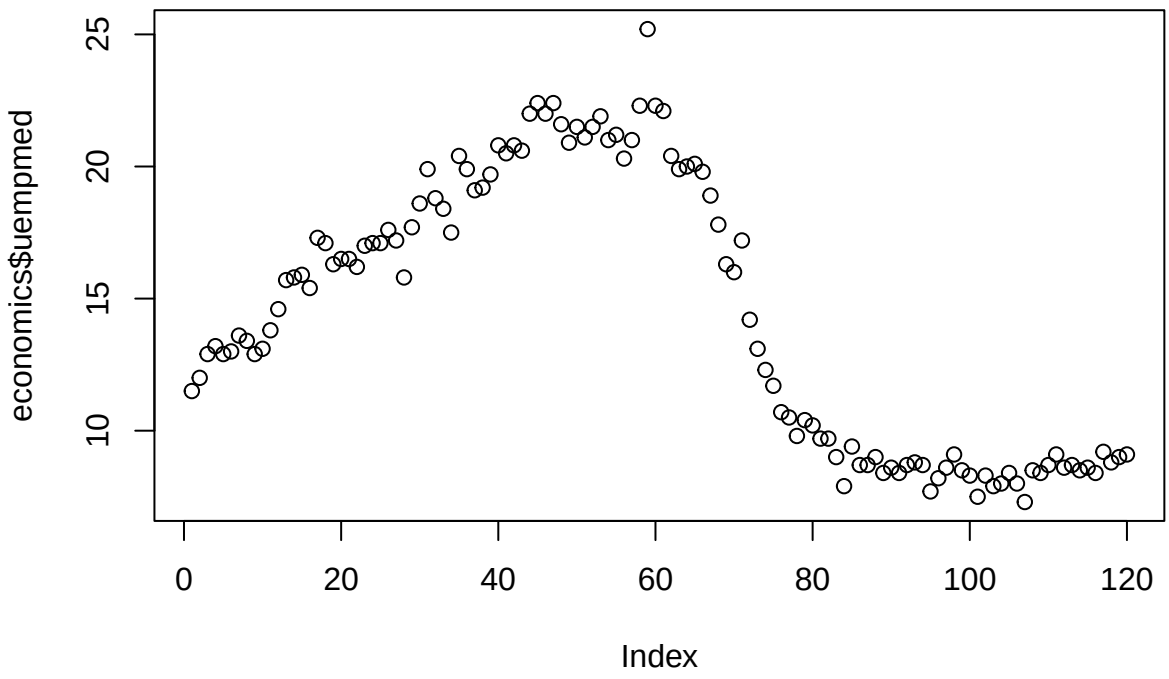
Cross validation

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

Before starting let's prepare the data.

```
data(economics)
economics <- economics[dim(economics)[1]:(dim(economics)[1]-119),]
economics$index <- 1:120
plot(economics$uempmed)
```



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.

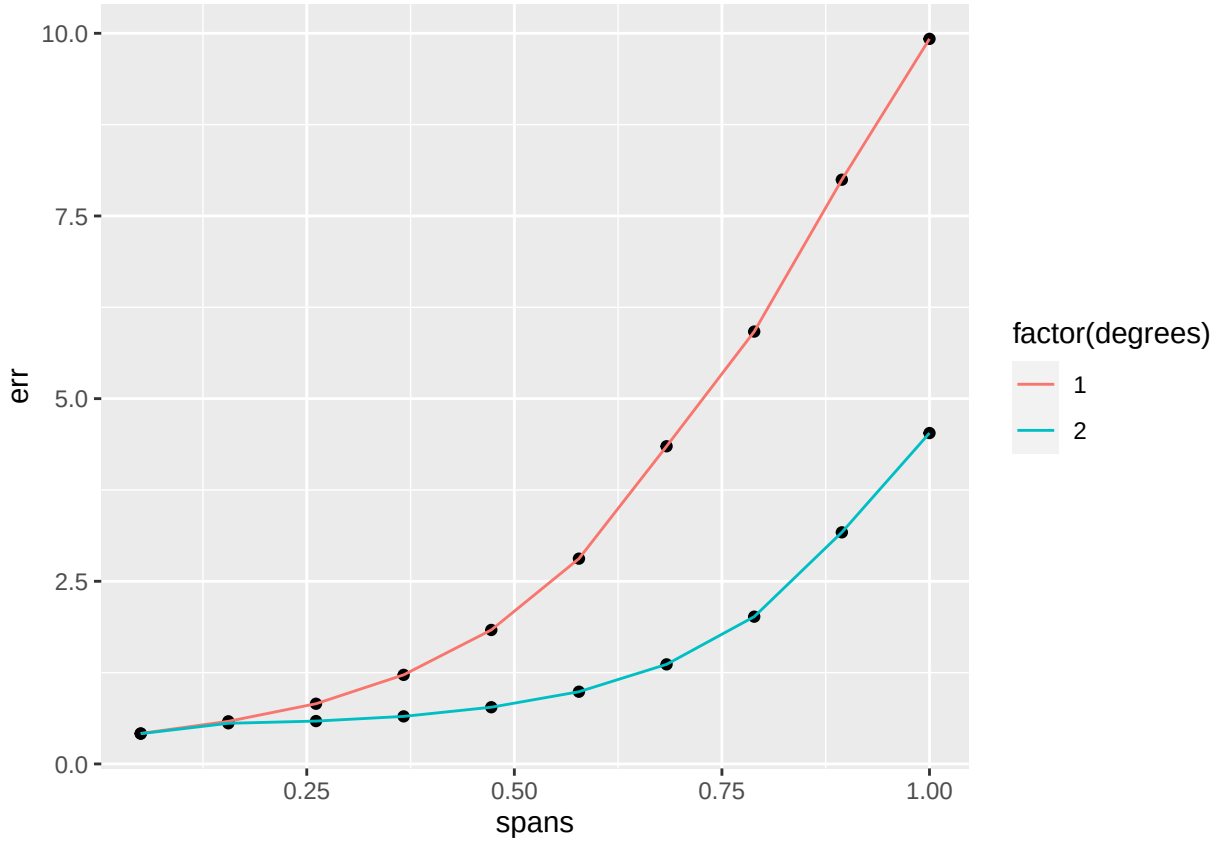
```
set.seed(2022)

degree_list <- list()
span_values <- seq(0.05,1,length=10)
for(deg in 1:2){ #polynomial degree
  err <- list()
  for(k in 1:length(span_values)){ #smoothness
    score <- list()
    for(i in 1:(nrow(economics))){
      training = economics[-i,]
      model = loess(uempmed ~ index, data = training, span = span_values[k], degree=deg)
      validation = economics[i,]
      pred = predict(model, validation)
      # error of ith fold
```

```
      score[[i]] = (validation$uempmed - 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
p <- ggplot(df_toplot, aes(x=spans, y=err, group=factor(degrees))) +
  geom_point() + geom_line(aes(col=factor(degrees)))
p
```



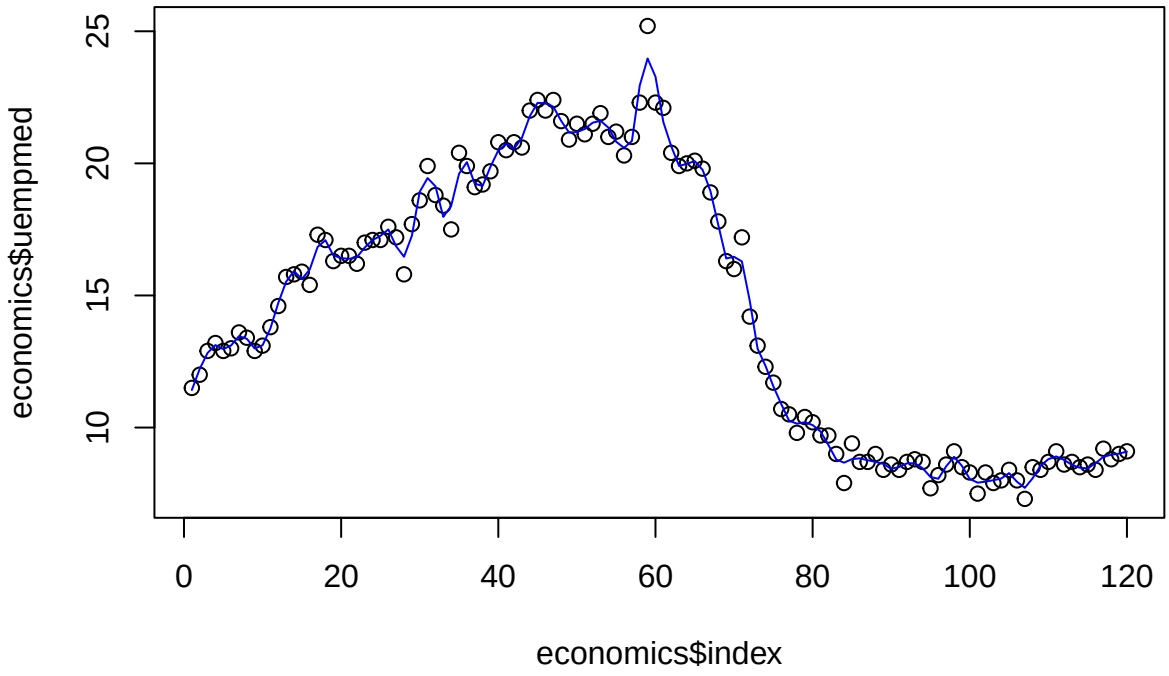
Let's find the parameters corresponding to the minimum error.

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

```
##    spans degrees       err
## 11  0.05       2 0.4142198
```

Let's plot the resulting smoothed curve.

```
res <- loess(uempmed ~ index, data = economics, span =best$spans, degree=best$degrees)
plot(economics$index, economics$uempmed)
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:

```
flds <- caret::createFolds(1:nrow(economics),
                           k = 3, list = TRUE,
                           returnTrain = FALSE)
flds
```

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

```
class(flds)
```

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

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

```
## $Fold1
##  [1]   6  11  12  14  15  22  23  25  28  29  31  32  38  41  43  45  49  56  58
## [20]  59  61  64  66  67  68  69  70  71  86  89  93  94  95 103 106 108 113 114
## [39] 115 120
```

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

```
##  [1]   6  11  12  14  15  22  23  25  28  29  31  32  38  41  43  45  49  56  58
## [20]  59  61  64  66  67  68  69  70  71  86  89  93  94  95 103 106 108 113 114
## [39] 115 120
```

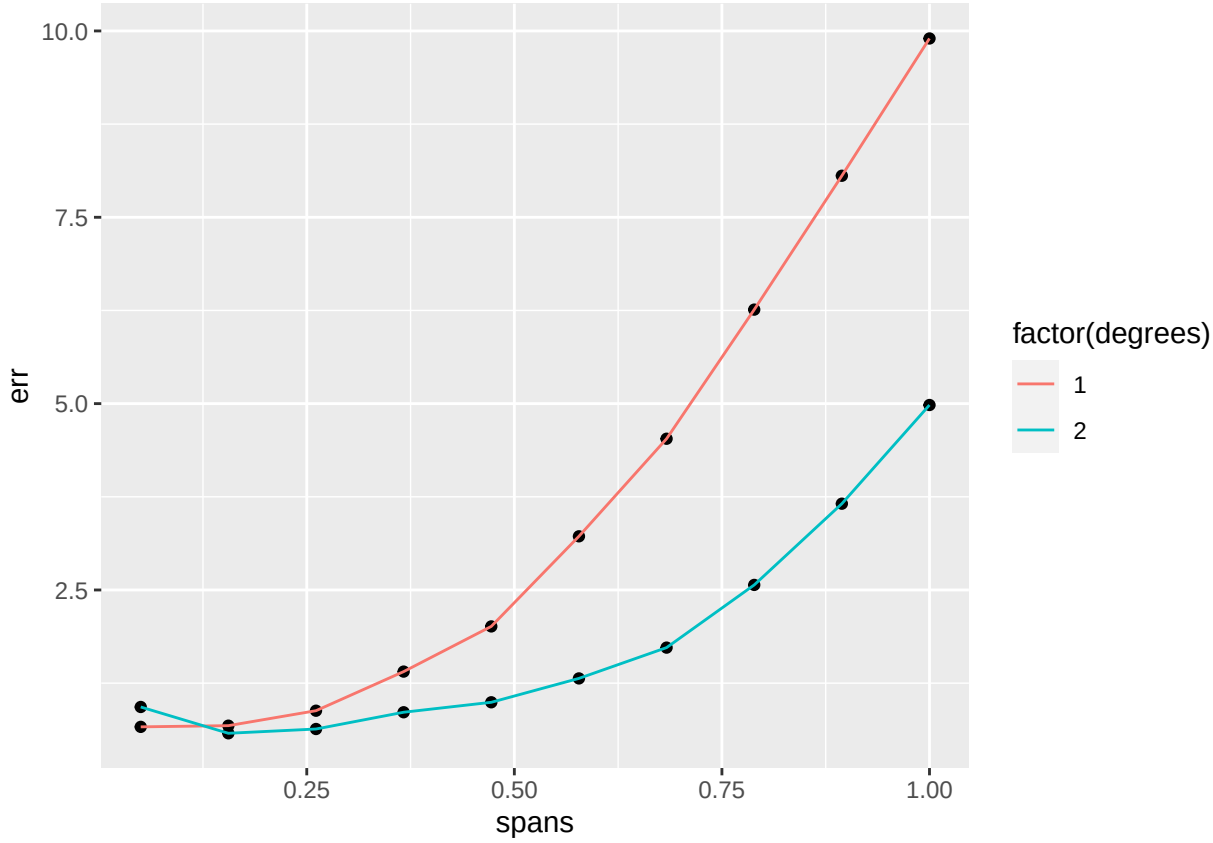
Perform an iteration similarly to the one for LOOCV:

```
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 <- economics[unlist(flds[i]),]
      training <- economics[unlist(flds[-i]),]
      model = loess(uempmed ~ index, data = training, span = span_values[k], degree=deg)
      pred = predict(model, validation)
```

```
      score[[i]] <- mean((pred - validation$uempmed)^2, na.rm=TRUE)
    }
    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))


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



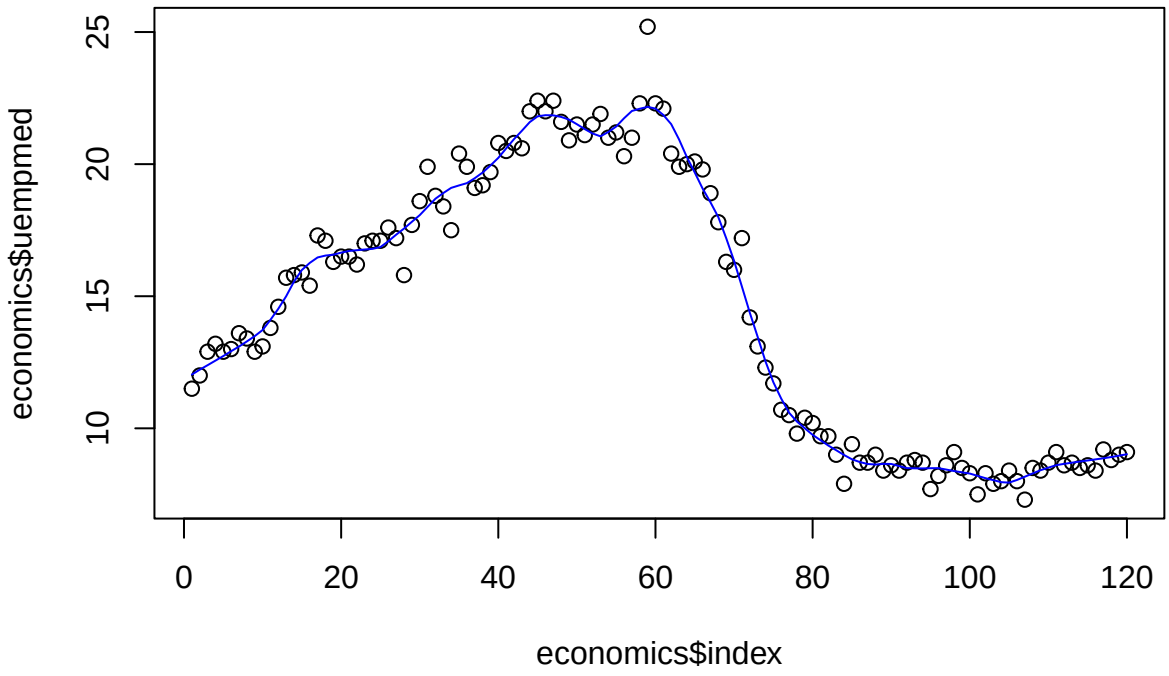
Let us find the parameters corresponding to the minimum error.

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

```
##        spans degrees       err
## 12 0.1555556       2 0.5775325
```

Let us plot the resulting regression line.

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



Using caret for CV

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:

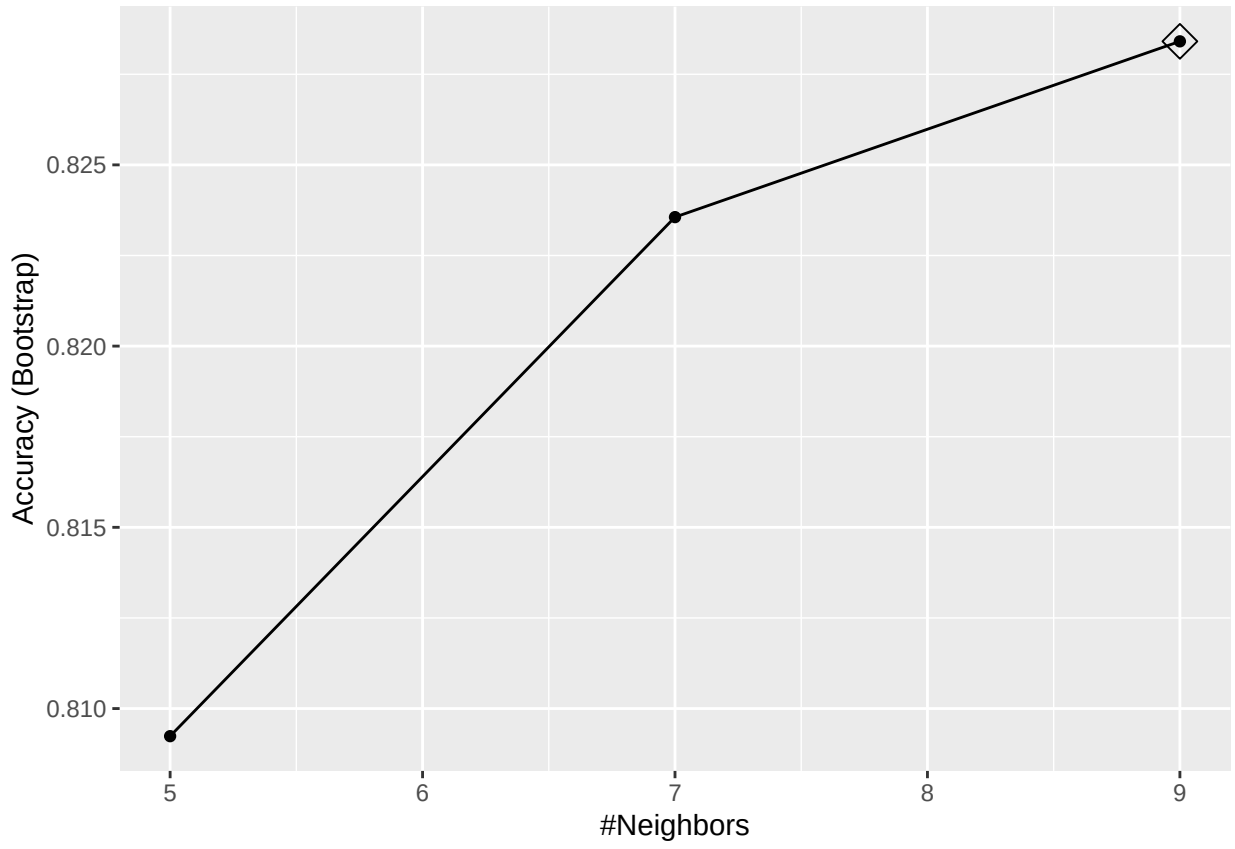
```
train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
confusionMatrix(y_hat_knn, mnist_27$test$y)
```

```
## 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:

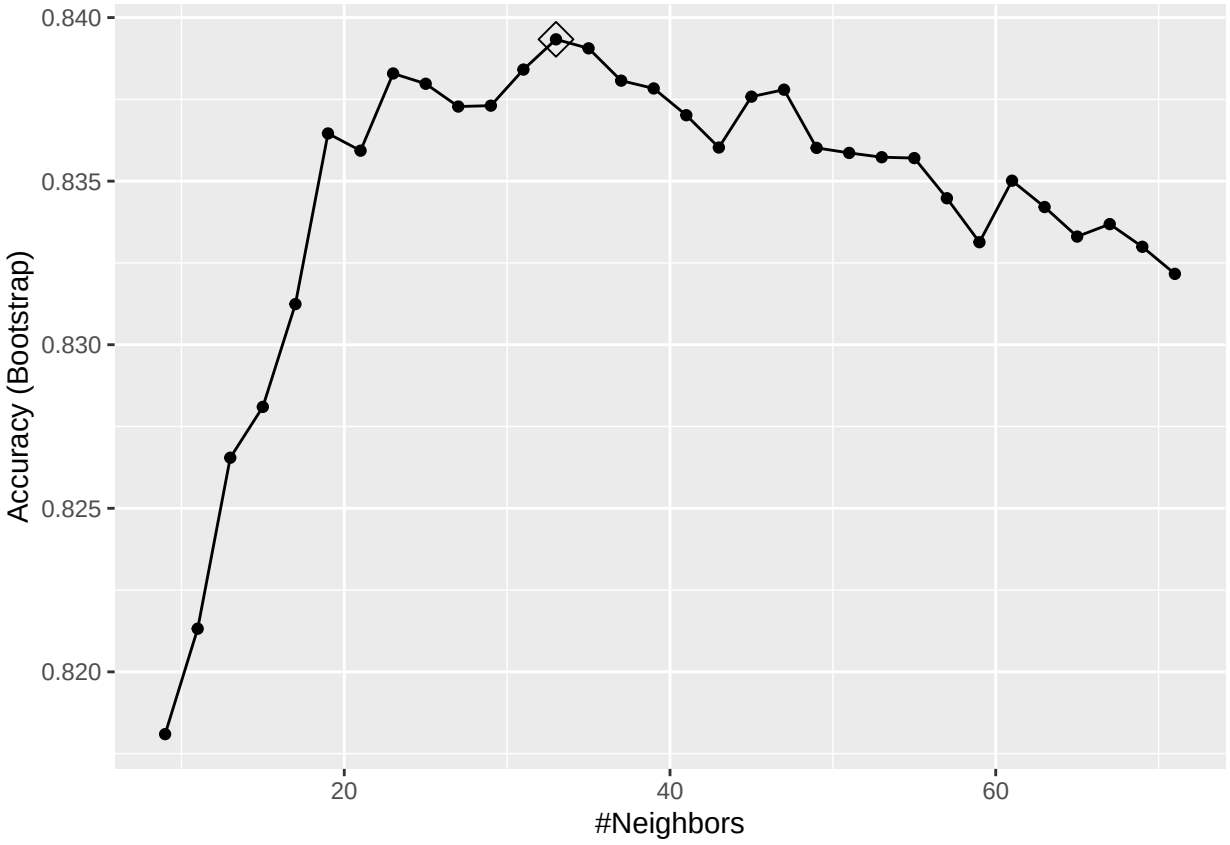
```
ggplot(train_knn, highlight = TRUE)
```



By default, cross validation is performed by taking 25 bootstrap samples comprising 25% of the observations. For the kNN method, the default is to try $k = (5, 7, 9)$.

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

```
set.seed(2022)
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
## 13 33
```

```
train_knn$finalModel
```

```
## 33-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 93 18
##          7 13 76
##
##                Accuracy : 0.845
##                  95% CI : (0.7873, 0.8922)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6879
##
##  Mcnemar's Test P-Value : 0.4725
##
##             Sensitivity : 0.8774
##             Specificity : 0.8085
##          Pos Pred Value : 0.8378
##          Neg Pred Value : 0.8539
##              Prevalence : 0.5300
##          Detection Rate : 0.4650
##    Detection Prevalence : 0.5550
##       Balanced Accuracy : 0.8429
##
##        'Positive' Class : 2
##
```

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

```
## $train
## $train$Training90
##  [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
##
##
## $test
## $test$Testing90
##  [1]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109
## [20] 110 111 112 113 114 115 116 117 118 119 120
```

Example 2: rolling window

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

```
## $train
## $train$Training90
##  [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
##
## $train$Training91
##  [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
##
## $train$Training92
##  [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
## [26] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
## [51] 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
## [76] 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
##
##
## $test
## $test$Testing90
##  [1]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109
## [20] 110 111 112 113 114 115 116 117 118
##
```

```
## $test$Testing91
##  [1]  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110
## [20] 111 112 113 114 115 116 117 118 119
##
## $test$Testing92
##  [1]  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111
## [20] 112 113 114 115 116 117 118 119 120
```

Example 3: recursive window

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

```
## $train
## $train$Training90
##  [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
##
## $train$Training91
##  [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
##
## $train$Training92
##  [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
##
##
## $test
## $test$Testing90
##  [1]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109
## [20] 110 111 112 113 114 115 116 117 118
##
## $test$Testing91
##  [1]  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110
## [20] 111 112 113 114 115 116 117 118 119
##
## $test$Testing92
##  [1]  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111
## [20] 112 113 114 115 116 117 118 119 120
```

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)
```