crossvalidation

J. Di Iorio, F. Chiaromonte

3/8/2021

Libraries

We are going to use **tidyverse** and **ggplot2**.

```
library(tidyverse) # for data manipulation and visualization
```

```
## -- Attaching packages --------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(ggplot2) # for plots
library(caret) # for statistical learning techniques
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

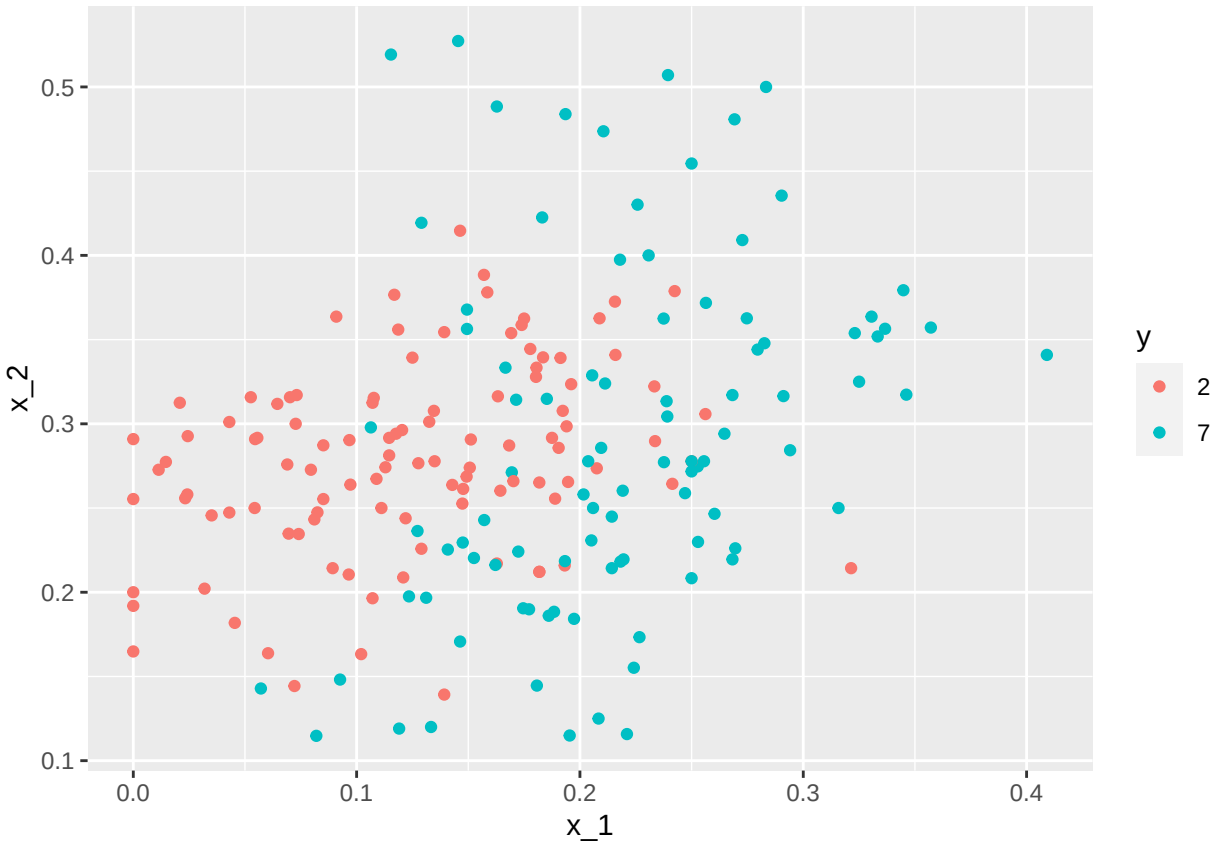
```
library(dslabs)
```

Data

We will use dataset already presented in the previous lessons (the **economics** dataset from **ggplot2** package) but also the \textbf{mnist_27} dataset in the **dslabs** package.

The **mnist** dataset is a 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 \textbf{mnist_27} data set we only include a randomly selected set of 2s and 7s 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()
```



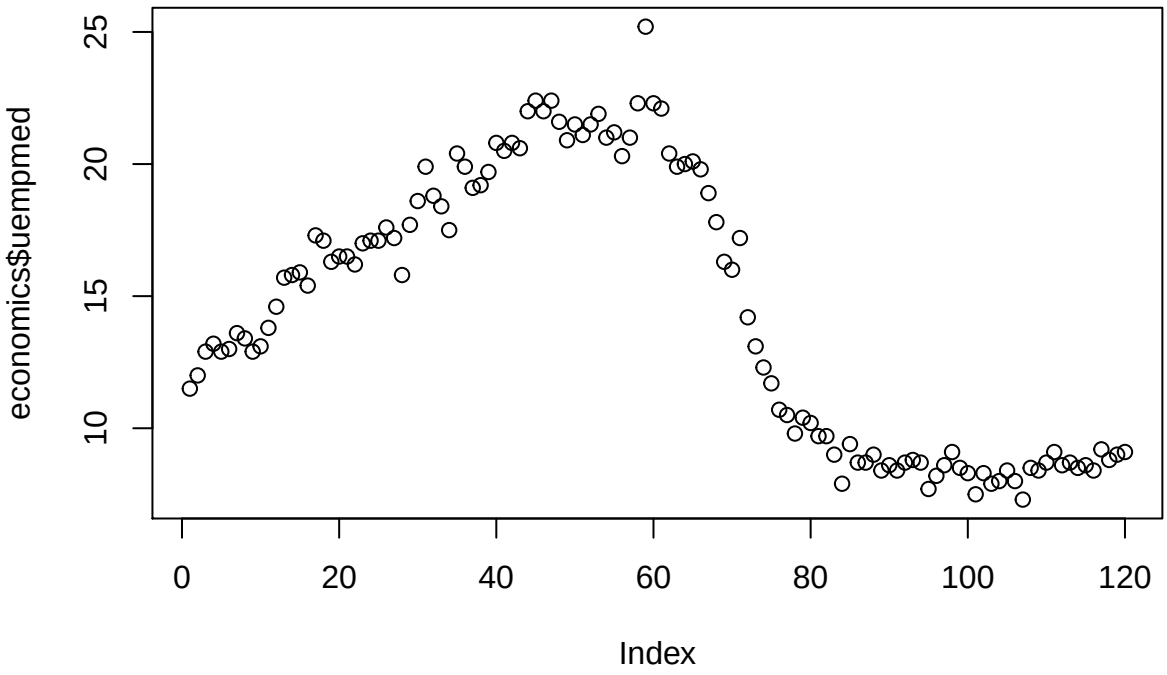
Crossvalidation

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

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

Before starting let us prepare the data.

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



LOOCV

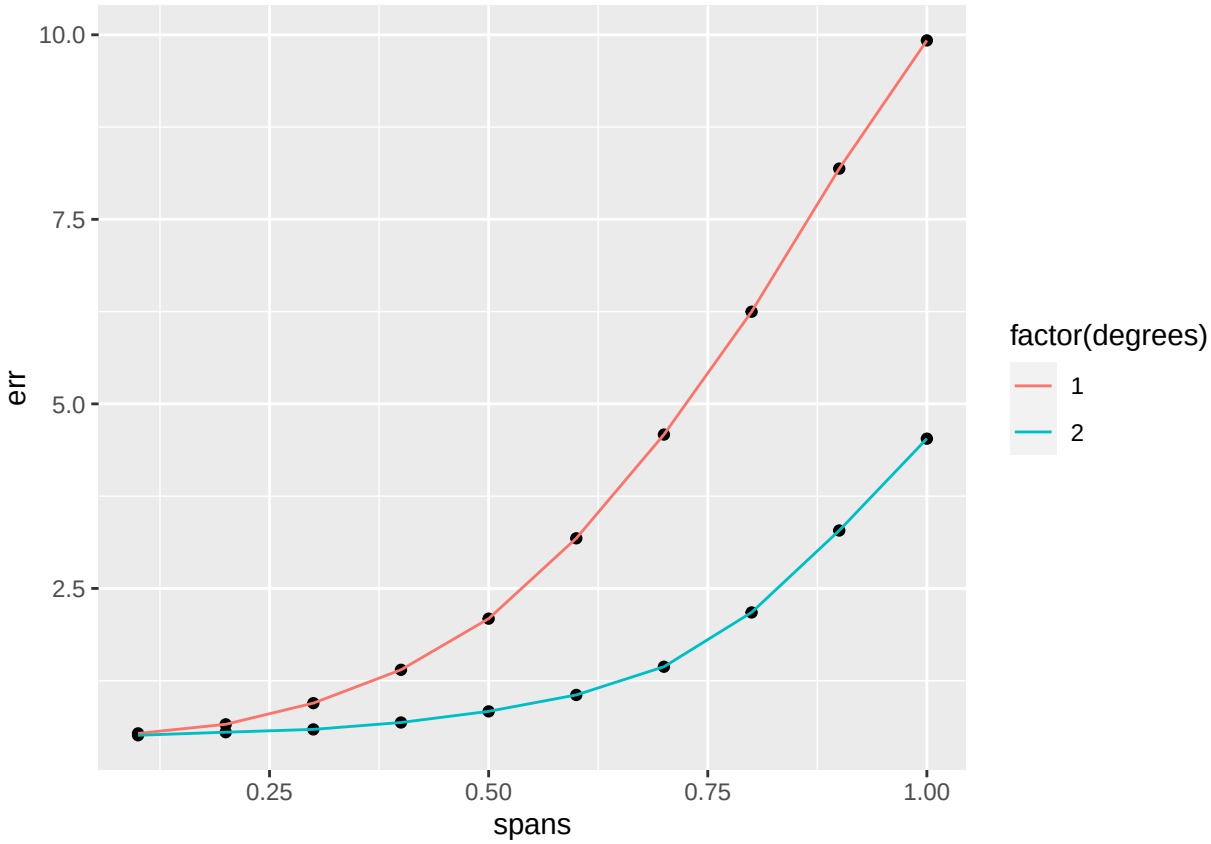
In this approach, we reserve only one data point from the available dataset, and train the model on the rest of the data. This process iterates for each data point.

```
degree_list <- list()
span_values <- seq(0.1,1,0.1)
for(deg in 1:2){
  err <- list()
  for(k in 1:length(span_values)){
    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
    }
  err[[k]] <- mean(unlist(score),na.rm=TRUE) # returns a vector
  }
  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=fa
p
```



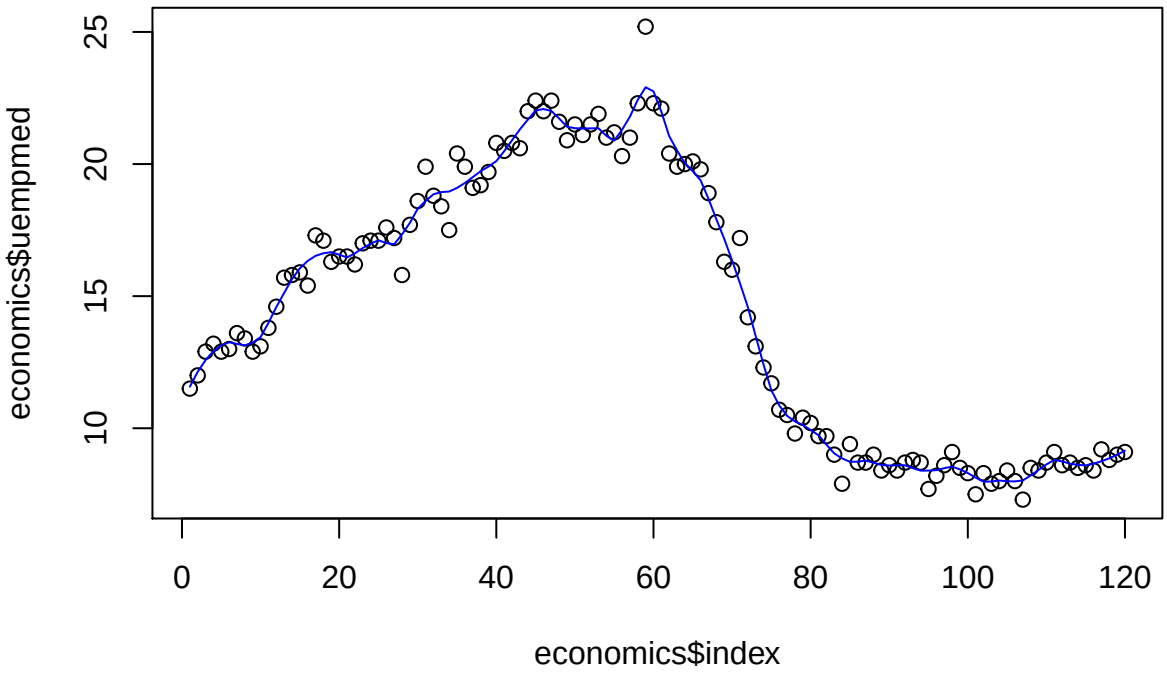
Let us find the parameters corresponding to the minimum error.

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

```
##    spans degrees       err
## 11   0.1       2 0.5108204
```

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')
```



k-fold CV

Let us validate the parameter using the k-fold Crossvalidation. Below are the steps for it:

1. Randomly split your entire dataset into k folds;
2. For each k-fold in your dataset, build your model on k˘1 folds of the dataset. Then, test the model to check the effectiveness for kth fold;
3. Record the error you see on each of the 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 your performance metric for the model;

Create the folds

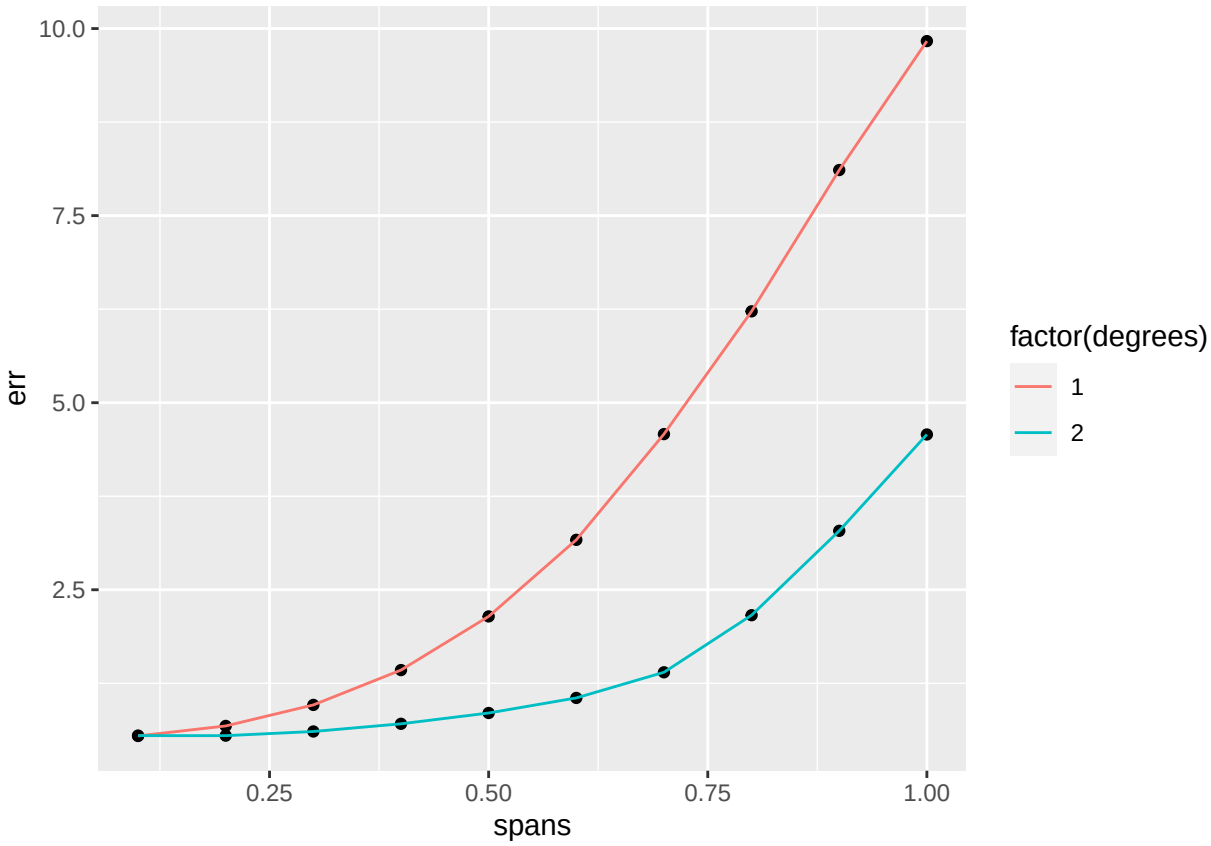
```
library(caret)
flds <- createFolds(1:120, k = 10, list = TRUE, returnTrain = FALSE)
```

```
degree_list <- list()
span_values <- seq(0.1,1,0.1)
for(deg in 1:2){ #polinomials degree
  err <- list()
  for(k in 1:length(span_values)){ #smoothness
    score <- list()
    for(i in 1:10){
      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=fa
p
```



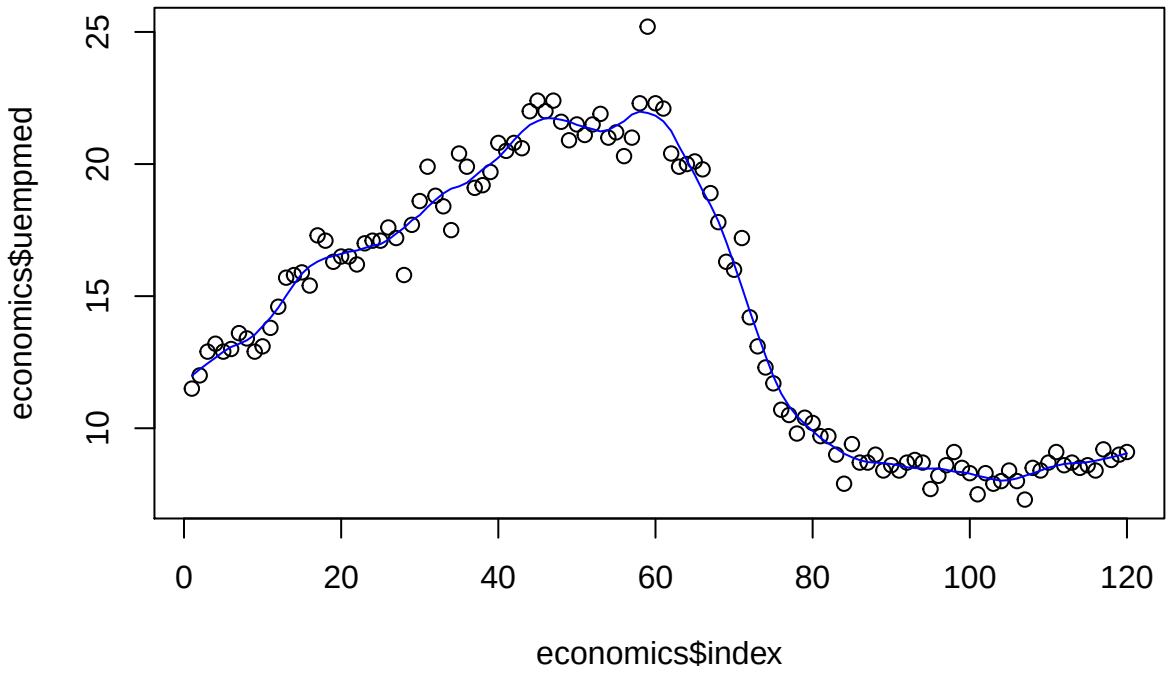
Let us find the parameters corresponding to the minimum error.

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

```
##   spans degrees       err
## 1   0.1       1 0.5464208
```

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

The crossvalidation is automatically implemented in the train function of the caret package. The caret train function lets us train different algorithms using similar syntax. So, for example, we can type:

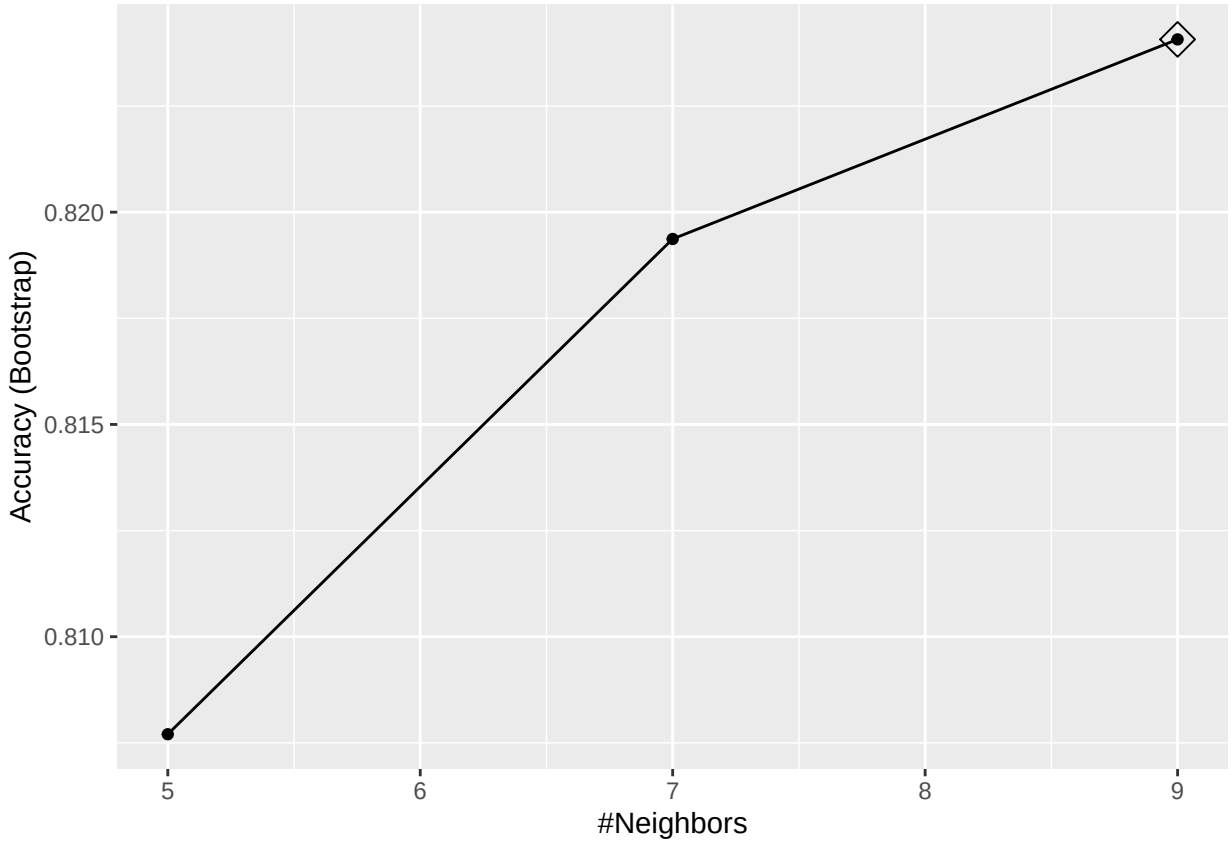
```
library(caret)
train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
confusionMatrix(y_hat_knn, mnist_27$test$y)
```

```
## 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
##
```

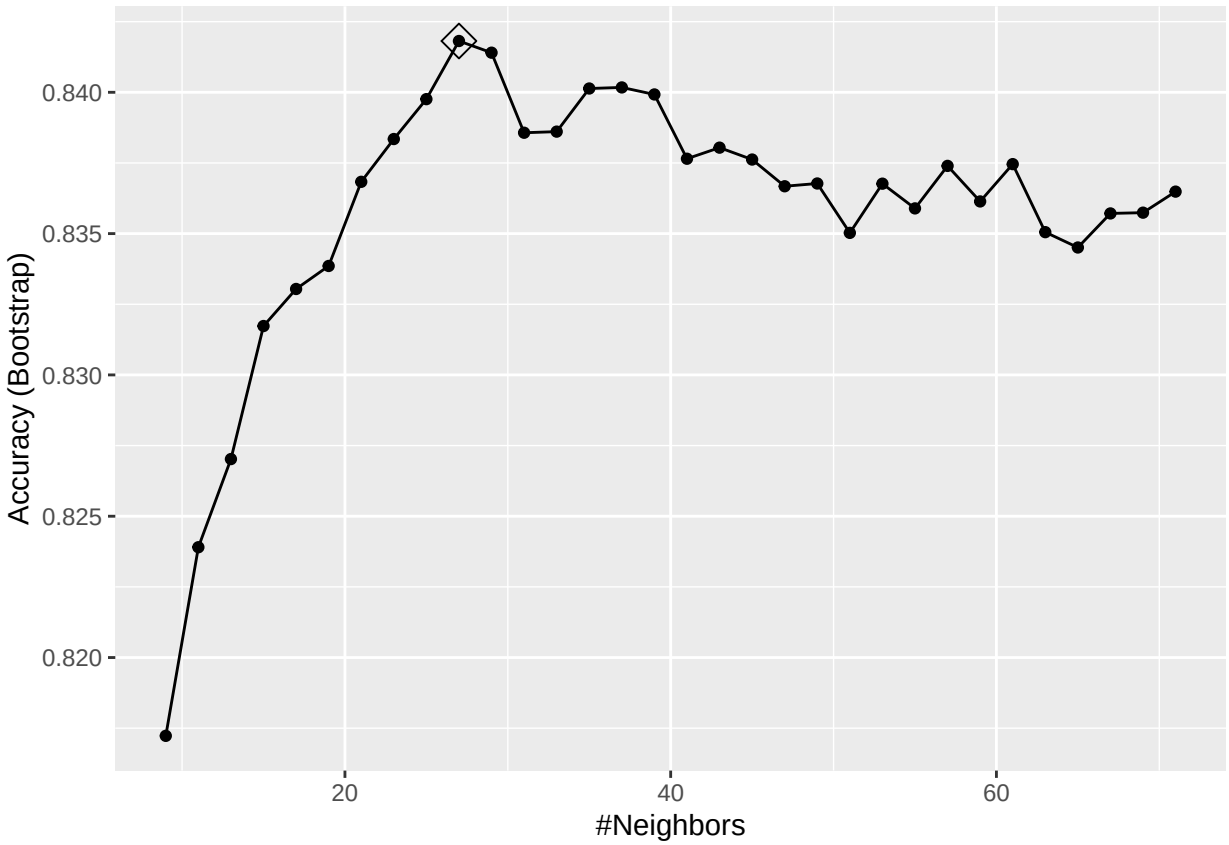
Including a tuning parameter, train automatically uses cross validation to decide among a few default values. You can quickly see the results of the crossvalidation using the **ggplot** function. The argument **highlight** highlights the max:

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



By default, the cross validation is performed by taking 25 bootstrap samples comprised of 25% of the observations. For the kNN method, the default is to try
$k = (5, 7, 9)$. We change this using the **tuneGrid** parameter.

```
set.seed(2008)
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 in the following way:

```
train_knn$bestTune
```

```
##     k
## 10 27
```

```
train_knn$finalModel
```

```
## 27-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 92 19
##          7 14 75
##
##                Accuracy : 0.835
##                  95% CI : (0.7762, 0.8836)
##     No Information Rate : 0.53
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6678
##
##  Mcnemar's Test P-Value : 0.4862
##
##             Sensitivity : 0.8679
##             Specificity : 0.7979
##          Pos Pred Value : 0.8288
##          Neg Pred Value : 0.8427
##              Prevalence : 0.5300
##          Detection Rate : 0.4600
##    Detection Prevalence : 0.5550
##       Balanced Accuracy : 0.8329
##
##        'Positive' Class : 2
##
```