K Nearest Neighbours

kNN

Show me your friends and I’ll tell you who you are

Characteristics:

Data-driven, not model-driven

Makes no assumptions about the data

Lazy-learning algorithm

lazy learning is a learning method in which generalization beyond the training data is delayed until a query is made to the system, as opposed to in eager learning, where the system tries to generalize the training data before receiving queries.

Basic Idea

For a given record to be classified, identify nearby records

“Near” means records with similar predictor values $X_1, X_2, \dots X_p$

Classify the record as whatever the predominant class is among the nearby records (the “neighbors”)

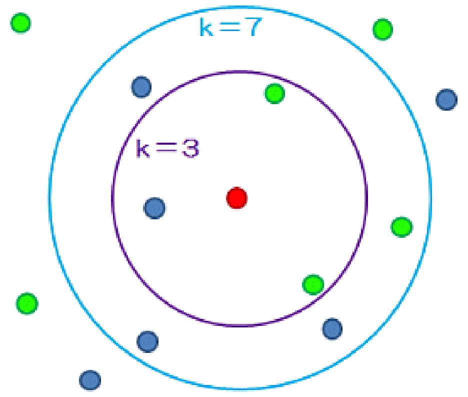
K-Nearest Neighbors method

Tell me who are your friends, I will tell you who you are!

We tend to group with people of similar attributes so does data.

K-NN: Introduction

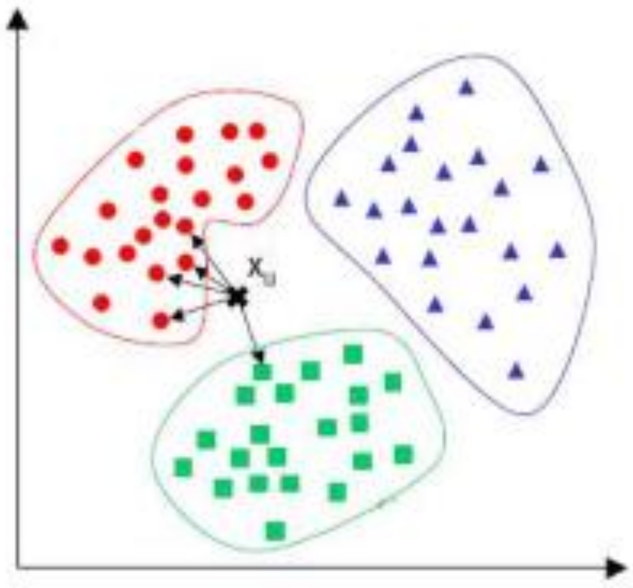
K-Nearest Neighbor is considered a lazy learning algorithm that classifies data sets based on their similarity with neighbors.



"K" stands for number of data set items that are considered for the classification.
Ex: Image shows classification for different k-values.

K-NN classification

Prediction Rule: Look at the K most similar training examples



For classification: assign the majority class label (majority voting)

The algorithm requires:
Parameter K: number of nearest neighbors to look for
Distance function: To compute the similarities between examples

Distance measures

Measuring Similarity between Objects

- Distance on Numeric Data: Euclidian distance, Manhattan distance
- Proximity Measure for Symmetric vs. Asymmetric Binary Variables
- Distance between Categorical Attributes, Ordinal Attributes, and Mixed Types
- Proximity Measure between Two Vectors: Cosine Similarity
- Correlation Measures between Two Variables: Covariance and Correlation Coefficient

Distance between scaled variables

Minkowski distance: A popular distance measure

$$d(i,j) = \sqrt[p]{|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \cdots + |x_{il} - x_{jl}|^p}$$

- Properties
  - d(i, j) > 0 if i ≠ j, and d(i, i) = 0 (Positivity)
  - d(i, j) = d(j, i) (Symmetry)
  - d(i, j) ≤ d(i, k) + d(k, j) (Triangle Inequality)

Choosing K

Small K

- Creates many small regions for each class
- May lead to overfit, non-smooth decision boundaries

Large K

- Creates fewer larger regions
- Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit). Prediction is going to be too general.

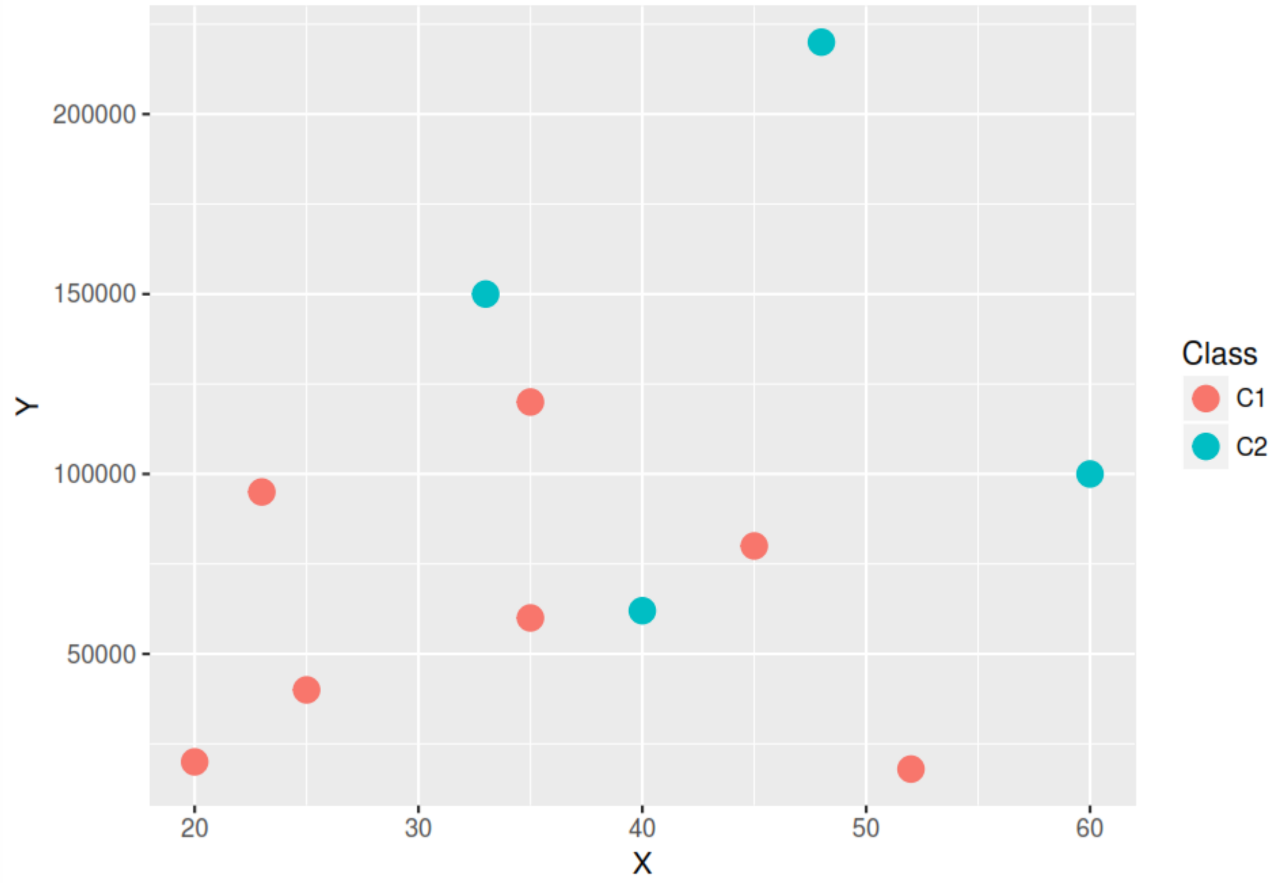
Choosing K

- Often data dependent and heuristic based
- Or using cross-validation (using some hold-out data)
- In general, a K too small or too big is bad!

Doing in R

```r
example<-read.csv("example.csv")
```

```r
library(ggplot2)
ggplot(example, aes(x=X, y=Y, color=Class))+
  geom_point(size=4)
```

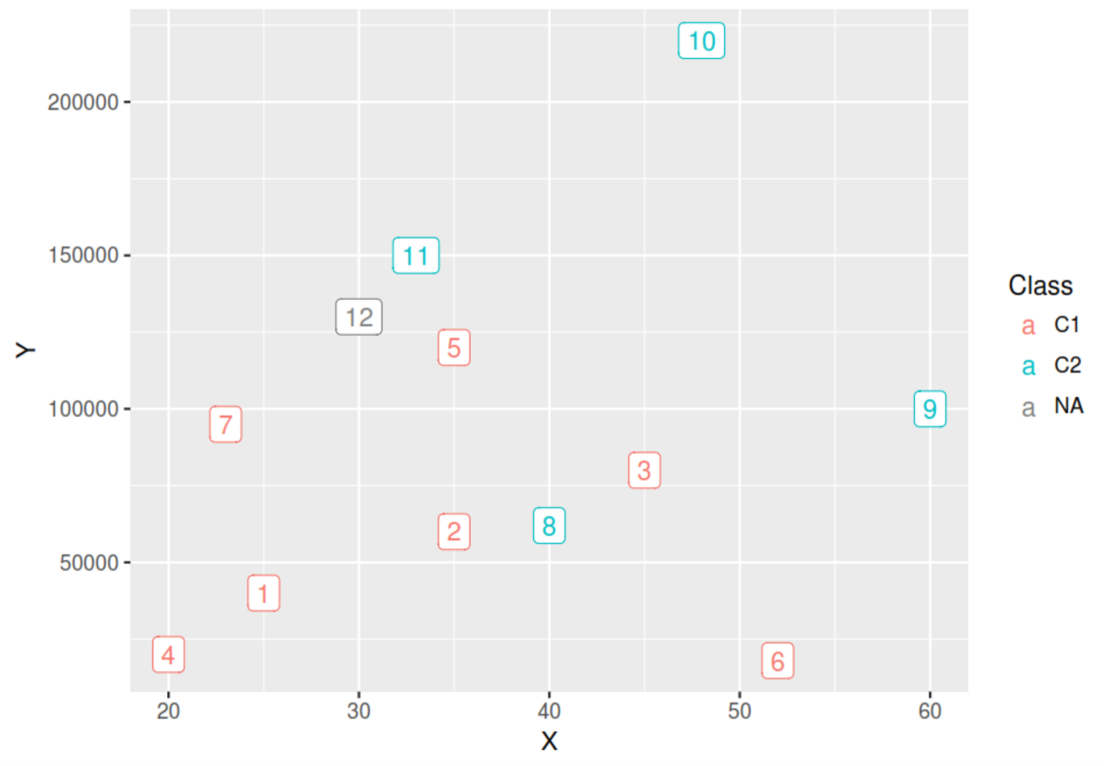


Doing in R

Add case N12 with unknown label: we need to predict its label
Plot with rownumbers

```
example[12,]<-c(30,130000,NA)


ggplot(example, aes(x=X, y=Y, color=Class))+
geom_point(size=4)+
  geom_label(label=rownames(example))
```



Doing in R

Make the distance matrix
Select distances from the case N 12 (with unknown label)
sort by descending order

```
d<-as.matrix(dist(example[,1:2]))
# look for the case N 12
sort(d[,12])
```

```
##        12         5        11         9         7         3         8
##      0.00  10000.00  20000.00  30000.01  35000.00  50000.00  68000.00
##         2         1        10         4         6
##  70000.00  90000.00  90000.00 110000.00 112000.00
```

Cases 5,11,9, predicted class label: C2

```
example
```

```
##      X      Y Class
## 1  25  40000    C1
## 2  35  60000    C1
## 3  45  80000    C1
## 4  20  20000    C1
## 5  35 120000    C1
## 6  52  18000    C1
## 7  23  95000    C1
## 8  40  62000    C2
## 9  60 100000    C2
## 10 48 220000    C2
## 11 33 150000    C2
## 12 30 130000  <NA>
```

Normalization of independent variables

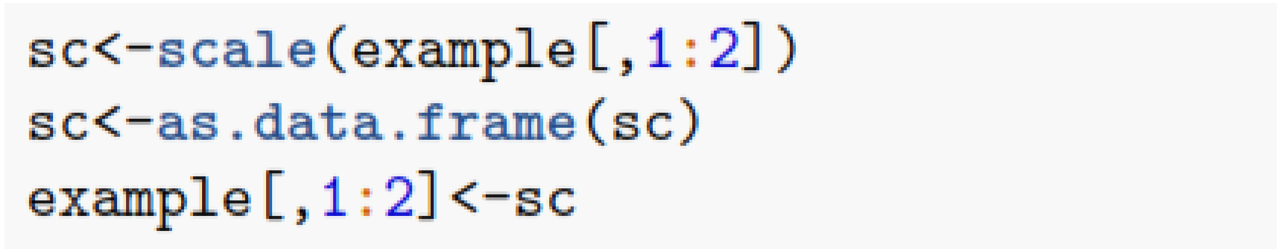
- Note: Features should be on the same scale
- The variable with “bigger” variance will have more importance in calculating the distance.
  - Example: Age in years, Income in 000’ AMD
- This should be fixed with normalization
- Do normalization before running the model

Types of normalization

Min-Max normalization: $X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$

Z-score: $X_{new} = \frac{X - \bar{X}}{\sigma}$

Calculate z-score for each column



```
sc<-scale(example[,1:2])
sc<-as.data.frame(sc)
example[,1:2]<-sc
```

Doing in R

cases 11,5,7 predicted class: C1

```
example
```

```
##              X           Y Class
## 1  -0.9935045 -0.87805908    C1
## 2  -0.1769255 -0.53540188    C1
## 3   0.6396536 -0.19274468    C1
## 4  -1.4017941 -1.22071628    C1
## 5  -0.1769255  0.49256973    C1
## 6   1.2112590 -1.25498200    C1
## 7  -1.1568203  0.06424823    C1
## 8   0.2313641 -0.50113616    C2
## 9   1.8645222  0.14991253    C2
## 10  0.8846273  2.20585573    C2
## 11 -0.3402413  1.00655553    C2
## 12 -0.5852150  0.66389833  <NA>
```

Doing in R

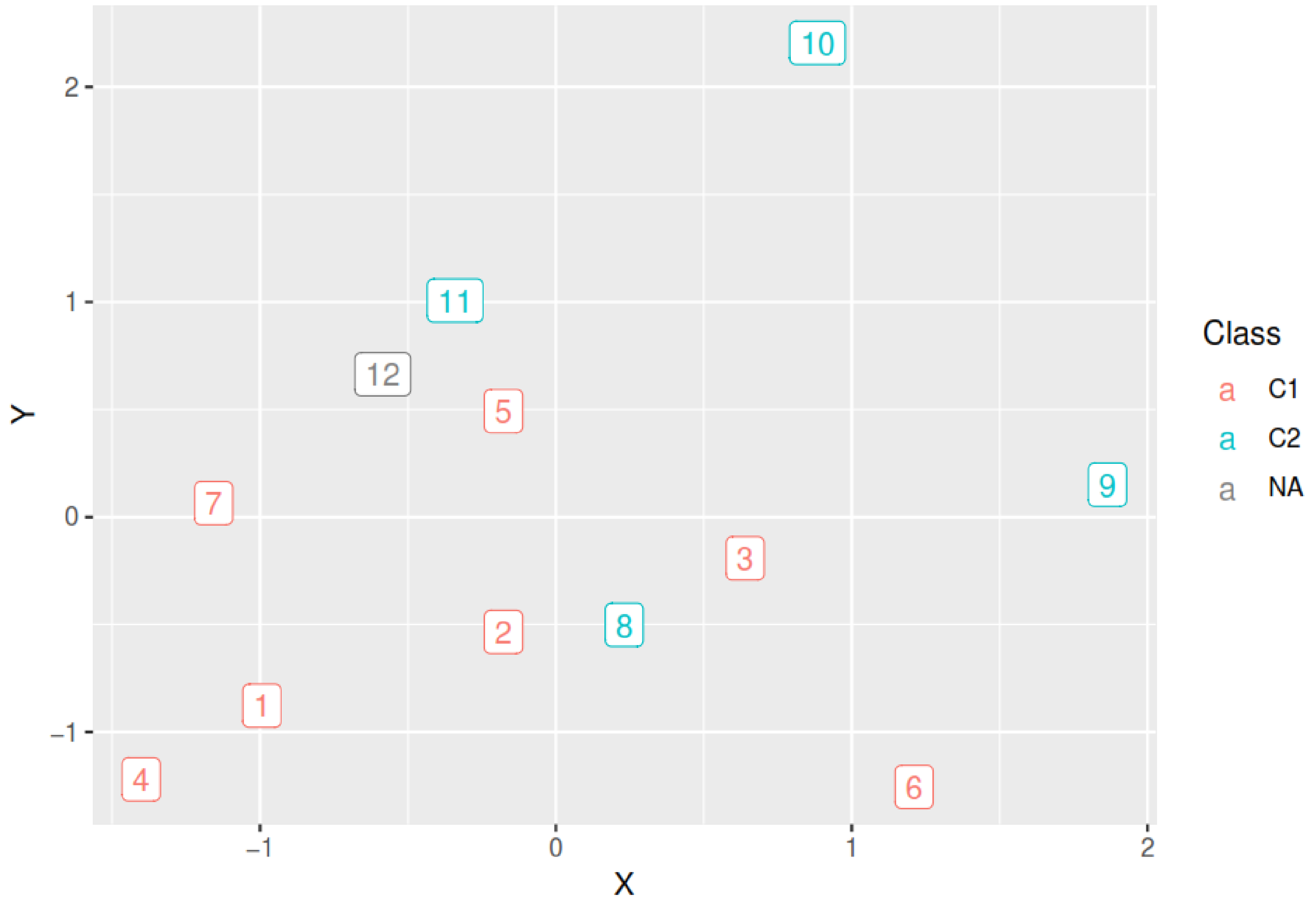
The problem ?
We need to scale

```r
d<-as.matrix(dist(example[,1:2]))
# look for the case N 12
sort(d[,12])
```

```
##        12        11         5         7         2         8         3
## 0.0000000 0.4212198 0.4427797 0.8284401 1.2668944 1.4227110 1.4947041
##         1         4        10         9         6
## 1.5950965 2.0539166 2.1302744 2.5030769 2.6285776
```

scatter plot with scaled variables

```
ggplot(example, aes(x=X, y=Y, color=Class))+
geom_point(size=4)+
  geom_label(label=rownames(example))
```

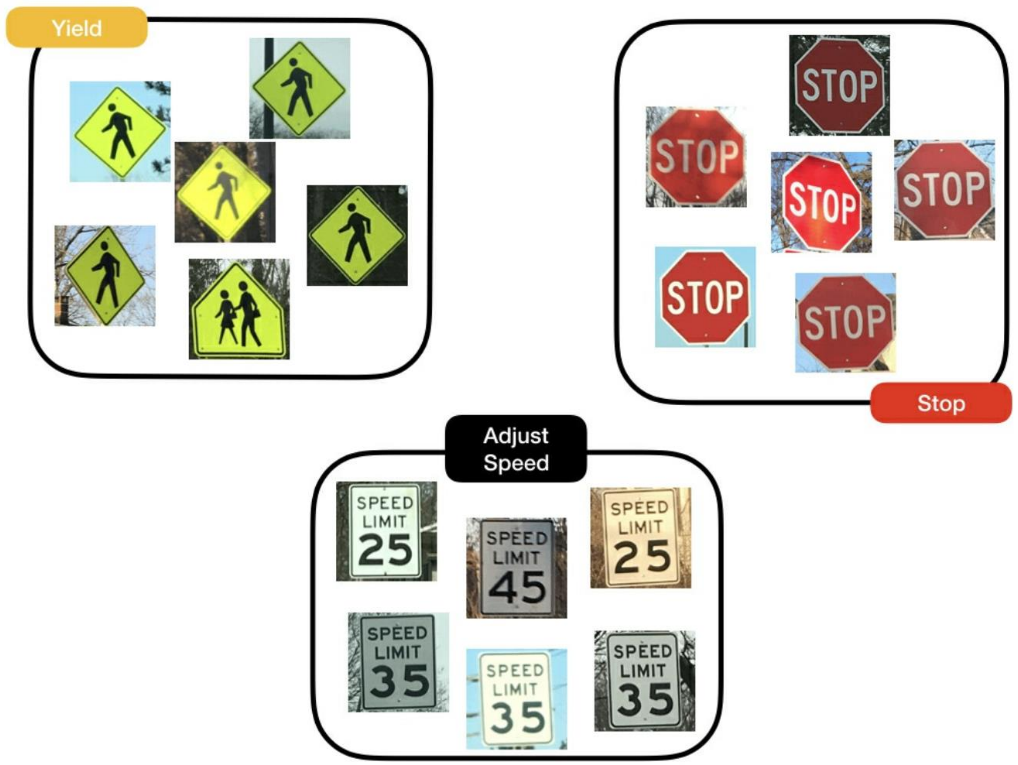


Doing in R

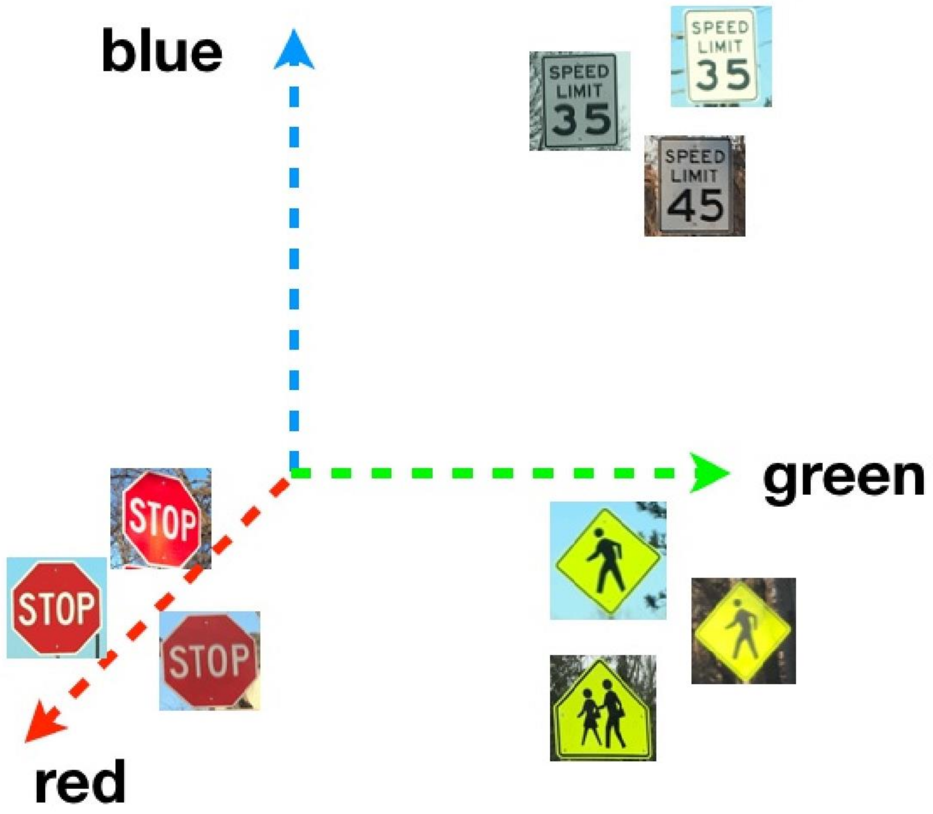
Classification tasks for driverless cars

Doing in R

The dependent variable is not binary, it has 3 levels:

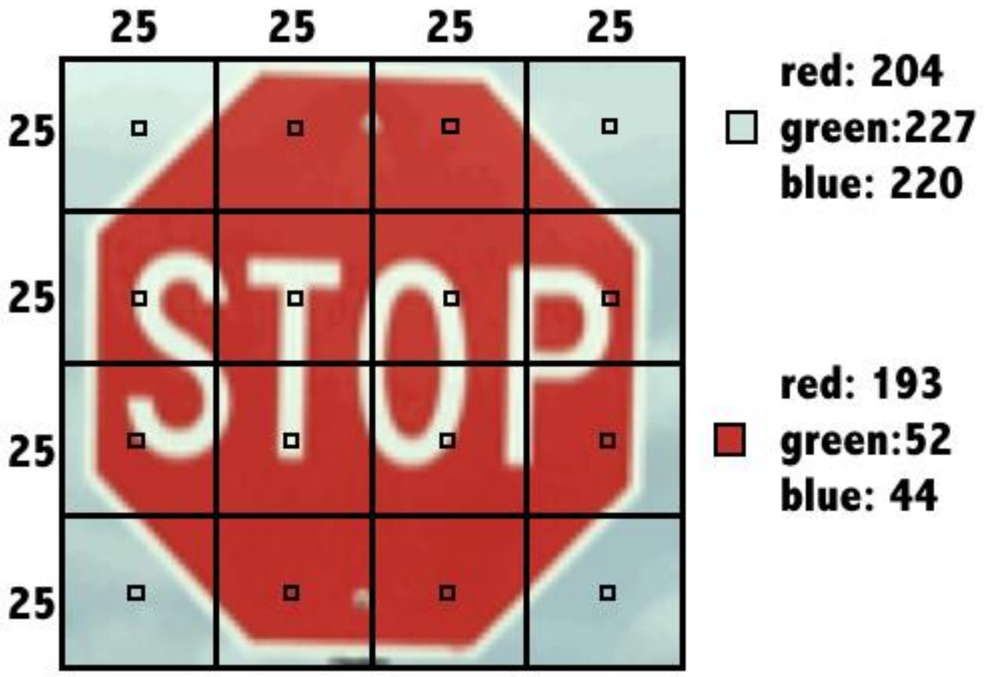


We can differentiate the signs by the color



The data

Each previously observed street sign was divided into a 4x4 grid, and the red, green, and blue level for each of the 16 center pixels is recorded as illustrated here.
Overall 16 variables for each sign



Data structure

The variable sample shows if the case belongs to test or train set
sign_type is the actual label

```r
signs<-read.csv("signs.csv")
str(signs)
```

```
## 'data.frame':    205 obs. of  50 variables:
##  $ sample   : Factor w/ 2 levels "test","train": 2 2 2 2 2 2 2 1 2 2 ...
##  $ sign_type: Factor w/ 3 levels "pedestrian","speed",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ r1       : int  155 142 57 22 169 75 136 118 149 13 ...
##  $ g1       : int  228 217 54 35 179 67 149 105 225 34 ...
##  $ b1       : int  251 242 50 41 170 60 157 69 241 28 ...
##  $ r2       : int  135 166 187 171 231 131 200 244 34 5 ...
##  $ g2       : int  188 204 201 178 254 89 203 245 45 21 ...
##  $ b2       : int  101 44 68 26 27 53 107 67 1 11 ...
##  $ r3       : int  156 142 51 19 97 214 150 132 155 123 ...
##  $ g3       : int  227 217 51 27 107 144 167 123 226 154 ...
##  $ b3       : int  245 242 45 29 99 75 134 12 238 140 ...
##  $ r4       : int  145 147 59 19 123 156 171 138 147 21 ...
##  $ g4       : int  211 219 62 27 147 169 218 123 222 46 ...
##  $ b4       : int  228 242 65 29 152 190 252 85 242 41 ...
##  $ r5       : int  166 164 156 42 221 67 171 254 170 36 ...
##  $ g5       : int  233 228 171 37 236 50 158 254 191 60 ...
##  $ b5       : int  245 229 50 3 117 36 108 92 113 26 ...
##  $ r6       : int  212 84 254 217 205 37 157 241 26 75 ...
##  $ g6       : int  254 116 255 228 225 36 186 240 37 108 ...
##  $ b6       : int  52 17 36 19 80 42 11 108 12 44 ...
##  $ r7       : int  212 217 211 221 235 44 26 254 34 13 ...
##  $ g7       : int  254 254 226 235 254 42 35 254 45 27 ...
##  $ b7       : int  11 26 70 20 60 44 10 99 19 25 ...
##  $ r8       : int  188 155 78 181 90 192 180 108 221 133 ...
##  $ g8       : int  229 203 73 183 110 131 211 106 249 163 ...
##  $ b8       : int  117 128 64 73 9 73 236 27 184 126 ...
##  $ r9       : int  170 213 220 237 216 123 129 135 226 83 ...
##  $ g9       : int  216 253 234 234 236 74 109 123 246 125 ...
##  $ b9       : int  120 51 59 44 66 22 73 40 59 19 ...
##  $ r10      : int  211 217 254 251 229 36 161 254 30 13 ...
##  $ g10      : int  254 255 255 254 255 34 190 254 40 27 ...
##  $ b10      : int  3 21 51 2 12 37 10 115 34 25 ...
##  $ r11      : int  212 217 253 235 235 44 161 254 34 9 ...
```

kNN with library class

first divide the dataset into training and testing sets.
Then remove the variable sample from both

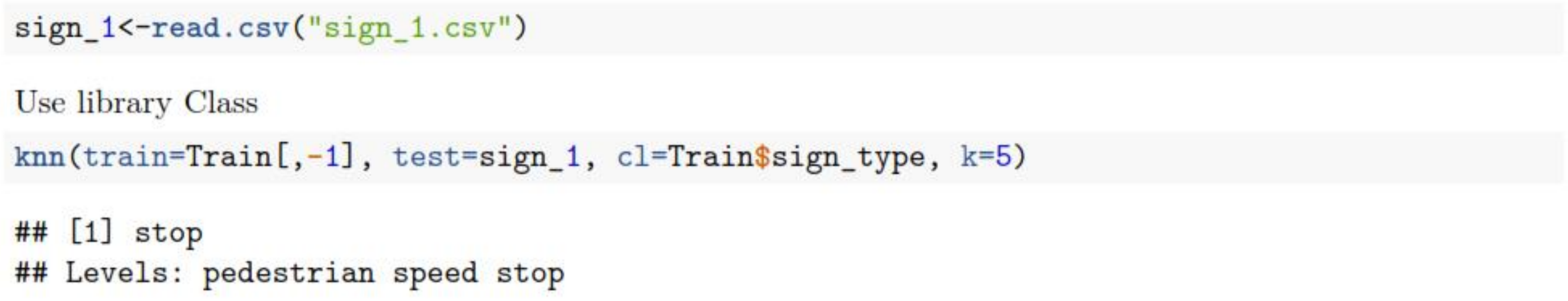
```r
library(class)
# training testing sets
Train<-signs[signs$sample=="train",]
Test<-signs[signs$sample=="test",]

Train$sample<-NULL
Test$sample<-NULL
```

kNN with library class

Make prediction for one case (sign _1). I randomly chose number of the neighbours
The function arguments.

- train: The training set without actual label
- test: the testing set without actual label
- cl: actual class labels from Train set
- k: the number of neighbors to consider



```
sign_1<-read.csv("sign_1.csv")
```

Use library Class

```
knn(train=Train[,-1], test=sign_1, cl=Train$sign_type, k=5)
```

```
## [1] stop
## Levels: pedestrian speed stop
```

Pay attention: there is no model or formula, it is just prediction

kNN with library class

Make prediction for the whole Test dataset.
Note: the first column of each dataframe is the actual class, so we take it out

```
knn1<-knn(Train[,-1], Test[,-1], cl=Train$sign_type, k=5)
knn1[1:20]
```

```
##  [1] pedestrian pedestrian pedestrian stop       pedestrian pedestrian
##  [7] pedestrian pedestrian pedestrian pedestrian pedestrian pedestrian
## [13] pedestrian pedestrian pedestrian pedestrian pedestrian pedestrian
## [19] pedestrian speed
## Levels: pedestrian speed stop
```

Check accuracy

```
table(knn1, Test$sign_type)
```

```
##
## knn1          pedestrian speed stop
##   pedestrian          18     0    0
##   speed                0    20    0
##   stop                 1     1   19
```

Average Accuracy

```
mean(knn1==Test$sign_type)
```

```
## [1] 0.9661017
```

Predict probabilities

- You can use knn function to predict probabilities of belonging to a class as well.
- use argument prob=T in the function
- access the probabilities using function attr, inside giving the object and attribute name “prob”

```
knn_p<-knn(Train[,-1], Test[,-1], cl=Train$sign_type,
           k=5, prob=T)
```

The probabilitie are saved as an attribute “prob” to access

```
attr(knn_p, "prob")
```

```
##  [1] 0.6 0.6 0.8 0.4 1.0 0.6 0.8 1.0 0.6 1.0 1.0 1.0 0.8 1.0 1.0 1.0 1.0
## [18] 1.0 0.8 0.8 0.6 1.0 1.0 1.0 1.0 1.0 1.0 0.6 0.4 1.0 1.0 1.0 1.0 1.0
## [35] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 1.0 0.8 1.0 1.0 1.0
## [52] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

```
df<-data.frame(class=knn1, probs=attr(knn_p, "prob"))
head(df)
```

```
##        class probs
## 1 pedestrian   0.6
## 2 pedestrian   0.6
## 3 pedestrian   0.8
## 4       stop   0.4
## 5 pedestrian   1.0
## 6 pedestrian   0.6
```

Predciting probabilities

These are the predicted probabilities.
Case 1: there were 5 neighbors, 3 of them were pedestrian sign so we predict the case as being pedestrian with the probability of 0.6 (3/6)

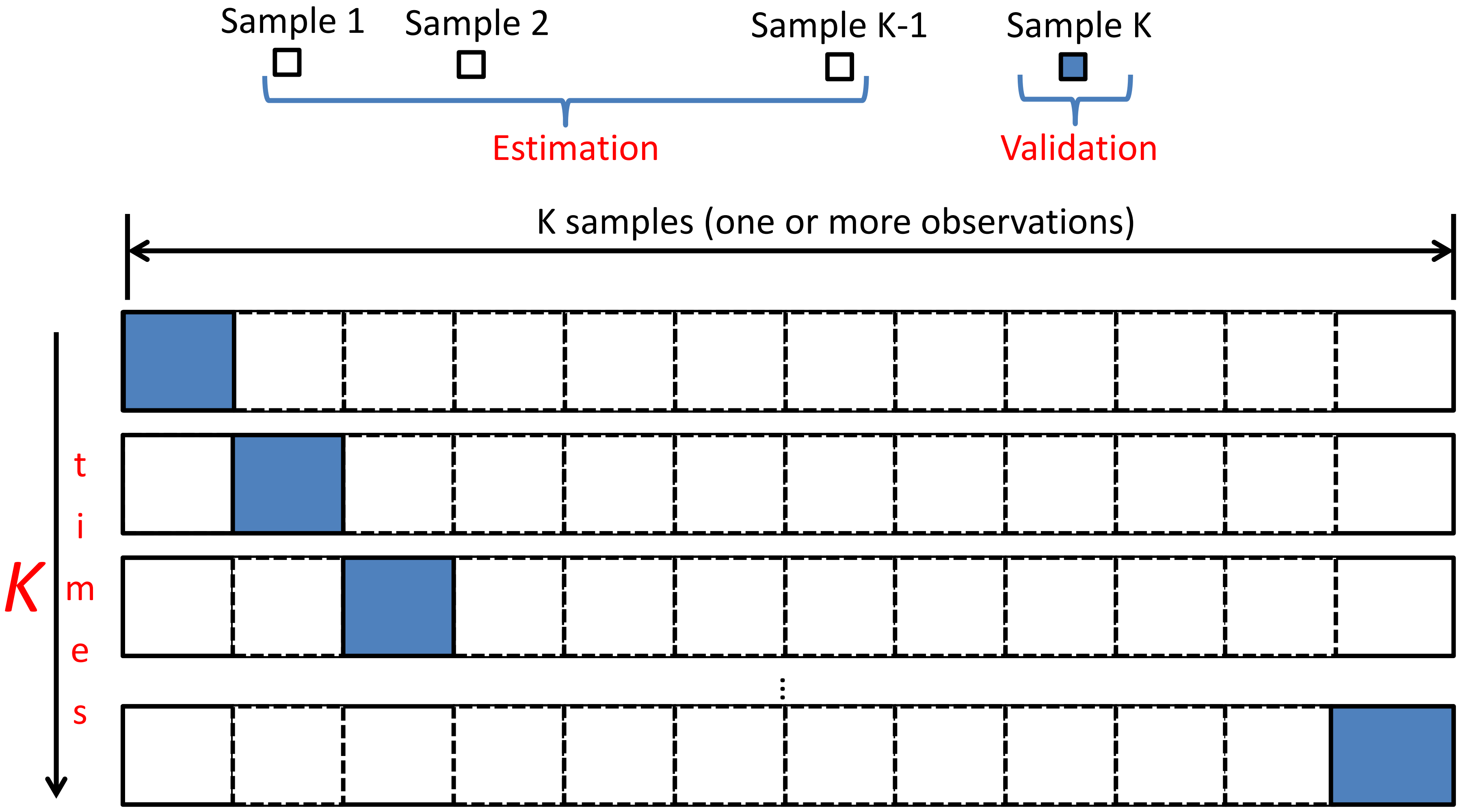
```
attr(knn_p, "prob")
```

```
##  [1] 0.6 0.6 0.8 0.4 1.0 0.6 0.8 1.0 0.6 1.0 1.0 1.0 0.8 1.0 1.0 1.0 1.0
## [18] 1.0 0.8 0.8 0.6 1.0 1.0 1.0 1.0 1.0 1.0 0.6 0.4 1.0 1.0 1.0 1.0 1.0
## [35] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 1.0 0.8 1.0 1.0 1.0
## [52] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

```
df<-data.frame(class=knn1, probs=attr(knn_p, "prob"))
head(df)
```

```
##        class probs
## 1 pedestrian   0.6
## 2 pedestrian   0.6
## 3 pedestrian   0.8
## 4       stop   0.4
## 5 pedestrian   1.0
## 6 pedestrian   0.6
```

How to chose K ?: Cross validation



Doing in R

- we will use package caret for knn
- caret has vast functionality of training different models

- read the dataframe Diabetes.csv

- Our goal is to predict if someone has a diabetes (Positive case)

- trainControl creates object that will be used in modeling. In this case we specify that we want to do cross-validation (method="cv") with 10 folds (number=10).
- Don't forget to set seed

```r
diab<-read.csv("Diabetes.csv")
```

```r
library(caret)
```

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

```r
set.seed(1)
ctrl<-trainControl(method="cv", number=10)
```

Doing it in R

function train from caret

- give the formula
- give the data
- set the method for training (method=“knn)
- trControl specifies the training control object
- prePreprocess: calculate the Z-scores
- tuneLength: for how many parameters of k do you want to build model ?

```
set.seed(1)
ctrl<-trainControl(method="cv", number=10)
```

```
knn_c<-train(Class~., data=diab, method="knn",
trControl=ctrl, preProcess = c("center","scale"), tuneLength=10)
```

Doing it in R

- The column k shows the number of neighbours that are used during the training
- Column Accuracy shows the average accuracy on the testing sets (total 10 testing sets)
- Overall 10 different values for k are used for modeling

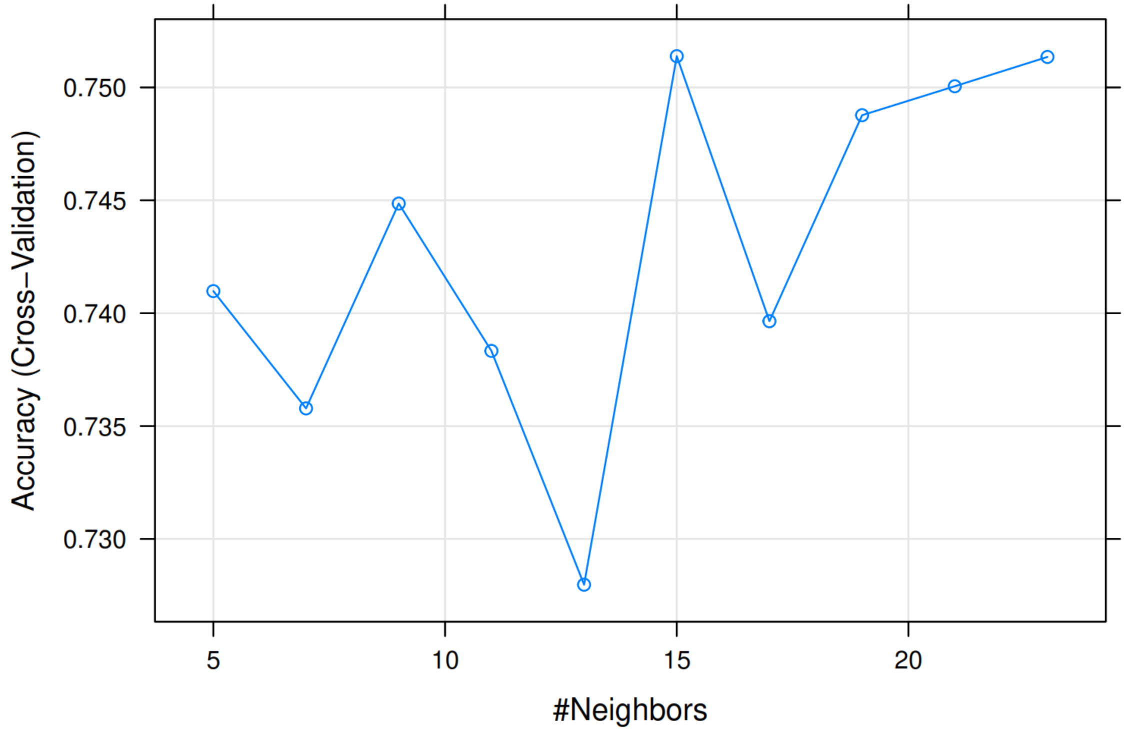
```r
knn_c$results
```

```
##     k  Accuracy     Kappa AccuracySD    KappaSD
## 1   5 0.7409774 0.4052329 0.02682073 0.06478774
## 2   7 0.7357826 0.3957593 0.03217106 0.07401431
## 3   9 0.7448565 0.4126696 0.03740766 0.08810021
## 4  11 0.7383288 0.3945516 0.04445408 0.10681977
## 5  13 0.7279733 0.3716559 0.04106226 0.09055321
## 6  15 0.7513841 0.4234201 0.03744945 0.08609421
## 7  17 0.7396446 0.3937532 0.03679603 0.08056096
## 8  19 0.7487697 0.4140229 0.04292794 0.09816277
## 9  21 0.7500513 0.4133438 0.03692821 0.09148527
## 10 23 0.7513500 0.4206874 0.03617822 0.08450982
```

Doing it in R

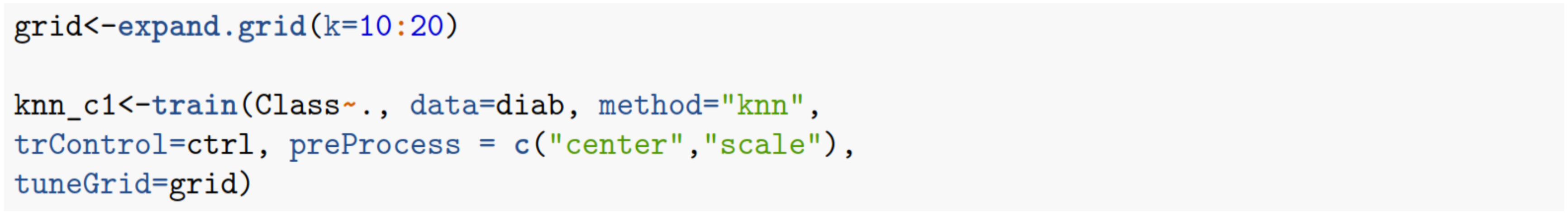
plot to visually see which value of k is the best
The best value for k is 15

```
plot(knn_c)
```



Doing it in R

- If you want to specify your vectors of k's for the model, use **tuneGrid** instead of **tuneLenght**

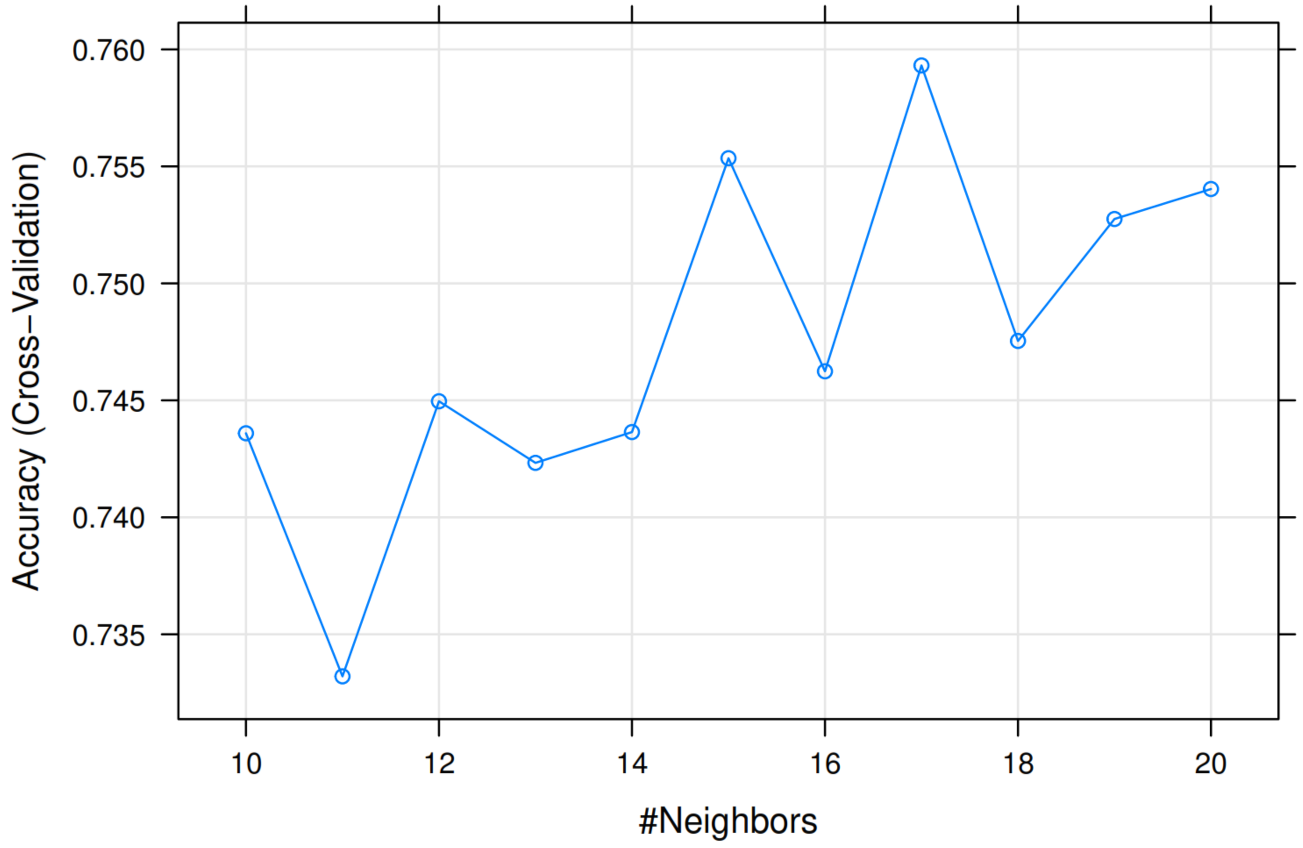


```
grid<-expand.grid(k=10:20)

knn_c1<-train(Class~., data=diab, method="knn",
trControl=ctrl, preProcess = c("center","scale"),
tuneGrid=grid)
```

Doing it in R

```r
plot(knn_c1)
```



Advantages

- Simple
- No assumptions required about Normal distribution, etc.
- Effective at capturing complex interactions among variables without having to define a statistical model

Disatvantages

- Required size of training set increases exponentially with # of predictors, *p*

  This is because expected distance to nearest neighbor increases with *p* (with large vector of predictors, all records end up “far away” from each other)
- In a large training set, it takes a long time to find distances to all the neighbors and then identify the nearest one(s)
- These constitute “curse of dimensionality”