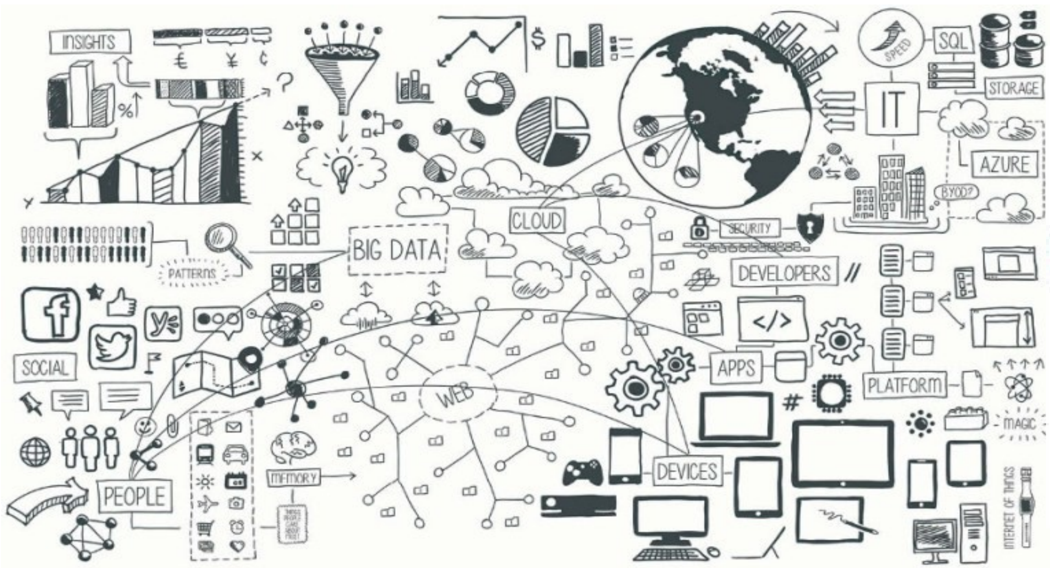
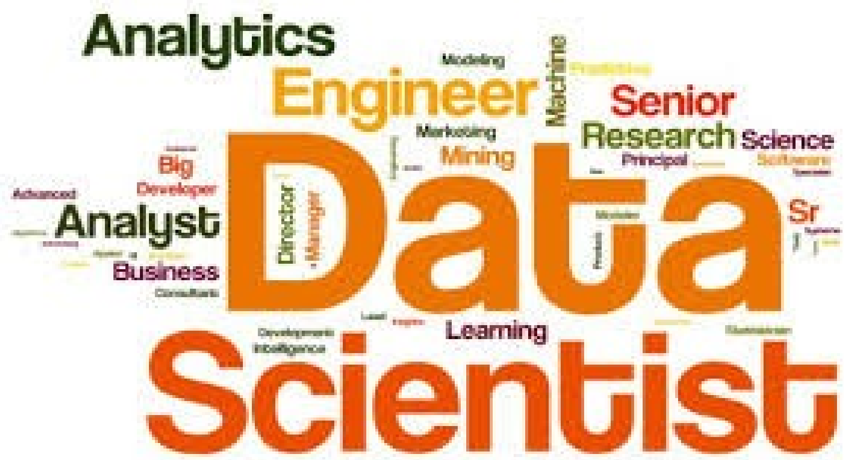
Data Mining (Minería de Datos)

The k-NN technique



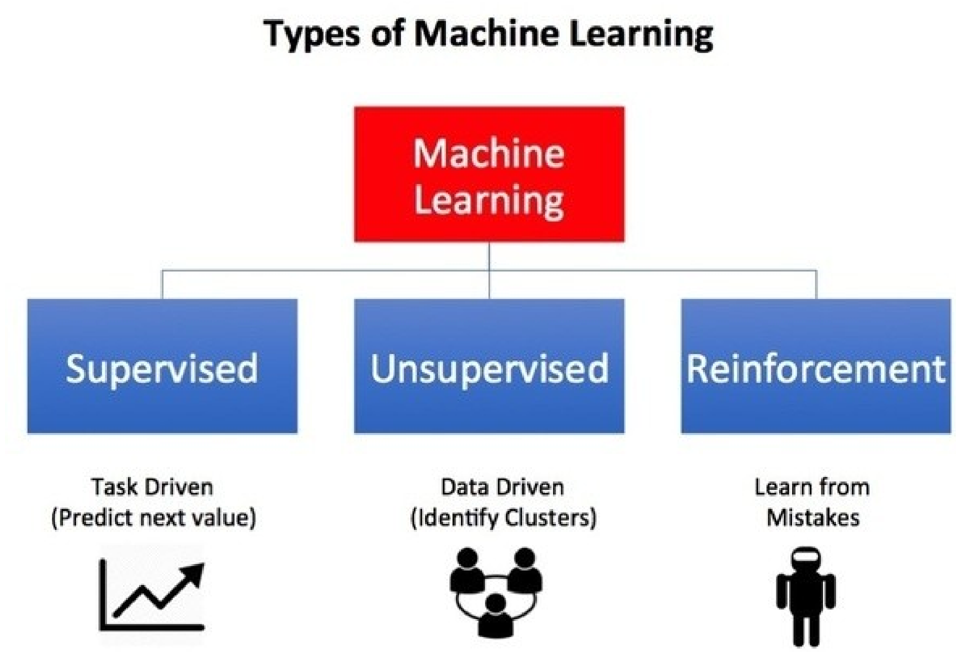


Rodrigo Manzanas

Joaquín Bedia

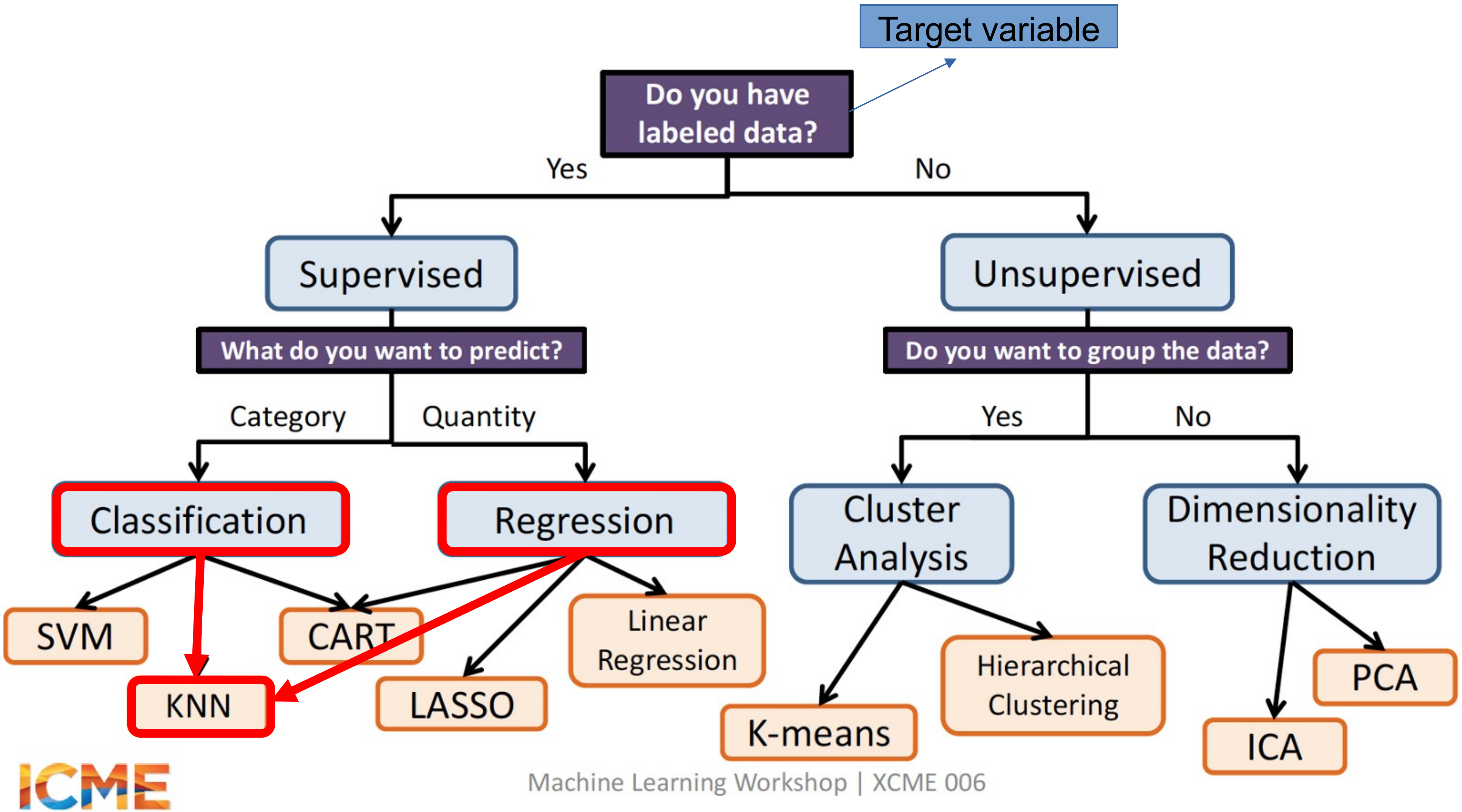
Grupo de Meteorología

Univ. de Cantabria – CSIC
MACC / IFCA



NOTA: Las líneas de código R en esta presentación se muestran sobre un fondo gris

Nov	2	Presentación, introducción y perspectiva histórica
	4	Paradigmas, problemas canonicos y data challenges
	9	Reglas de asociación
	11	Practica: Reglas de asociación
	16	Evaluación, sobrejuste y crossvalidacion
	18	Practica: Crossvalidacion
	23	Árboles de clasificacion y decision
	25	Practica: Árboles de clasificación
		T01. Datos discretos
	30	**Técnicas de vecinos cercano (k-NN)**
Dic	**2**	**Práctica: Vecinos cercanos**
	9	Comparación de Técnicas de Clasificación.
	14	Reducción de dimensión no lineal
	16	Reducción de dimensión no lineal
		T02. Clasificación
	17	Árboles de clasificación y regresion (CART)
	20	Práctica: Árboles de clasificación y regresion (CART)
	21	Practica: El paquete CARET
		T03. Prediccion
Ene	11	Ensembles: Bagging and Boosting
	13	Random Forests y **Gradient boosting**
	14	**Técnicas de agrupamiento**
	20	**Técnicas de agrupamiento**
	24	Predicción Condicionada
	26	Sesión de refuerzo/repaso.
Ene	27	Examen



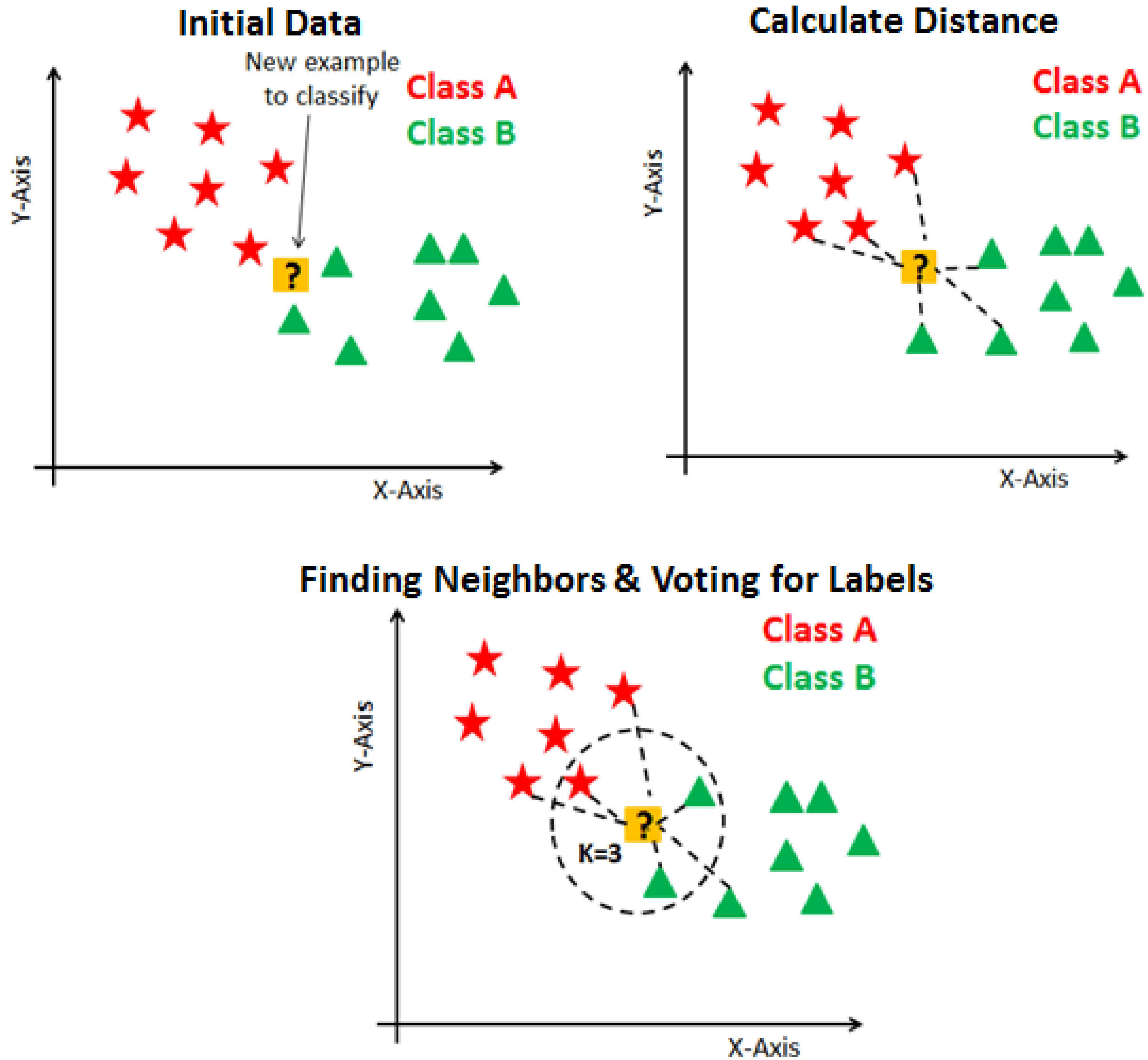
Target variable
Do you have labeled data?
Yes
No
Supervised
Unsupervised
What do you want to predict?
Do you want to group the data?
Category
Quantity
Yes
No
Classification
Regression
Cluster Analysis
Dimensionality Reduction
SVM
CART
KNN
LASSO
Linear Regression
K-means
Hierarchical Clustering
PCA
ICA
Machine Learning Workshop | XCME 006

How does it work?

"… dime con quién vas y te diré quién eres …"

Non-parametric:
No assumption is made on the underlying data distribution

Lazy (or instance-based) learning:
There is no explicit training phase. All the training data is needed during the testing phase



How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
…					
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
…					
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
...					
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
...					
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica
151	5.4	2.7	4.6	1.4	?

New instance to be classified

How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width		Species
1	5.1	3.5	1.4	0.2		setosa
2	4.9	3.0	1.4	0.2		setosa
3	4.7	3.2	1.3	0.2		setosa
…						
55	6.5	2.8	4.6	1.5		versicolor
56	5.7	2.8	4.5	1.3		versicolor
57	6.3	3.3	4.7	1.6		versicolor
…						
148	6.5	3.0	5.2	2.0		virginica
149	6.2	3.4	5.4	2.3		virginica
150	5.9	3.0	5.1	1.8		virginica
151	5.4	2.7	4.6	1.4		?

New instance to be classified

STEP1: Computing distances

$$d_{151,1}=\sqrt{(5.4-5.1)^2+(2.7-3.5)^2+(4.6-1.4)^2+(1.4-0.2)^2}=3.52$$

How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
…					
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
…					
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica
151	5.4	2.7	4.6	1.4	?

New instance to be classified

STEP1: Computing distances

$$d_{151,1} = \sqrt{(5.4-5.1)^2 + (2.7-3.5)^2 + (4.6-1.4)^2 + (1.4-0.2)^2} = 3.52$$

$$d_{151,2} = 3.47$$
$$d_{151,3} = 3.62$$
$$\ldots$$
$$d_{151,55} = 1.11$$
$$d_{151,56} = 0.35$$
$$d_{151,57} = 1.10$$
$$\ldots$$
$$d_{151,148} = 1.42$$
$$d_{151,149} = 1.61$$
$$d_{151,150} = 0.87$$

How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
…					
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
…					
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica
151	5.4	2.7	4.6	1.4	?

New instance to be classified

STEP1: Computing distances

$$d_{151,1} = \sqrt{(5.4-5.1)^2 + (2.7-3.5)^2 + (4.6-1.4)^2 + (1.4-0.2)^2} = 3.52$$

$d_{151,2} = 3.47$
$d_{151,3} = 3.62$
…
$d_{151,55} = 1.11$
$d_{151,56} = 0.35$
$d_{151,57} = 1.10$
…
$d_{151,148} = 1.42$
$d_{151,149} = 1.61$
$d_{151,150} = 0.87$

STEP2: Ordering distances

$d_{151,56} = 0.35$
$d_{151,150} = 0.87$
$d_{151,57} = 1.10$
$d_{151,55} = 1.11$
$d_{151,148} = 1.42$
$d_{151,149} = 1.61$
$d_{151,2} = 3.47$
$d_{151,1} = 3.52$
$d_{151,3} = 3.62$

How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
...					
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
...					
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica
151	5.4	2.7	4.6	1.4	?

New instance to be classified

STEP1: Computing distances

$$d_{151,1}=\sqrt{(5.4-5.1)^2+(2.7-3.5)^2+(4.6-1.4)^2+(1.4-0.2)^2}=3.52$$

$d_{151,2}=3.47$
$d_{151,3}=3.62$
...
$d_{151,55}=1.11$
$d_{151,56}=0.35$
$d_{151,57}=1.10$
...
$d_{151,148}=1.42$
$d_{151,149}=1.61$
$d_{151,150}=0.87$

STEP2: Ordering distances

$\boxed{d_{151,56}=0.35}$ $\xrightarrow{k=1}$ *151 = Versicolor*
$\boxed{d_{151,150}=0.87}$ $\xrightarrow{k=2}$ *151 = Virginica*
$\boxed{d_{151,57}=1.10}$ $\xrightarrow{k=3}$ *151 = Versicolor*
$d_{151,55}=1.11$
$d_{151,148}=1.42$
$d_{151,149}=1.61$
$d_{151,2}=3.47$
$d_{151,1}=3.52$
$d_{151,3}=3.62$

STEP3: NN-based classification

How does it work?

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
…					
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
…					
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica
151	5.4	2.7	4.6	1.4	?

STEP1: Computing distances

$$d_{151,1}=\sqrt{(5.4-5.1)^2+(2.7-3.5)^2+(4.6-1.4)^2+(1.4-0.2)^2}=3.52$$

$d_{151,2}=3.47$

$d_{151,3}=3.62$

…

$d_{151,55}=1.11$

$d_{151,56}=0.35$

$d_{151,57}=1.10$

…

$d_{151,148}=1.42$

$d_{151,149}=1.61$

$d_{151,150}=0.87$

STEP2: Ordering distances

$\boxed{d_{151,56}=0.35}$ $\xrightarrow{k=1}$ *151 = Versicolor*

$\boxed{d_{151,150}=0.87}$ $\xrightarrow{k=2}$ *151 = virginica*

$\boxed{d_{151,57}=1.10}$ $\xrightarrow{k=3}$ *151 = Versicolor*

$d_{151,55}=1.11$

$d_{151,148}=1.42$

$d_{151,149}=1.61$

$d_{151,2}=3.47$

$d_{151,1}=3.52$

$d_{151,3}=3.62$

STEP3: NN-based classification

Pros:

- Easy to understand
- Versatile: Classification and regression problems
- High accuracy (benchmark method)

Cons:

- High memory requirements, computationally expensive
- Sensitive to scale of the data
- Can suffer from biases towards skewed distributions
- Performance can be severely degraded in high dimensional problems

Applications:

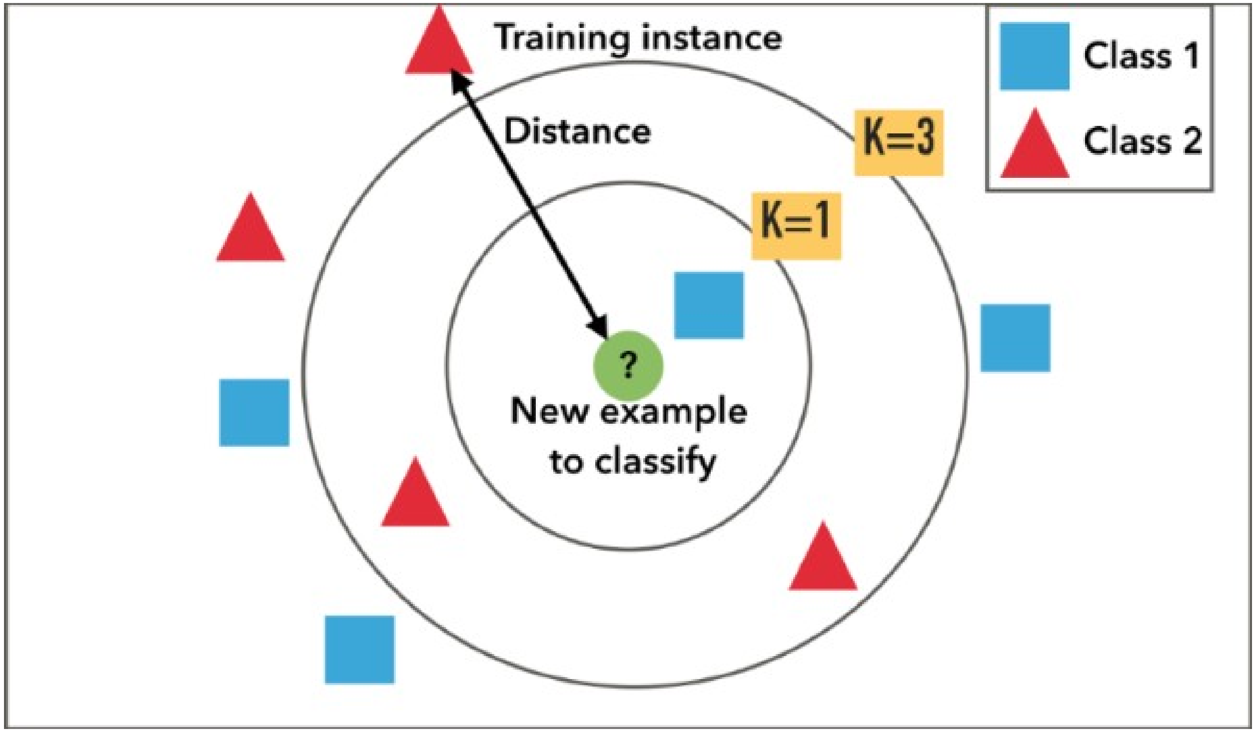
- Economic sciences: concession of loans
- Political sciences: classifying potential voters
- Handwriting detection (e.g. OCR)
- Image/video recognition
- Genetics

Fitting the method

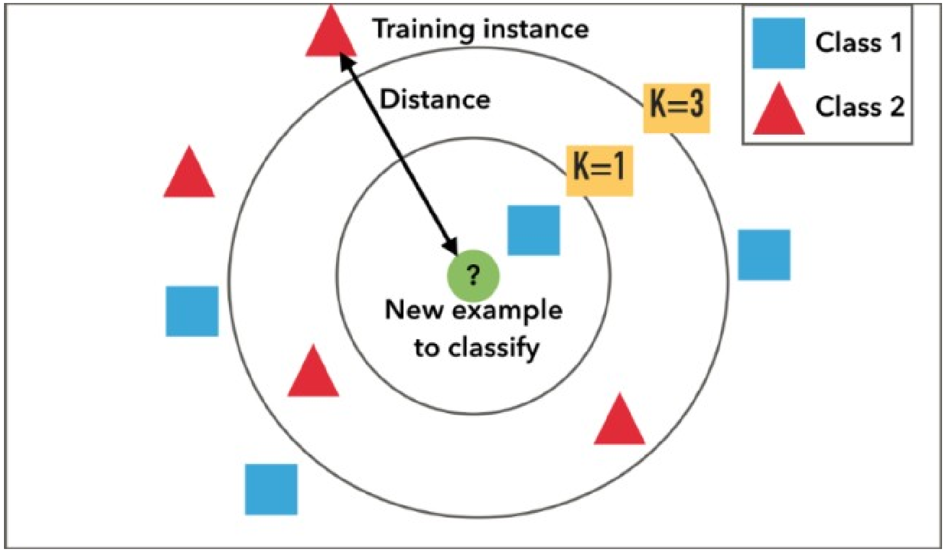
Distance metric
Different distances are used, depending on the application. *Euclidean* is the most common

Number of neighbors (*k*)
This is the unique model parameter. Must be properly chosen

Classifying criterion
- Majority vote
- Weighted vote
- Random
- etc.



Distance metric



Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

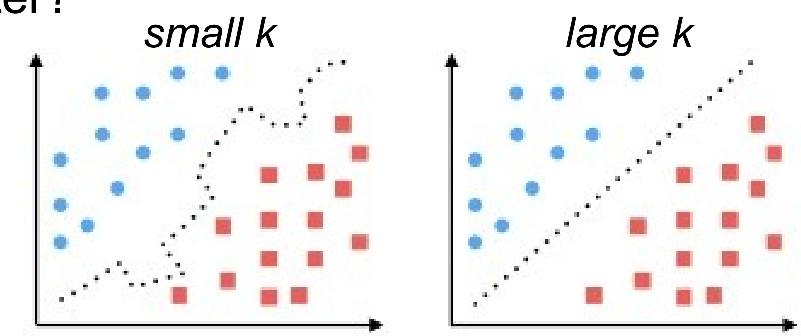
$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Number of neighbors (*k*)

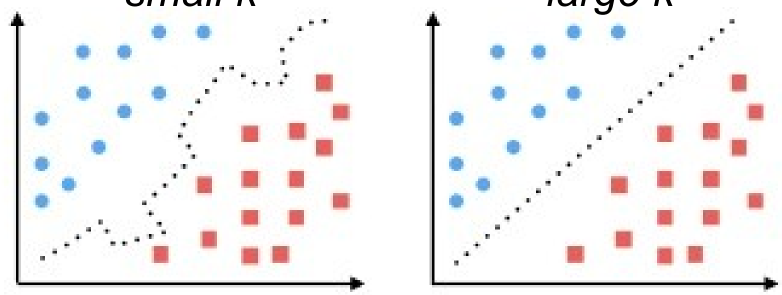
Which one classifies better?



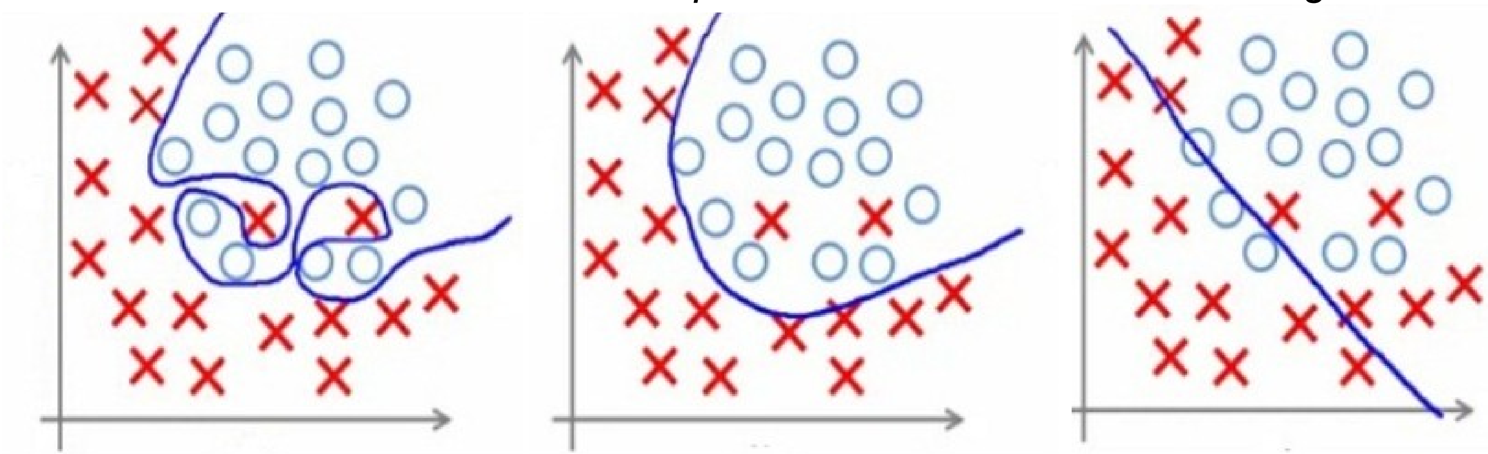
Number of neighbors (*k*)

Which one classifies better?

small k *large k*



too small k *optimum k* *too large k*



Over-fitting

(forcefitting -- too good to be true)

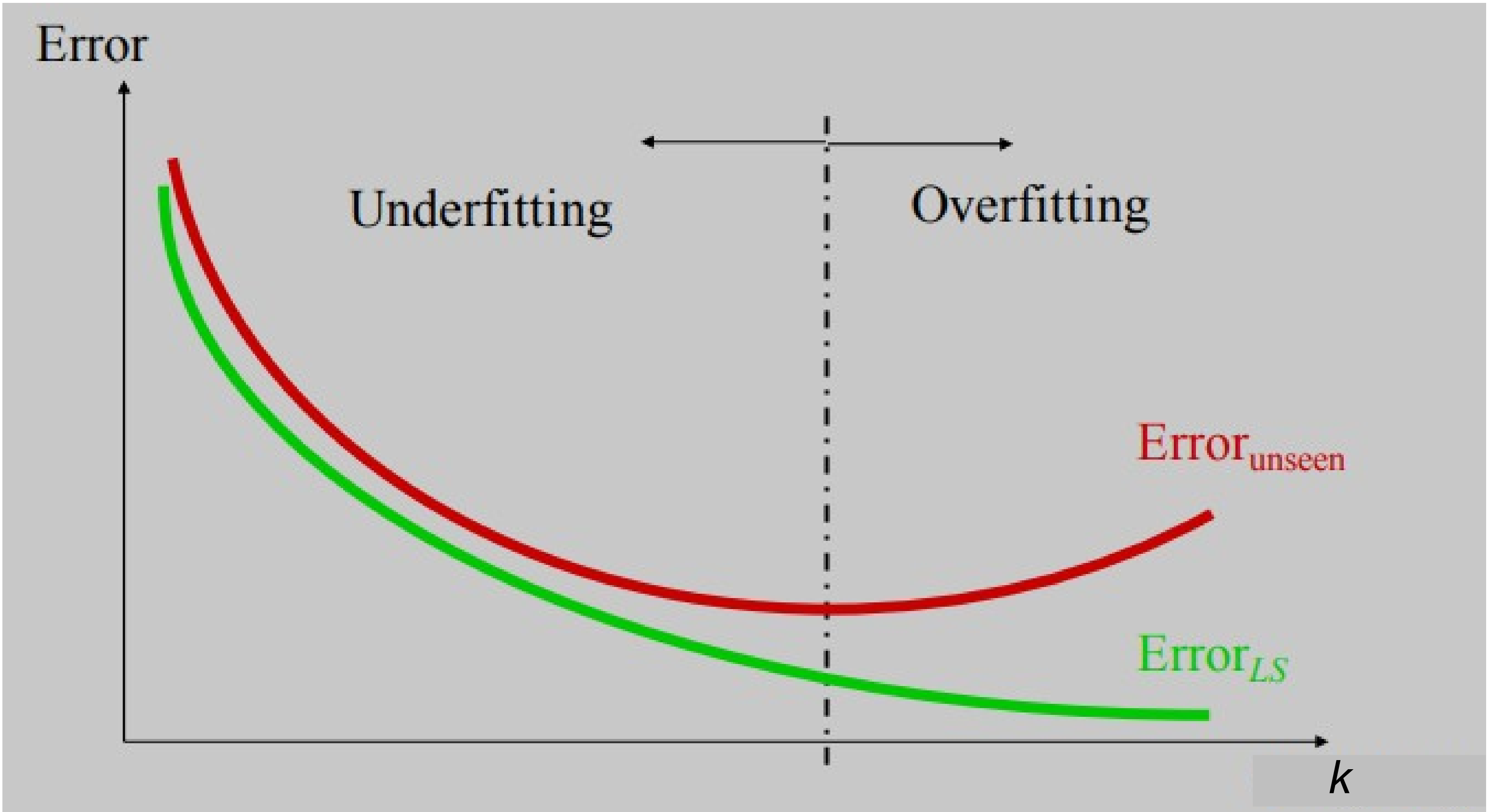
Appropriate-fitting

Under-fitting

(too simple to explain the variance)

Number of neighbors (*k*)

Cross-validation is needed to find the optimal *k*



Examples in R

Based on the iris dataset, classify the following new instance:
(sepal l., sepal w., petal l., petal w.) = (5.4, 2.7, 4.6, 1.4)

```
# new instance
d.new = c(5.4, 2.7, 4.6, 1.4)
```

Examples in R

Based on the iris dataset, classify the following new instance:
(sepal l., sepal w., petal l., petal w.) = (5.4, 2.7, 4.6, 1.4)

```
# new instance
d.new = c(5.4, 2.7, 4.6, 1.4)
```

```
# euclidean distance between the new instance and all the others
eucli = c()
for (i in 1:nrow(iris)) {
  eucli[i] = sqrt(sum((d.new - iris[i,-5])^2))
}
```

Examples in R

<u>Based on the iris dataset, classify the following new instance:</u>
(sepal l., sepal w., petal l., petal w.) = (5.4, 2.7, 4.6, 1.4)

```
# new instance
d.new = c(5.4, 2.7, 4.6, 1.4)
```

```
# euclidean distance between the new instance and all the others
eucli = c()
for (i in 1:nrow(iris)) {
  eucli[i] = sqrt(sum((d.new - iris[i,-5])^2))
}
```

```
## ordering distances
ind.sort = sort(eucli, index.return = T)
```

Examples in R

<u>Based on the iris dataset, classify the following new instance:</u>
(sepal l., sepal w., petal l., petal w.) = (5.4, 2.7, 4.6, 1.4)

```
# new instance
d.new = c(5.4, 2.7, 4.6, 1.4)
```

```
# euclidean distance between the new instance and all the others
eucli = c()
for (i in 1:nrow(iris)) {
  eucli[i] = sqrt(sum((d.new - iris[i,-5])^2))
}
```

```
## ordering distances
ind.sort = sort(eucli, index.return = T)
```

```
# classifying based on the nearest neighbor
pred.k1 = iris$Species[ind.sort$ix[1]]
```

Examples in R

Based on the iris dataset, classify the following new instance:
(sepal l., sepal w., petal l., petal w.) = (5.4, 2.7, 4.6, 1.4)

```
# new instance
d.new = c(5.4, 2.7, 4.6, 1.4)
```

```
# euclidean distance between the new instance and all the others
eucli = c()
for (i in 1:nrow(iris)) {
  eucli[i] = sqrt(sum((d.new - iris[i,-5])^2))
}
```

```
## ordering distances
ind.sort = sort(eucli, index.return = T)
```

```
# classifying based on the nearest neighbor
pred.k1 = iris$Species[ind.sort$ix[1]]
```

```
# classifying based on the 10 nearest neighbors
pred.k10 = iris$Species[ind.sort$ix[1:10]]
summary(pred.k10)
```

Examples in R

Divide iris into train y test (75% and 25% of the total dataset, respectively) and find the test error for the nearest neighbor method. Use the function "knn" from package "class"

Examples in R

Divide iris into train y test (75% and 25% of the total dataset, respectively) and find the test error for the nearest neighbor method. Use the function "knn" from package "class"

```
# train/test division
n = nrow(iris)
indtrain = sample(1:n, round(0.75*n))
indtest = setdiff(1:n, indtrain)
iris.train = iris[indtrain,]
iris.test = iris[indtest,]
```

Examples in R

Divide iris into train y test (75% and 25% of the total dataset, respectively) and find the test error for the nearest neighbor method. Use the function “knn” from package “class”

```
# train/test division
n = nrow(iris)
indtrain = sample(1:n, round(0.75*n))
indtest = setdiff(1:n, indtrain)
iris.train = iris[indtrain,]
iris.test = iris[indtest,]
```

```
# classifying using the nearest neighbor method
library(class)
pred = knn(train = iris.train[,-5], test = iris.test[,-5], cl = iris.train$Species, k = 1)
```

Examples in R

Divide iris into train y test (75% and 25% of the total dataset, respectively) and find the test error for the nearest neighbor method. Use the function "knn" from package "class"

```
# train/test division
n = nrow(iris)
indtrain = sample(1:n, round(0.75*n))
indtest = setdiff(1:n, indtrain)
iris.train = iris[indtrain,]
iris.test = iris[indtest,]
```

```
# classifying using the nearest neighbor method
library(class)
pred = knn(train = iris.train[,-5], test = iris.test[,-5], cl = iris.train$Species, k = 1)
```

```
# validating method
table(pred, iris.test$Species)
pred          setosa versicolor virginica
  setosa        11          0         0
  versicolor     0         14         1
  virginica      0          1        11
acc.class(pred, iris.test$Species)
```

```
# evaluation function
acc.class = function(x, y) {
  stopifnot(length(x) == length(y))
  return(sum(diag(table(x, y))) / length(x))
}
```

Examples in R

Use the package "caret" (method "knn") to find the optimal k. To do so, check how the test error varies with increasing k (for values from 1 to 50) under a hold-out cross-validation scheme.

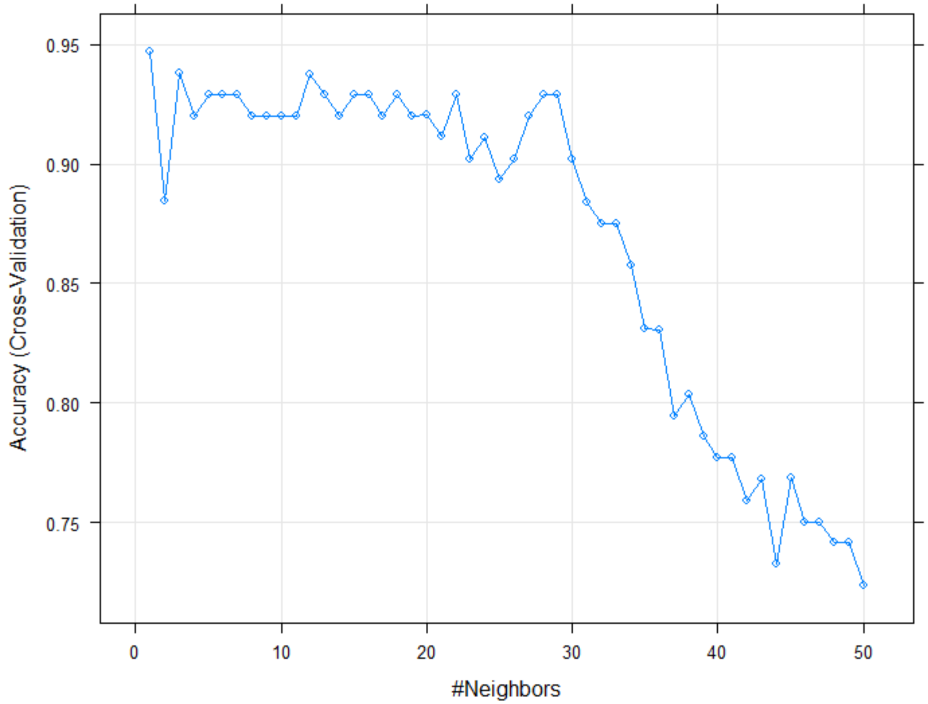
```
library(caret)
# defining hold-out cross-validation
trctrl = trainControl(method = "cv", number = 2)
# searching the optimal k
knn.fit = train(Species ~ ., iris.train,
                method = "knn",
                trControl = trctrl,
                tuneGrid = expand.grid(k = 1:50))
plot(knn.fit)
```

)

Examples in R

Use the package "caret" (method "knn") to find the optimal k. To do so, check how the test error varies with increasing k (for values from 1 to 50) under a hold-out cross-validation scheme.

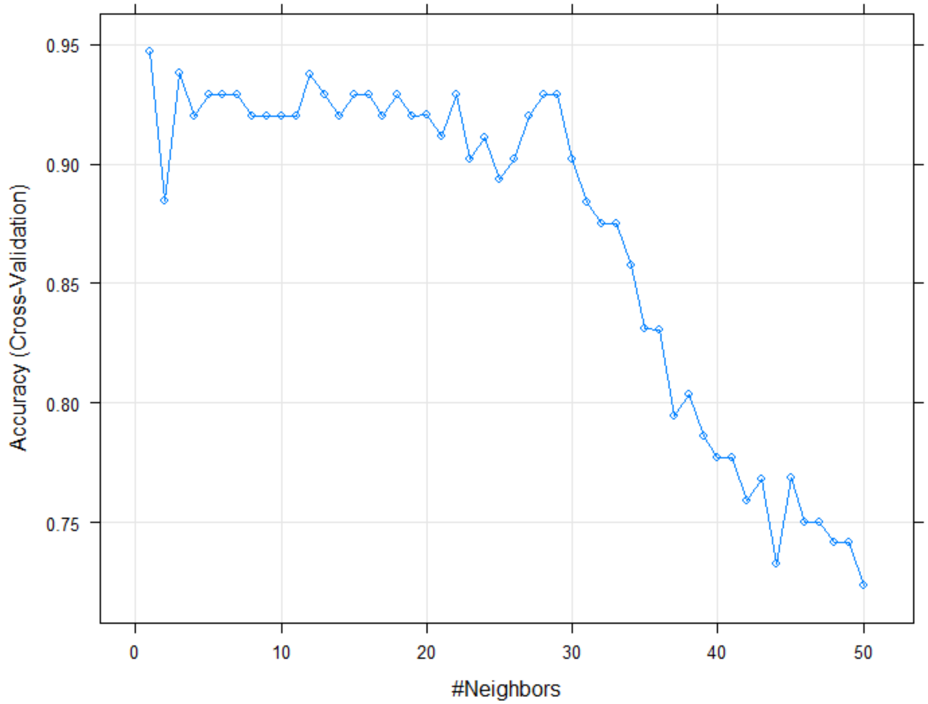
```
library(caret)
# defining hold-out cross-validation
trctrl = trainControl(method = "cv", number = 2)
# searching the optimal k
knn.fit = train(Species ~ ., iris.train,
                method = "knn",
                trControl = trctrl,
                tuneGrid = expand.grid(k = 1:50))
plot(knn.fit)
```



Examples in R

Use the package “caret” (method “knn”) to find the optimal k. To do so, check how the test error varies with increasing k (for values from 1 to 50) under a hold-out cross-validation scheme.

```
library(caret)
# defining hold-out cross-validation
trctrl = trainControl(method = "cv", number = 2)
# searching the optimal k
knn.fit = train(Species ~ ., iris.train,
                method = "knn",
                trControl = trctrl,
                tuneGrid = expand.grid(k = 1:50))
plot(knn.fit)
```




```
# predicting in test with the optimal k
pred = predict(knn.fit, iris.test)
acc = acc.class(pred, iris.test$Species)
```

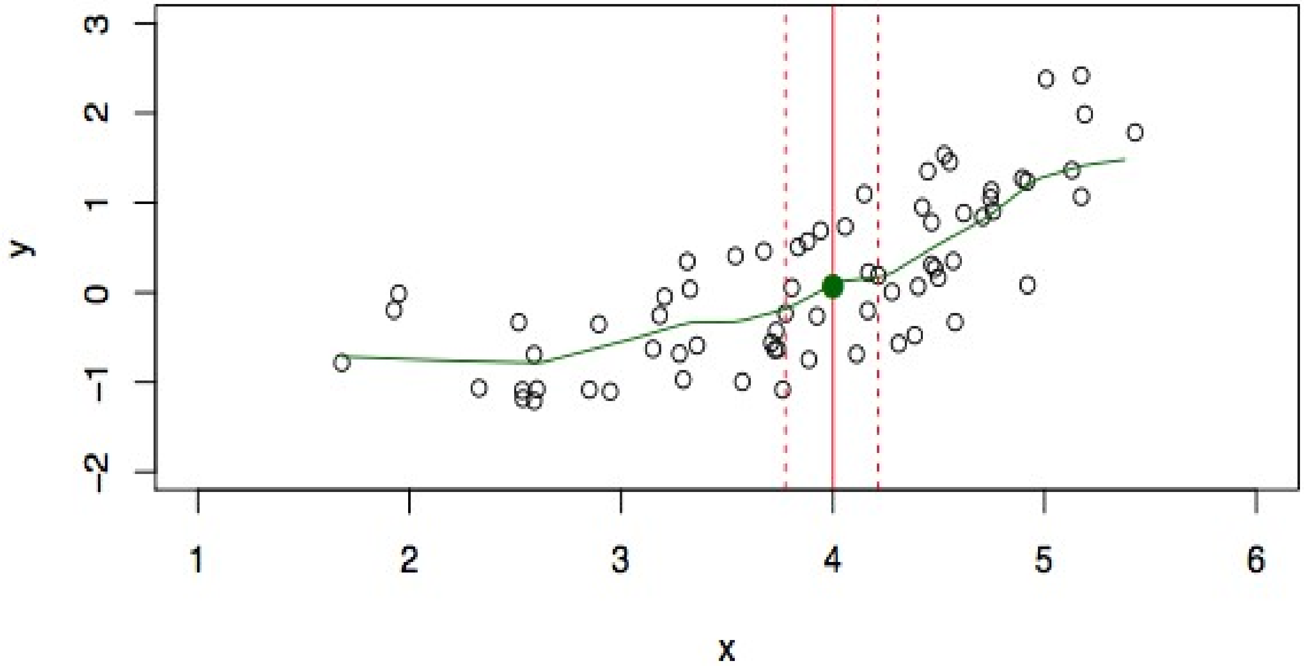
k-NN for regression

Aim:
Predicting a continuous target variable

What do we need?
An inference criterion: It can be a simple mean, a particular percentile, etc.

To take into account:
Predictor variables covering larger ranges may have more weight in the search of neighbours. Rescaling the predictor data is recommended to make the distance metric more meaningful

$$Z = \frac{X - \mu}{\sigma}$$



	Ozone	Solar.R	Wind	Temp
1	41	190	7.4	67
2	36	118	8.0	72
3	12	149	12.6	74
4	18	313	11.5	62
5	25	297	14.3	56
…				
500	23	234	9.3	65
501	45	321	16.7	?

Examples in R

For regression, we will work with the dataset "carseats" (included in the package "ISLR"). Our target variable will be "Sales". First, we will remove the all the categorical variables from the dataset, retaining only the continuous ones. We will use the function "knn.reg" from the package "FNN". As you did for the case of classification, divide the total dataset in 75% for train and 25% for test and see how the test error varies with k

Examples in R

For regression, we will work with the dataset "carseats" (included in the package "ISLR"). Our target variable will be "Sales". First, we will remove the all the categorical variables from the dataset, retaining only the continuous ones. We will use the function "knn.reg" from the package "FNN". As you did for the case of classification, divide the total dataset in 75% for train and 25% for test and see how the test error varies with k

```
library(ISLR)
attach(Carseats)
dataset = Carseats[, -c(7,10,11)]

# evaluation function
rmse <- function(x, y) {
  sqrt(mean((x - y)^2))
}

# train/test division
n = nrow(dataset)
indtrain = sample(1:n, round(0.75*n));
dataset.train = dataset[indtrain, ]
indtest = setdiff(1:n, indtrain);
dataset.test = dataset[indtest, ]
```

Examples in R

For regression, we will work with the dataset "carseats" (included in the package "ISLR"). Our target variable will be "Sales". First, we will remove the all the categorical variables from the dataset, retaining only the continuous ones. We will use the function "knn.reg" from the package "FNN". As you did for the case of classification, divide the total dataset in 75% for train and 25% for test and see how the test error varies with k

```
library(ISLR)
attach(Carseats)
dataset = Carseats[, -c(7,10,11)]

# evaluation function
rmse <- function(x, y) {
  sqrt(mean((x - y)^2))
}

# train/test division
n = nrow(dataset)
indtrain = sample(1:n, round(0.75*n));
dataset.train = dataset[indtrain, ]
indtest = setdiff(1:n, indtrain);
dataset.test = dataset[indtest, ]
```

```
# test error as a function of k
library(FNN)
kmax = 50
test.err = c()
for (k in 1:kmax) {
  pred = knn.reg(dataset.train[,-1], dataset.test[,-1],
dataset.train$Sales, k = k)
  test.err[k] = rmse(pred$pred,
as.numeric(dataset.test$Sales))
}
plot(1:kmax, test.err, type = "o", pch = 19,
xlab = "k", ylab = "RMSE"); grid()
```

Examples in R

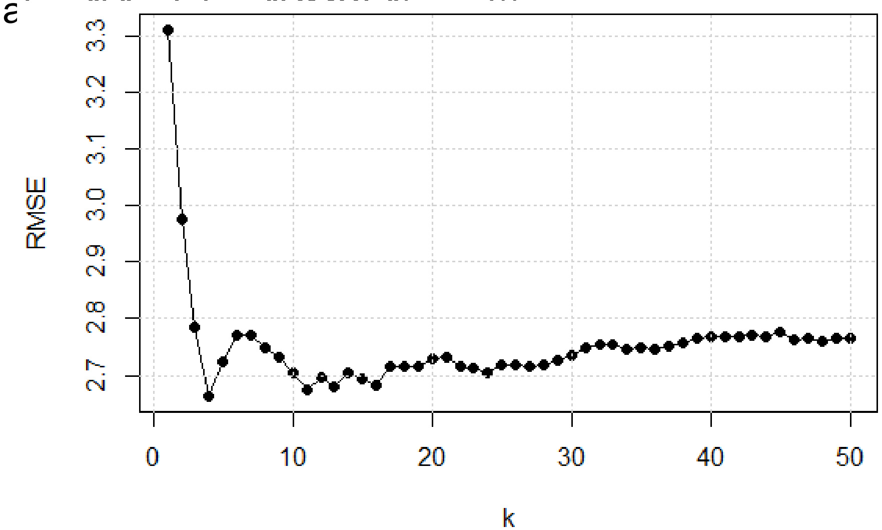
For regression, we will work with the dataset "carseats" (included in the package "ISLR"). Our target variable will be "Sales". First, we will remove the all the categorical variables from the dataset, retaining only the continuous ones. We will use the function "knn.reg" from the package "FNN". As you did for the case of classification, divide the total dataset in 75% for train and 25% for test and see how the test error varies with k

```
library(ISLR)
attach(Carseats)
dataset = Carseats[, -c(7,10,11)]

# evaluation function
rmse <- function(x, y) {
  sqrt(mean((x - y)^2))
}

# train/test division
n = nrow(dataset)
indtrain = sample(1:n, round(0.75*n));
dataset.train = dataset[indtrain, ]
indtest = setdiff(1:n, indtrain);
dataset.test = dataset[indtest, ]
```

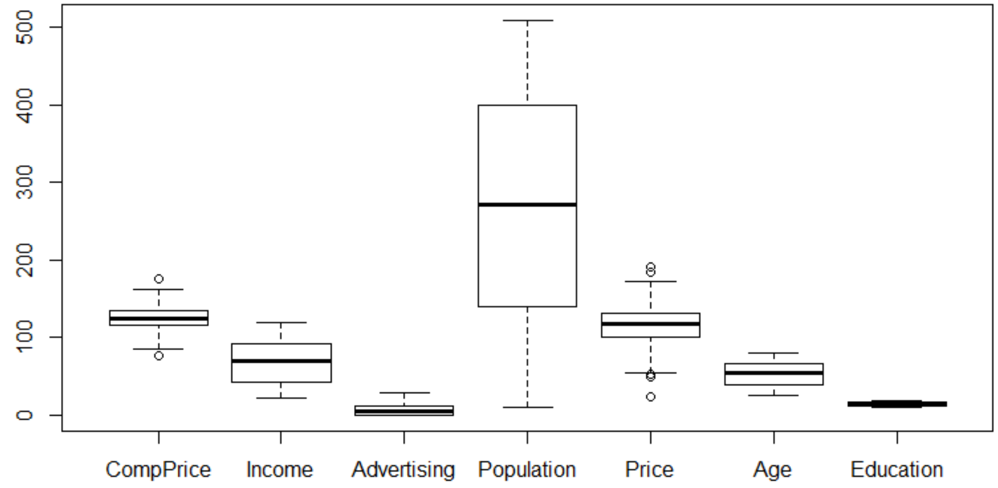
```
# test error as a function of k
library(FNN)
kmax = 50
test.err = c()
for (k in 1:kmax) {
  pred = knn.reg(dataset.train[,-1], dataset.test[,-1],
dataset.train$Sales, k = k)
  test.err[k] = rmse(pred$pred,
as.numeric(dataset.test$Sales))
}
plot(1:kmax, test.err, type = "o", pch = 19,
xla
```



Examples in R

<u>Let's continue by seeing the effect of properly rescaling the predictor data. Use the function "scale"</u>

```
# predictor ranges
boxplot(dataset[,-1])
```

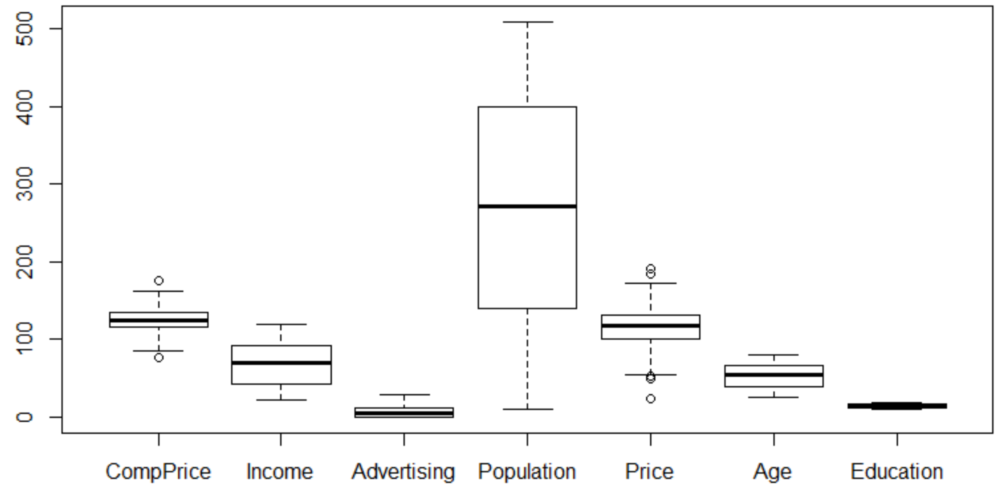


Examples in R

Let's continue by seeing the effect of properly rescaling the predictor data. Use the function "scale"

```
# predictor ranges
boxplot(dataset[,-1])
```

```
# test error as a function of k (for standardized data)
test.err2 = c()
for (k in 1:kmax) {
  pred = knn.reg(scale(dataset.train[,-1]),
scale(dataset.test[,-1]), dataset.train$Sales, k = k)
  test.err2[k] = rmse(pred$pred,
as.numeric(dataset.test$Sales))
}
```



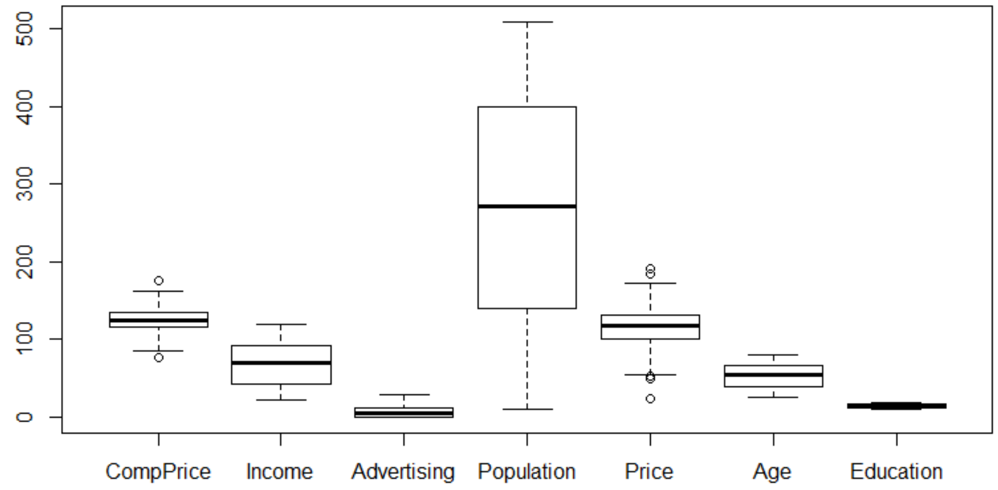
Examples in R

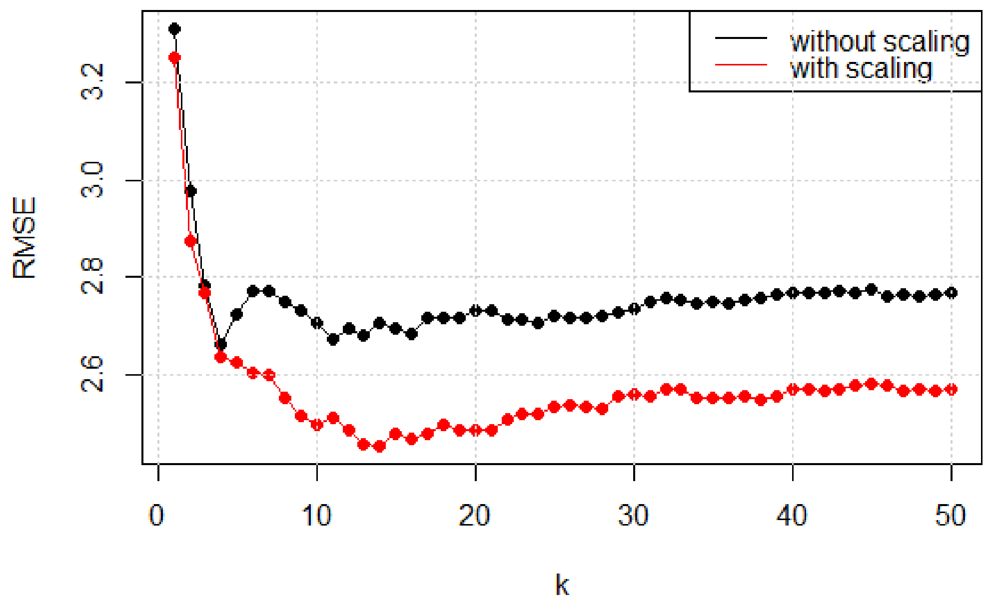
Let's continue by seeing the effect of properly rescaling the predictor data. Use the function "scale"

```
# predictor ranges
boxplot(dataset[,-1])
```

```
# test error as a function of k (for standardized data)
test.err2 = c()
for (k in 1:kmax) {
  pred = knn.reg(scale(dataset.train[,-1]),
scale(dataset.test[,-1]), dataset.train$Sales, k = k)
  test.err2[k] = rmse(pred$pred,
as.numeric(dataset.test$Sales))
}
```

```
# plotting results
matplot(1:kmax, cbind(test.err, test.err2),
        type = "o", pch = 19, lty = 1,
        col = c("black", "red"), xlab = "k", ylab = "RMSE")
legend("topright", c("without scaling", "with scaling"),
       lty = 1, col = c("black", "red"))
grid()
```





Examples in R

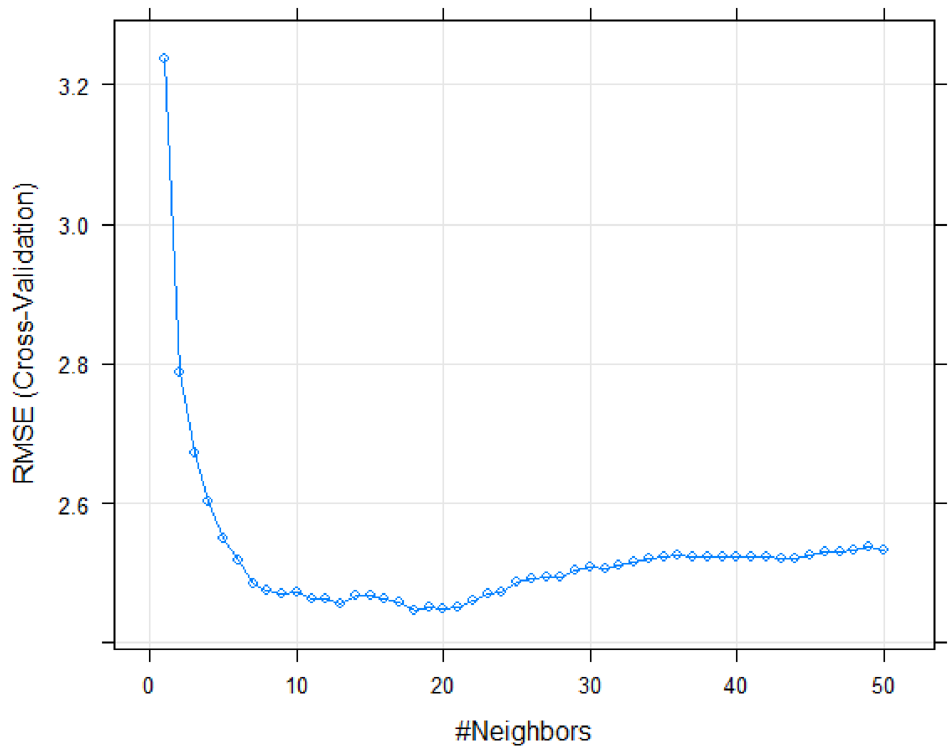
Do the same exercise, but this time using "caret". Recall to standardize your predictor data to obtain meaningful results.

Examples in R

Do the same exercise, but this time using “caret”. Recall to standardize your predictor data to obtain meaningful results

```
# defininig hold-out cross-validation
trctrl <- trainControl(method = "cv", number = 2)
# searching the optimal k (with standardized data)
knn.fit <- train(Sales ~ ., data = dataset.train,
                 method = "knn",
                 trControl = trctrl,
                 preProcess = c("center","scale"),
                 tuneGrid = expand.grid(k = 1:50))
plot(knn.fit)
```

```
# predicting in test with the optimal k
pred = predict(knn.fit, dataset.test)
rmse(pred, dataset.test$Sales)
```



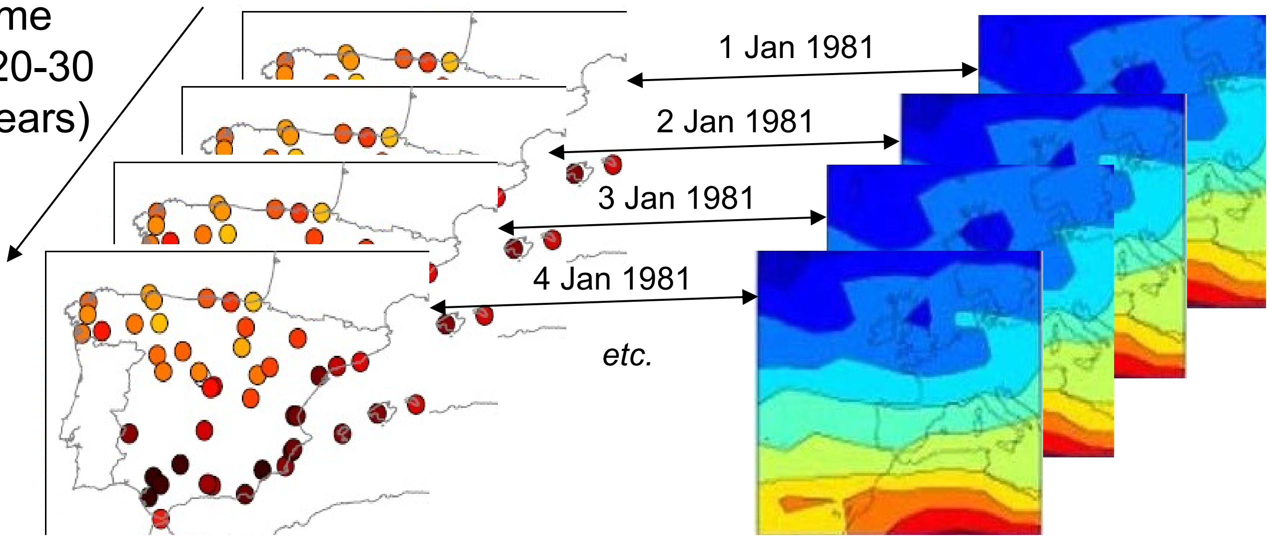
k-NN in meteorology

The analog technique (Lorenz, 1969): Similar atmospheric patterns lead to similar meteorological conditions

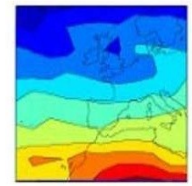
Problem: Y' (prediction)
for 26 Mar 2046?

Y: historical observations of the **target variable (e.g. precipitation)**

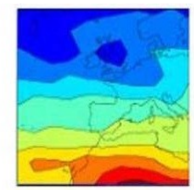
X: historical model simulations (large-scale atmospheric configurations)



1) Take X' for 26 Mar 2046: X_{2046}

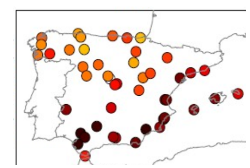


2) Search the nearest neighbor/s to X_{2046} within X



X (3-Jan-1981)

3) Infer a prediction based on the observed values in the days selected in 2)



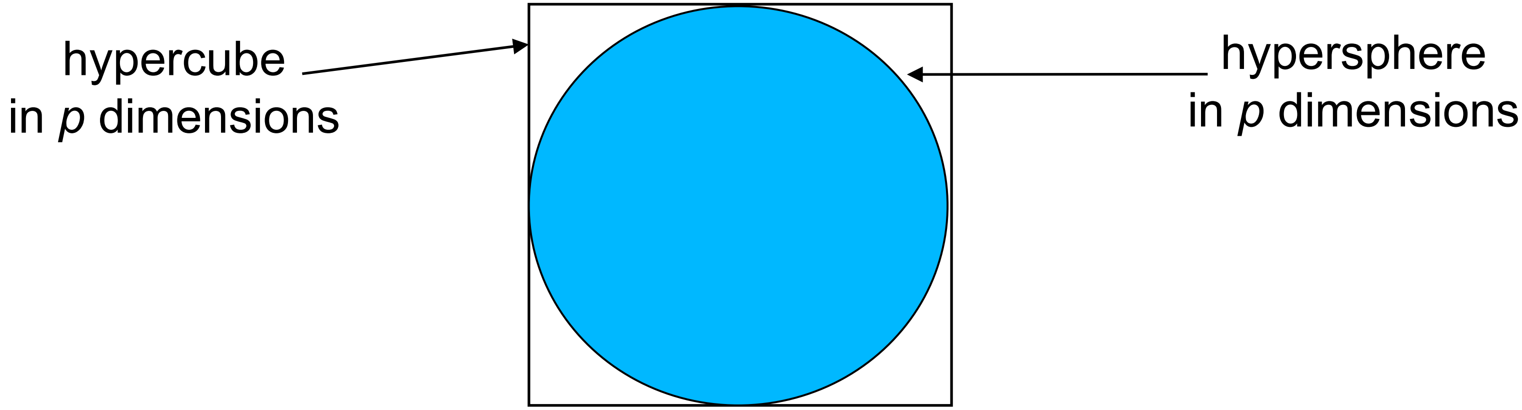
In meteorology, two important factors must be taken into account for the application of k-NN technique:

1) Predictor scaling
2) High dimensionality (the curse of dimensionality)

Also note that k-NN has no ability for extrapolation beyond the learning space –> Climate change

The curse of dimensionality

(David Scott, *Multivariate Density Estimation,* Wiley, 1992)

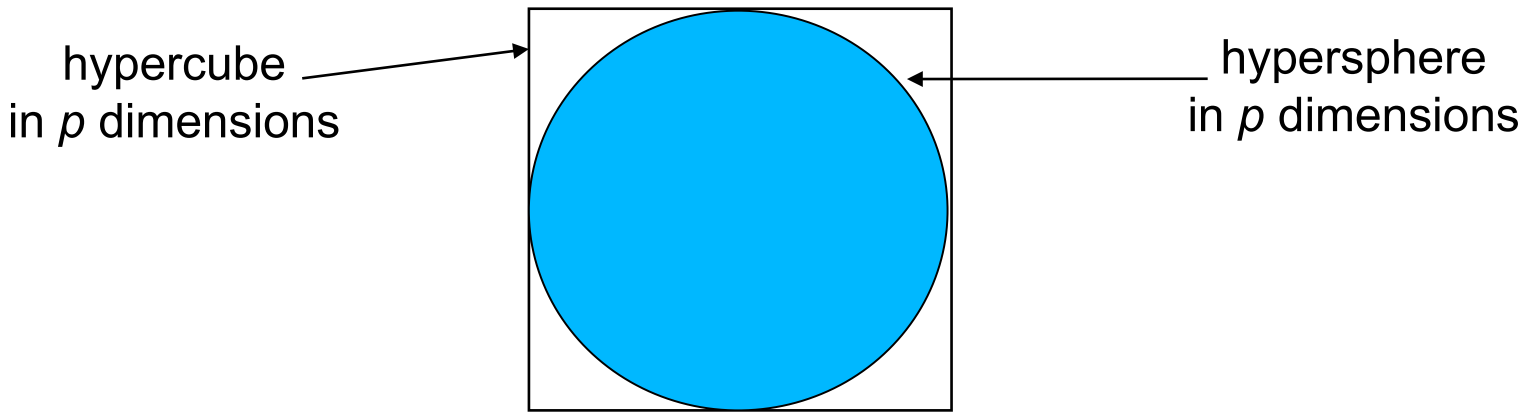


Volume of sphere relative to cube in *d* dimensions?

Dimension	**2**
Rel. vol.	**0.79**

The curse of dimensionality

(David Scott, *Multivariate Density Estimation,* Wiley, 1992)

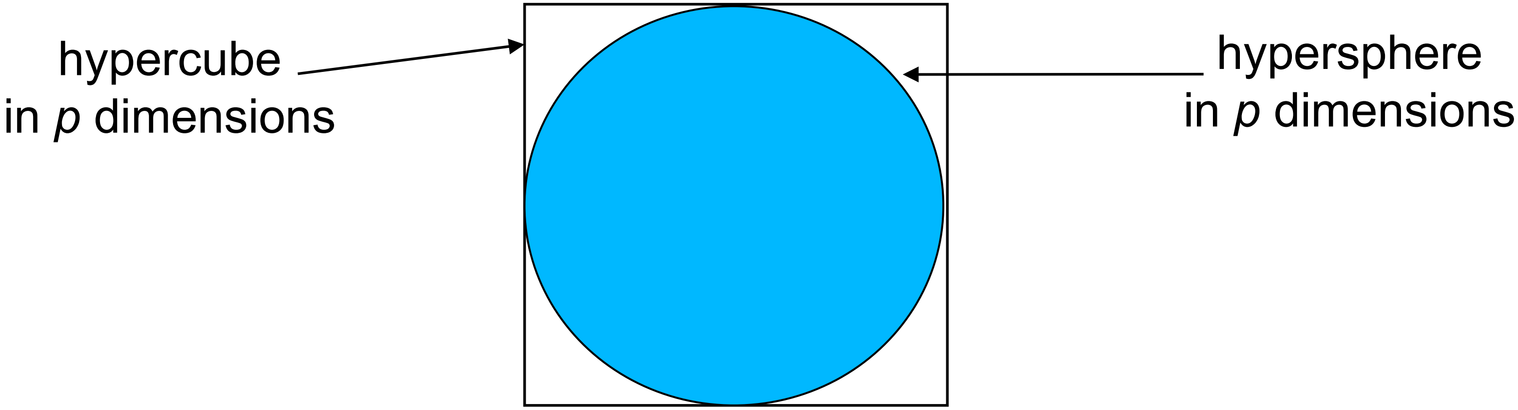


Volume of sphere relative to cube in *d* dimensions?

Dimension	**2**	**3**
Rel. vol.	**0.79**	**0.53**

The curse of dimensionality

(David Scott, *Multivariate Density Estimation,* Wiley, 1992)

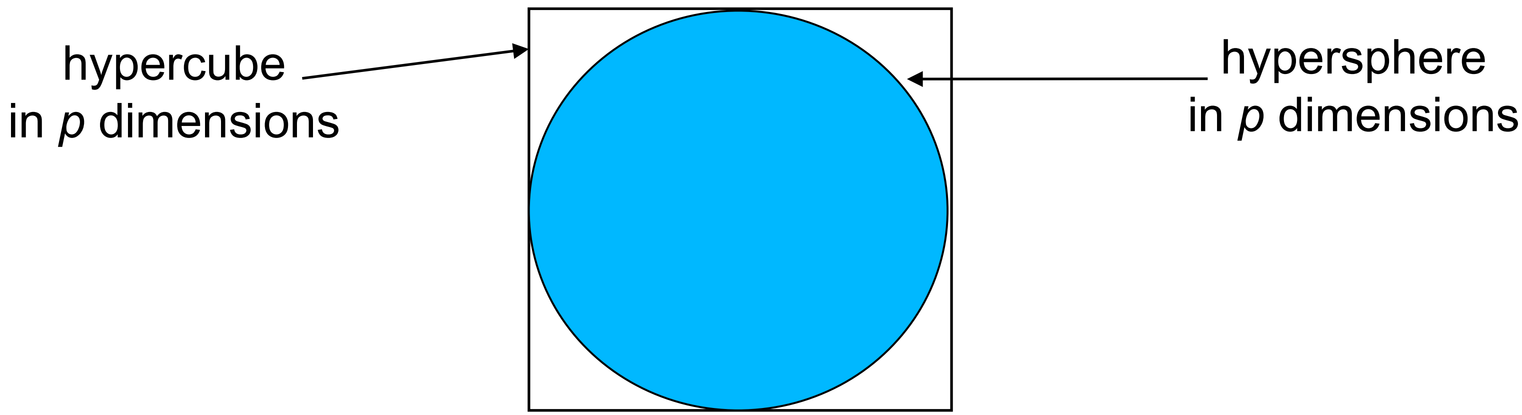


Volume of sphere relative to cube in *d* dimensions?

Dimension	**2**	**3**	**4**
Rel. vol.	**0.79**	**0.53**	**0.31**

The curse of dimensionality

(David Scott, *Multivariate Density Estimation,* Wiley, 1992)

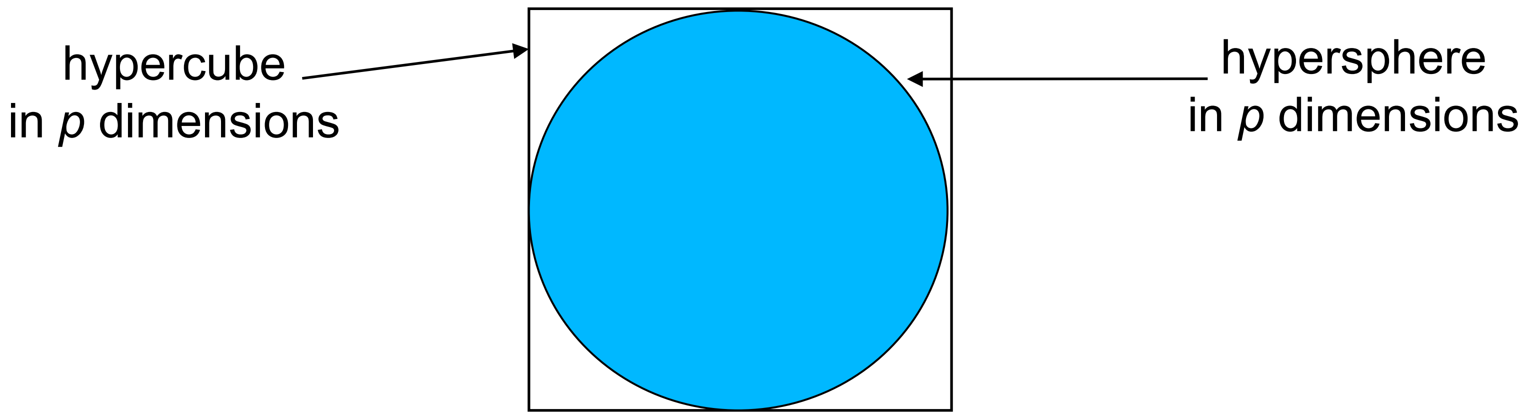


Volume of sphere relative to cube in *d* dimensions?

Dimension	**2**	**3**	**4**	**5**
Rel. vol.	**0.79**	**0.53**	**0.31**	**0.16**

The curse of dimensionality

(David Scott, *Multivariate Density Estimation,* Wiley, 1992)

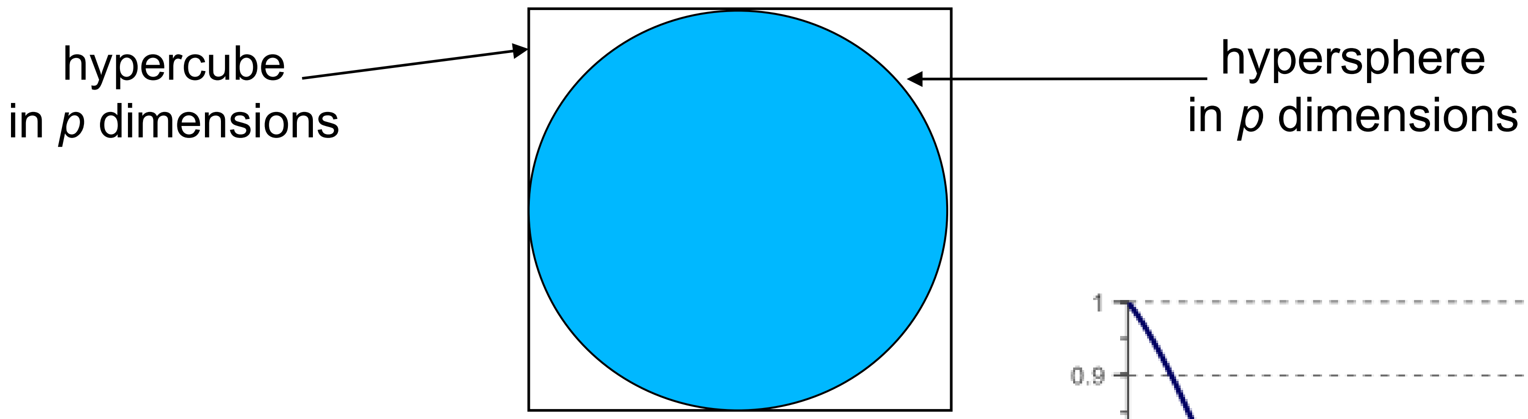


Volume of sphere relative to cube in *d* dimensions?

Dimension	**2**	**3**	**4**	**5**	**6**
Rel. vol.	**0.79**	**0.53**	**0.31**	**0.16**	**0.08**

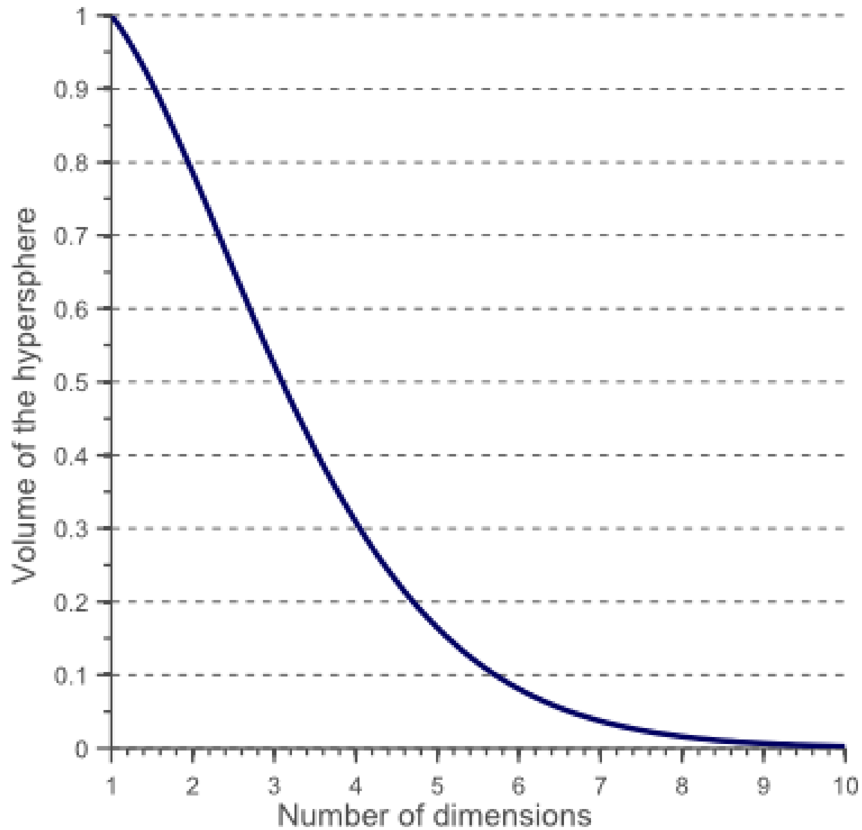
The curse of dimensionality

(David Scott, *Multivariate Density Estimation*, Wiley, 1992)



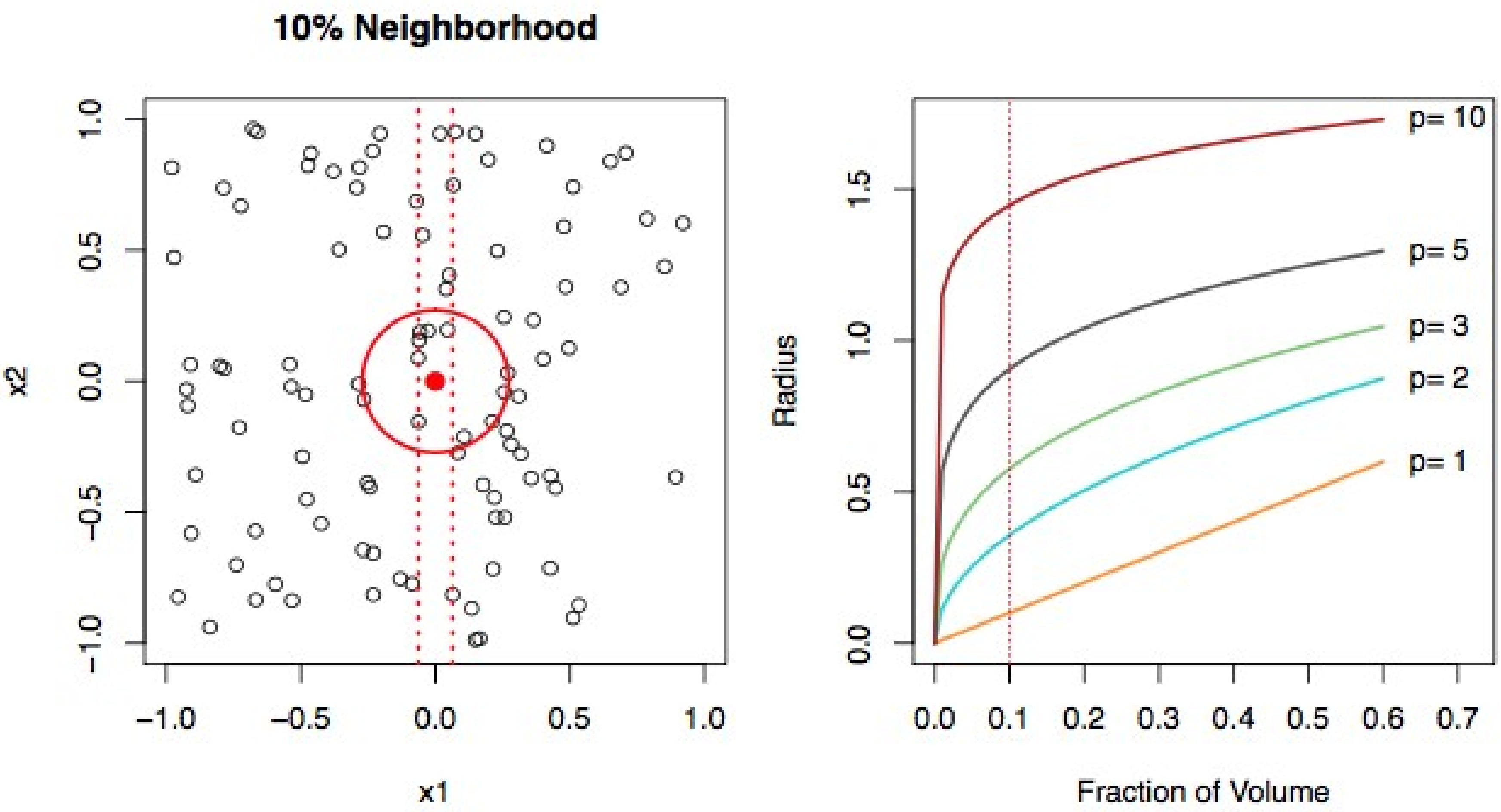
Volume of sphere relative to cube in *d* dimensions?

Dimension	2	3	4	5	6	7
Rel. vol.	0.79	0.53	0.31	0.16	0.08	0.04



As the dimensionality increases, a larger percentage of the training data resides in the corners of the feature space. Therefore, k-NN is unhelpful in high dimensional problems because there is little difference between the nearest and the farthest neighbor

The curse of dimensionality



The amount of training data needed to cover 10% of the feature range grows exponentially with the number of dimensions

Dimensionality reduction techniques (e.g. PCA ~ effective degrees of freedom) should be applied prior to using k-NN in order to help make the distance metric more meaningful