KNN, RANDOM FOREST, DT

Gates

TOPICS

1) kNN : k nearest neighbors

2) Random Forest

3) Decision Trees

LEARNING TYPES AND CONCEPTS

Gates

LEARNING STYLES

Supervised Learning

- ***Labeled input*** data exist to train a model. The model is then used to predict the class on unseen data.

Unsupervised Learning

- Input data are ***not labeled*** and the result is not known.

Semi-supervised Learning

- Input data is a ***mix*** of labeled and unlabeled examples.

Reinforcement Learning

- A model that **interacts with and learns from its environment.**
- Feedback is provided as ***punishments and rewards in the environment.***

Supervised Examples: Regression, Decision Tree, Random Forest, KNN, Logistic Regression, Naive Bayes, Support Vector Machines, Neural Networks

Unsupervised Examples: kmeans clustering, Association Rules

Reinforcement Learning Examples: Q-Learning, Temporal Difference (TD), Deep Adversarial Networks

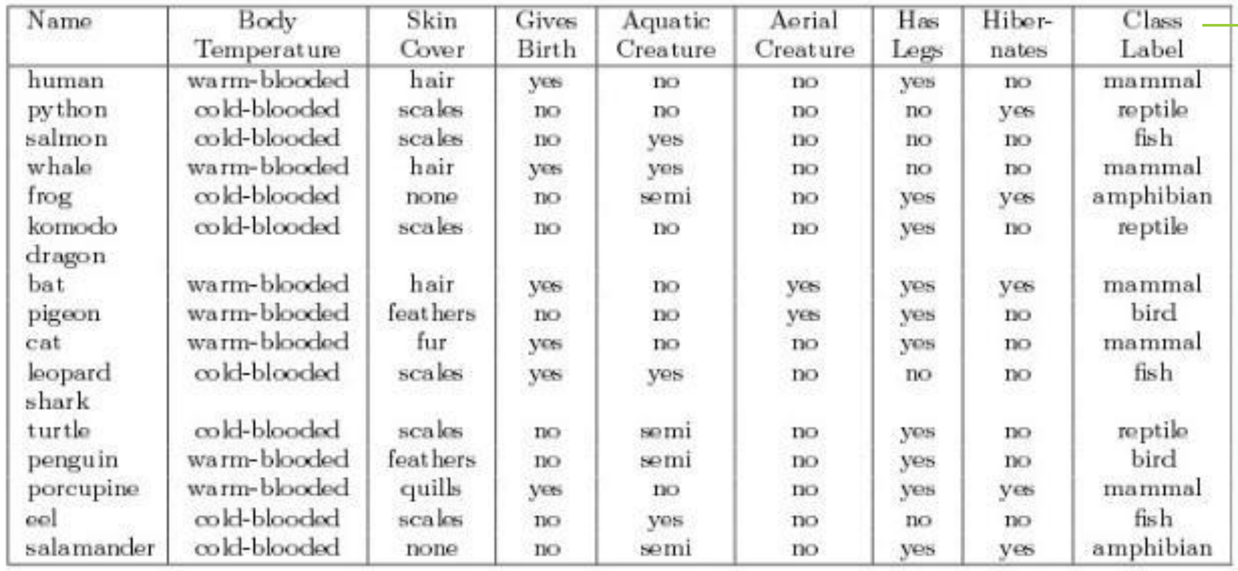
Interesting References:
https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

WHAT IS CLASSIFICATION

Given: a collection of records/vectors (*training dataset)* Each record contains a set of *attributes (variable values)*, **one of the attributes must be the *class*.**

Goal: Find a *model* (some function of the variable values) to identify the **class of a new vector/record.**

Table 4.1. The vertebrate data set.



Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

$\longrightarrow$ CLASS

CROSS-VALIDATION

- A *test set* is used to determine the accuracy of the model.
- Usually, the given data set is **divided into training and test sets**, with training sets used to build the model and the test set used to validate it.
- Training sets and testing sets should not overlap in values.
- **Cross-validation** (leave-one-out) is often used.

CONCEPTS FOR ML CLASSIFICATION

Input data: collection of records (also called an instance or example).

- For example: tuple(**x**, *y*), where **x** is the set (vector) of known attributes (variable values) and *y* is the **class label (called the target)**.

Classification: learning a target/class **function** f that maps any vector of attributes, **x** to a predefined class y.

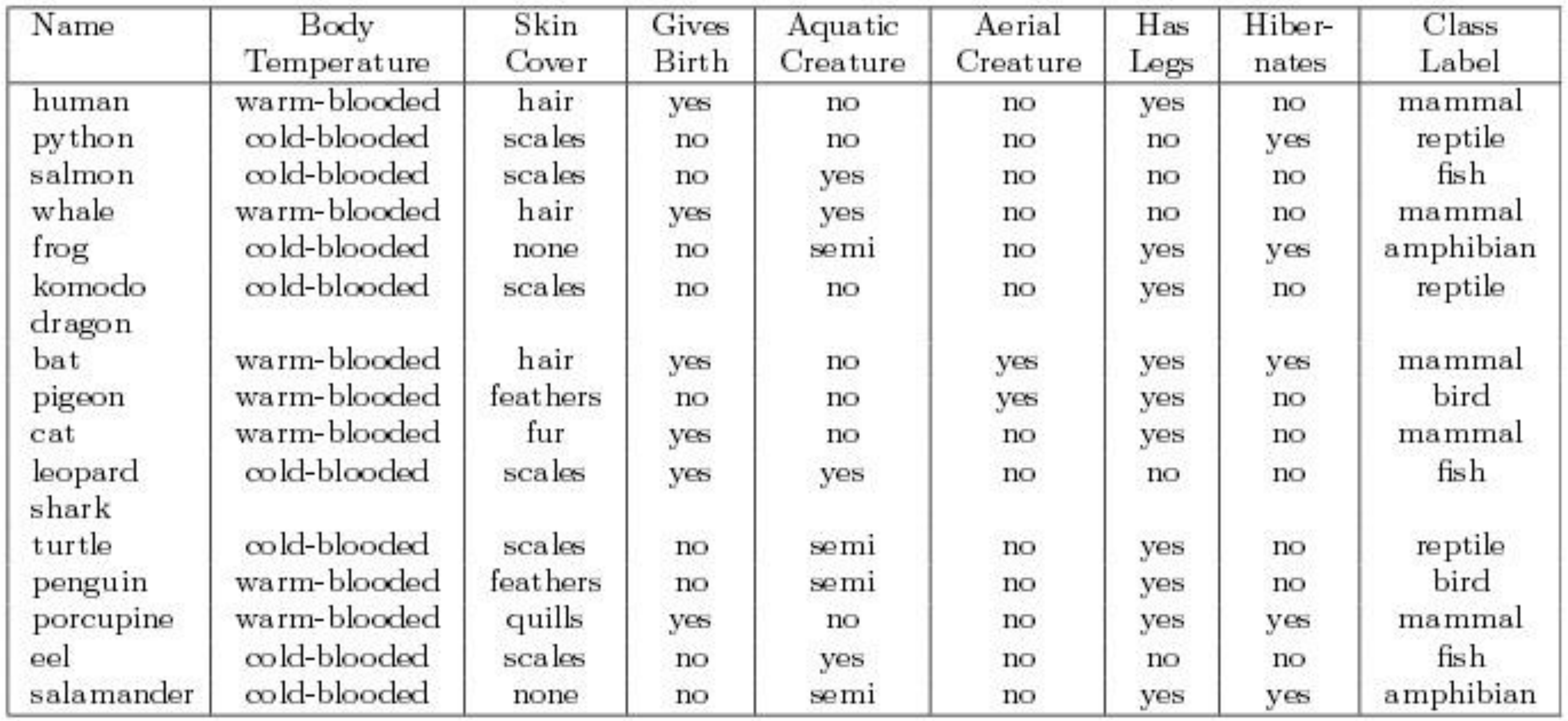
f: **x** → y f is a classification model

Descriptive Modeling: Classification model that can distinguish between objects of different classes.

Predictive Modeling: Using a classification model to predict a label/class given a vector/record **x**

EXAMPLE: FEATURE TABLE

Table 4.1. The vertebrate data set.



Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf, page 147

1. **What are the classes, y?**
2. **What are the records, x?**

K NEAREST NEIGHBORS

Gates

WHAT IS K-NN (K NEAREST NEIGHBOR)

1) The kNN algorithm looks for data points (also called vectors or rows) that are "closest" to the new data point.

2) Given this, the first question is: what do you mean by "closest"?

3) The next question is: How many of these "close" points should we look at? (this is k)

4) The third question is: Using these "close" points, how do we decide on the classification of our data point?

Our data point is the vector of values we are trying to classify or categorize.

The "close" points are called the **nearest neighbors**.

We can measure "closeness" with any number of distance or similarity measures – such as Euclidean, Manhattan, Cosine Similarity, etc.

CHOOSING K

1) In kNN, we must determine the value of k.

2) k is the number of **nearest neighbors** that we will consider in determining the value of our data point.

3) If k is too small – we risk **over-fitting**.

4) If k is too large – we risk mis-classification.

THE KNN GENERAL ALGORITHM

1) Suppose you have a **test sample** of data points that you want to classify.

2) Suppose you also have a **training sample** of data points for which you already know the classification or label.

3) For each test sample data point, you will calculate the distance between that point and all other points in the Training set.

4) Next, you will identify the k “closest” points in the training set to your test point.

5) Finally, using the k closest points, the majority vote (most common among the k points) is used as the classification.

ABOUT KNN

1) kNN is a type of **instance-based learning.**

2) kNN does not build or require a model and as such is **non-parametric**.

3) kNN and other instance-based learning methods require a measure of similarity or distance.

4) kNN is also known as a **lazy learner** because no model is built and Training data points are only used as they are needed (when they are near to the test point).

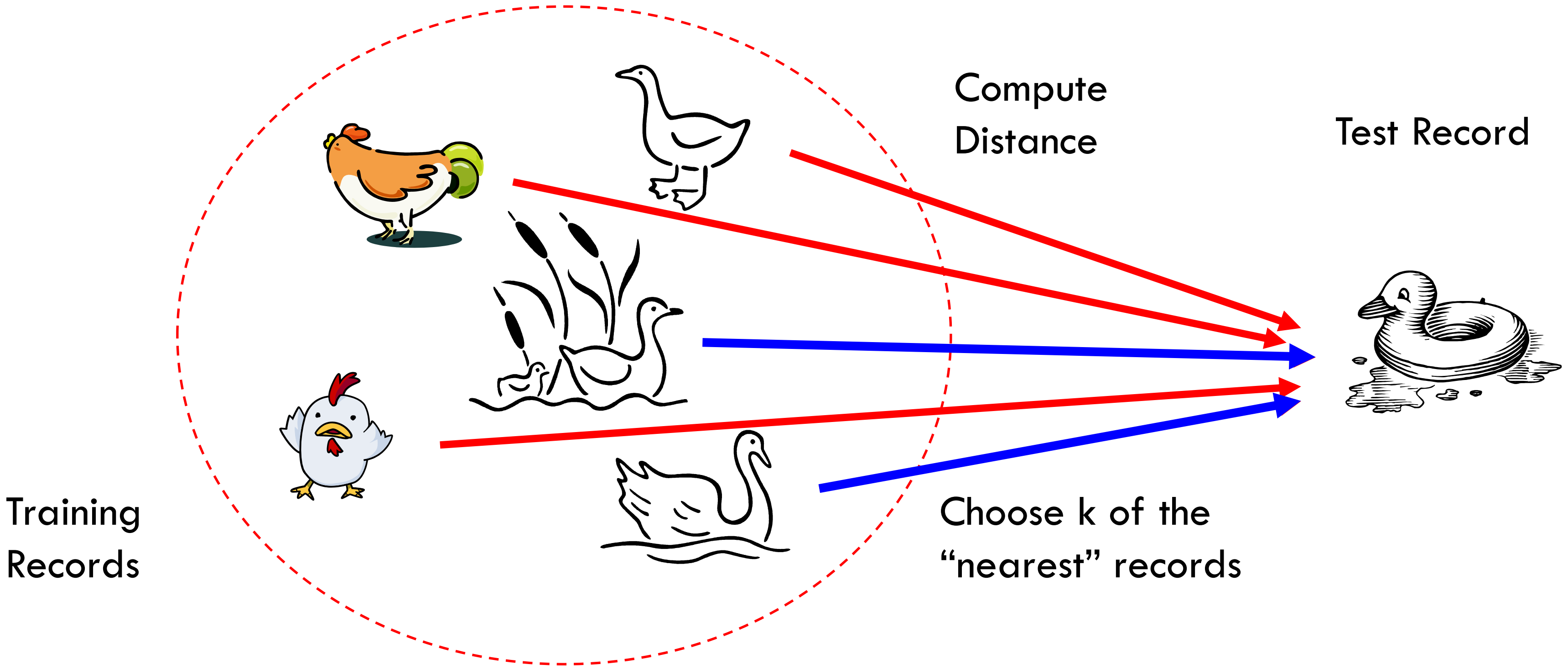
5) kNN can produce arbitrary decision boundaries. In other words, they do not have to be linear or polynomial, or matching any model. As such, they are more flexible.

6) The distance measure is critical. If a poor measure is used, results will be inaccurate. **Normalization (scaling) is often required** so that certain attributes do not dominate the distance measure.

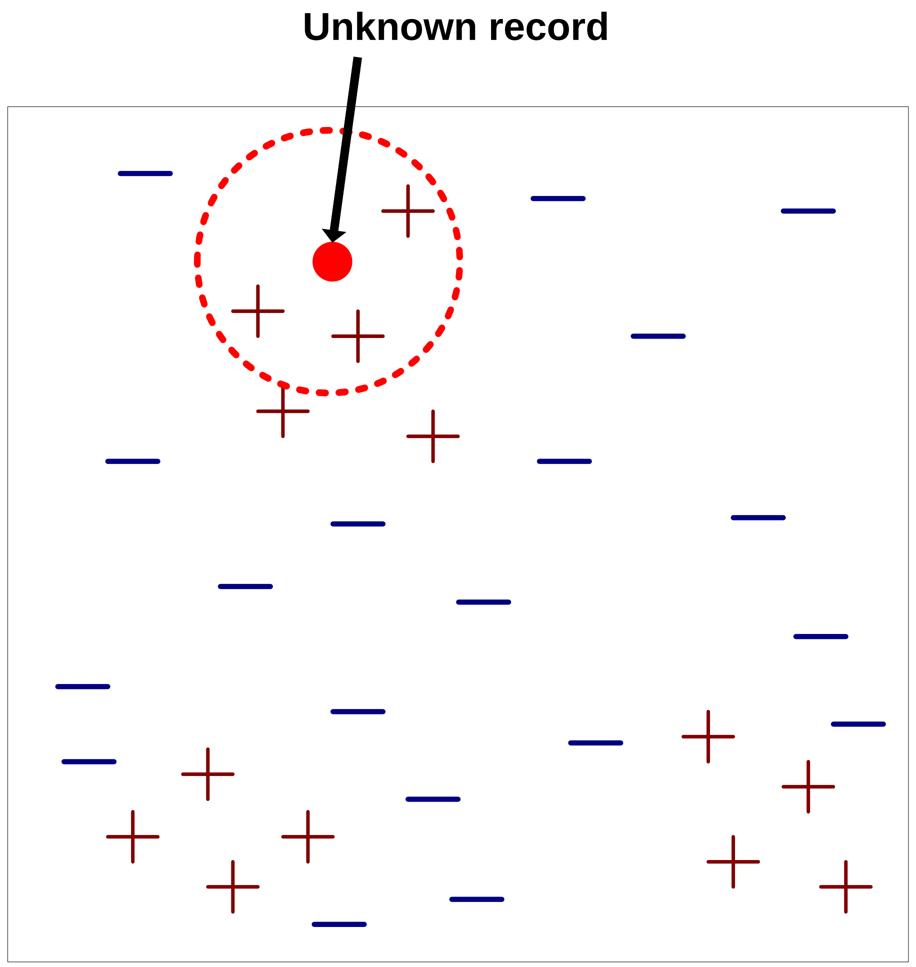
NEAREST NEIGHBOR CLASSIFIERS

Basic idea:

- If it walks like a duck, quacks like a duck, then it's probably a duck



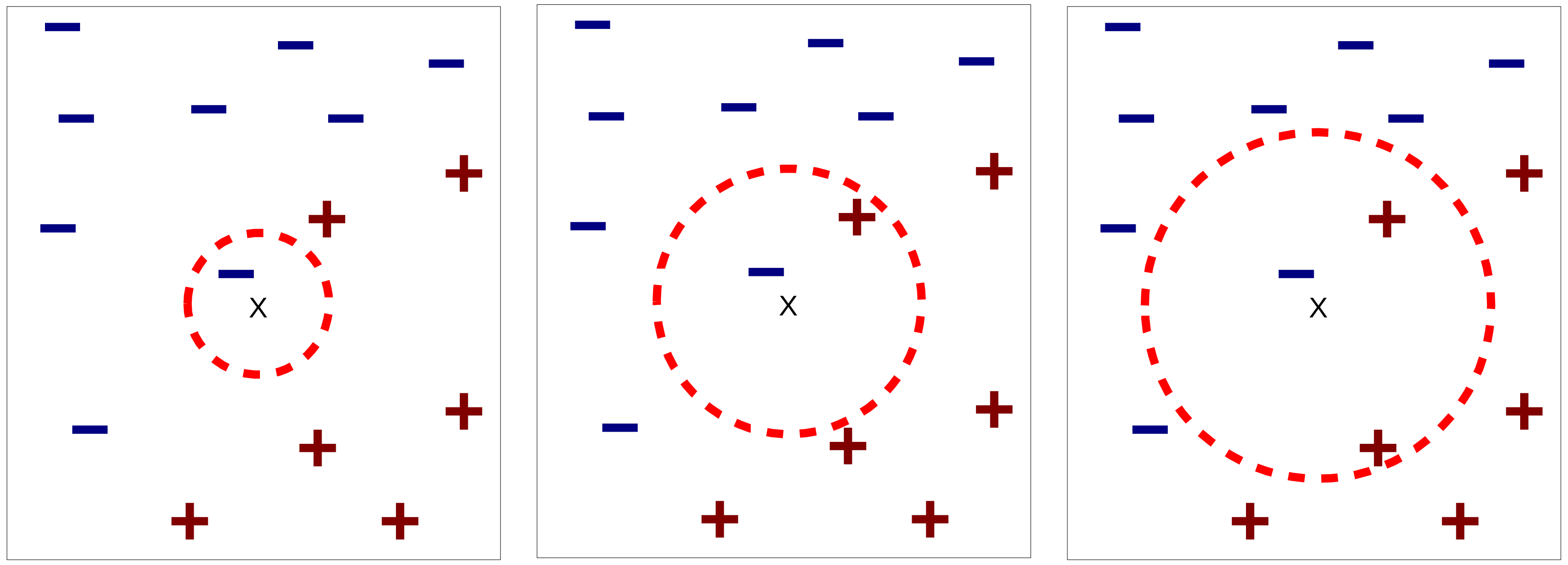
NEAREST-NEIGHBOR CLASSIFIERS



- **Requires three things**
  - The set of **stored records**
  - **Distance Metric** to compute distance between records
  - The **value of *k***, the number of nearest neighbors to retrieve

- To **classify an unknown record**:
  - **Compute distance** to other training records
  - **Identify *k* nearest neighbors**
  - **Use class labels of nearest neighbors** to determine the class label of unknown record (e.g., by taking majority vote)

DEFINITION OF NEAREST NEIGHBOR



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

A NEAREST NEIGHBOR DISTANCE OPTION

Compute distance between two points:

- **Euclidean distance**

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

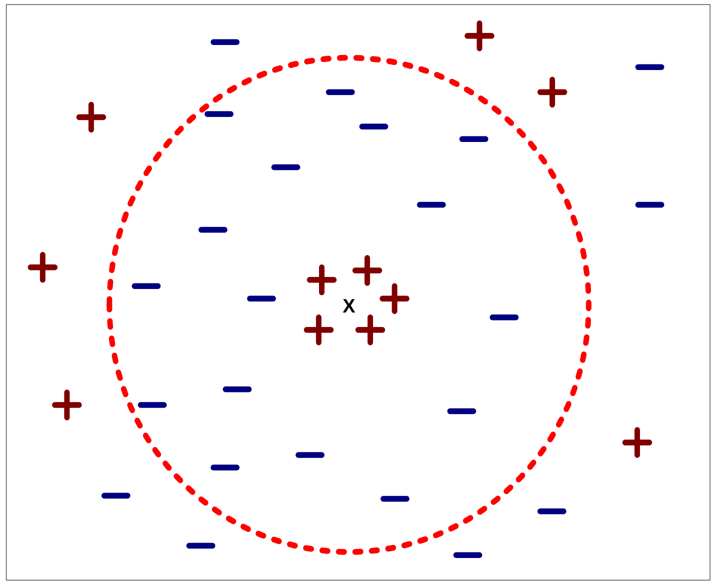
Determine the class from nearest neighbor list

- take the **majority vote** of class labels among the k-nearest neighbors
- Weigh the vote according to distance
  - weight factor, $w = 1/d^2$

NEAREST NEIGHBOR – DETERMINE K

Choosing the value of k:

- If k is too small, sensitive to noise points
- If k is too large, neighborhood may include points from other classes



NEAREST NEIGHBOR ISSUES

Scaling issues

- Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
- Example:
  - height of a person may vary from 1.5m to 1.8m
  - weight of a person may vary from 90lb to 300lb
  - income of a person may vary from \$10K to \$1M

KNN IN R

```
## The following libraries were used for both kNN and Random Forest:
library(datasets)
library(class) ## for knn
library(mlr) ## for vis
library(ggplot2)
library(plyr) ## load this BEFORE dplyr
library(dplyr)
library(lattice)
library(caret)
library(e1071)
library(ElemStatLearn)
library(gmodels)
library(GGally)
library(randomForest)
```

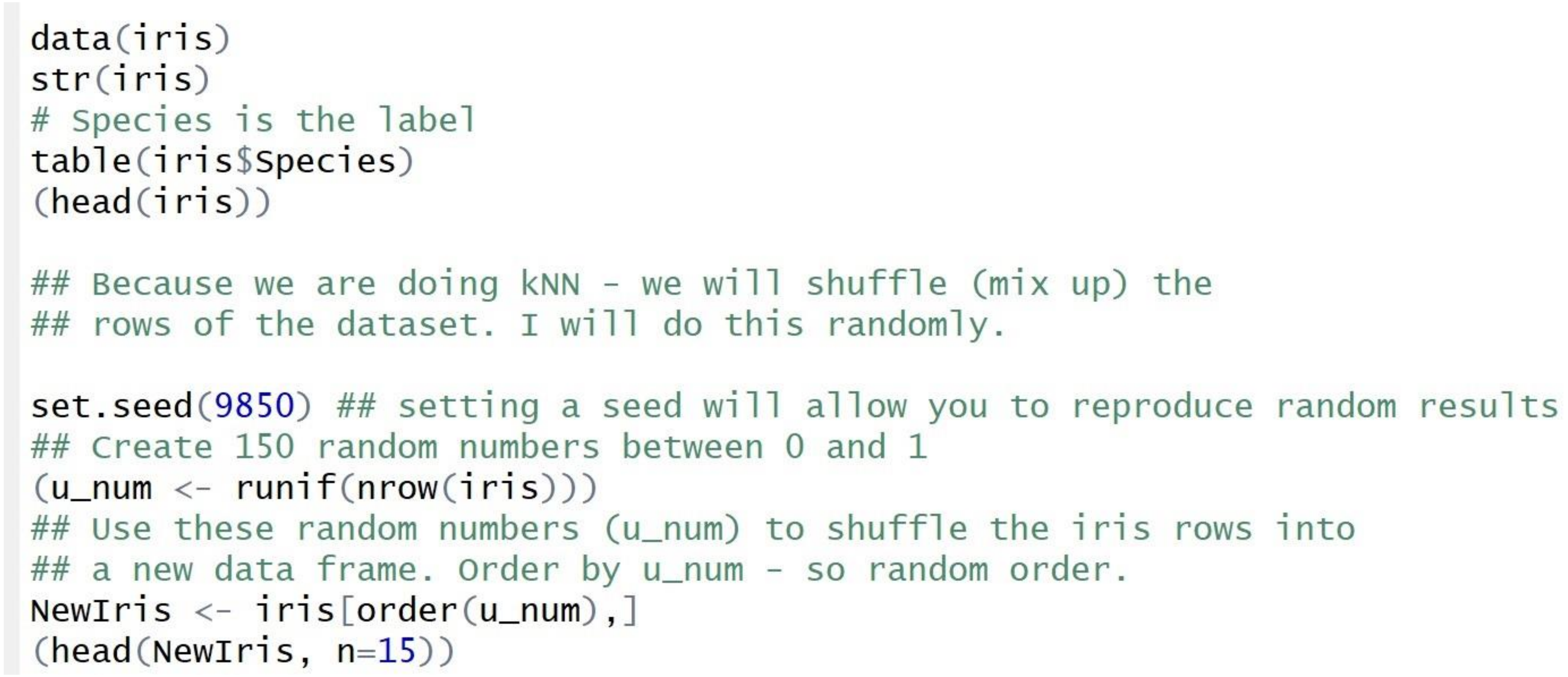
IRIS DATASET

For this R example, I will use the well known iris dataset in R.

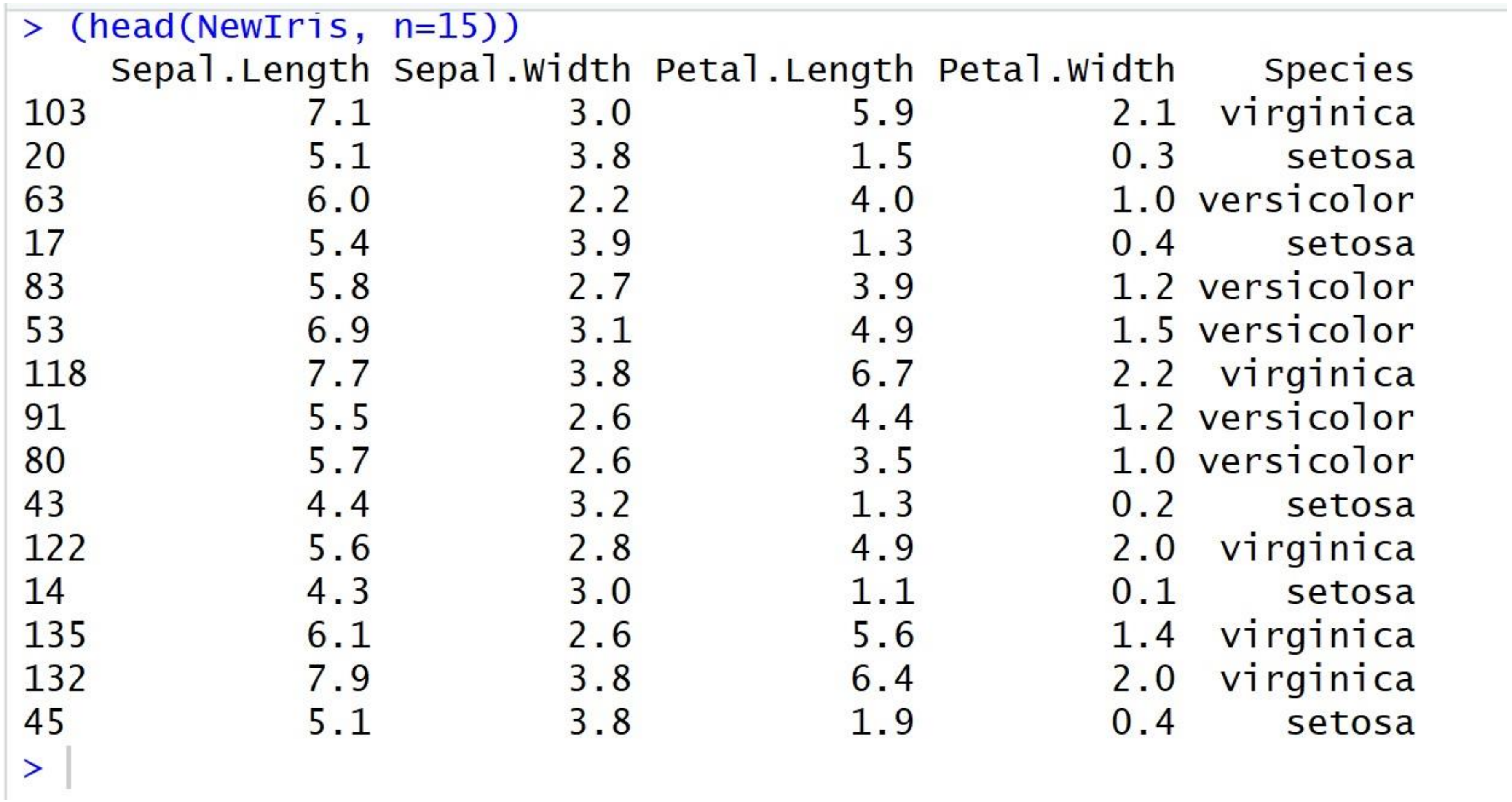
This will allow more time to focus on the method and less time to focus on cleaning and preparation.

It is always a good idea to use a simple dataset (such as Iris) to try out new methods and models.

It is also a good idea to practice investigating the data with statistical and visual EDA (exploratory data anlaysis.

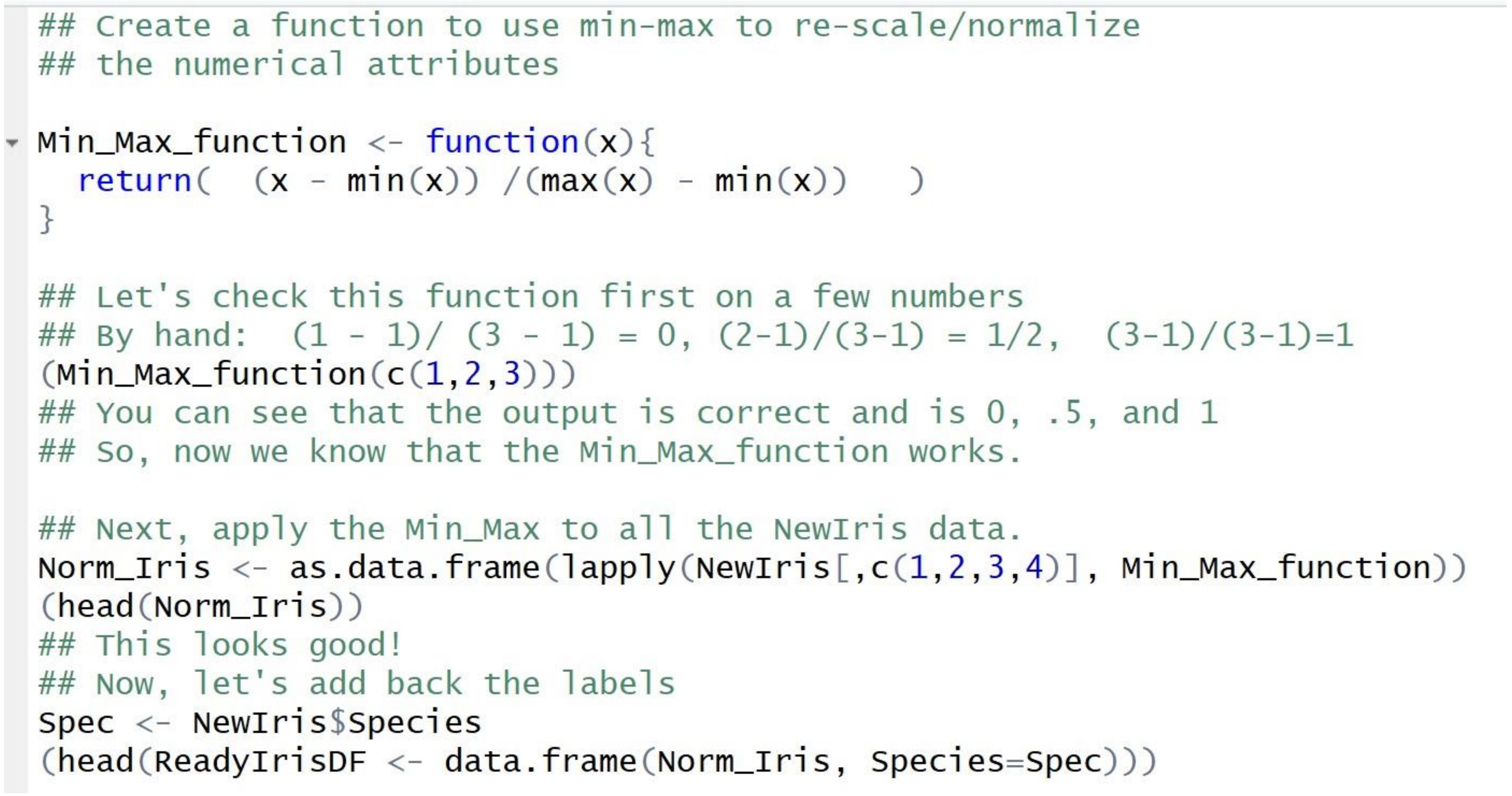


```
data(iris)
str(iris)
# Species is the label
table(iris$Species)
(head(iris))

## Because we are doing kNN - we will shuffle (mix up) the
## rows of the dataset. I will do this randomly.

set.seed(9850) ## setting a seed will allow you to reproduce random results
## Create 150 random numbers between 0 and 1
(u_num <- runif(nrow(iris)))
## Use these random numbers (u_num) to shuffle the iris rows into
## a new data frame. Order by u_num - so random order.
NewIris <- iris[order(u_num),]
(head(NewIris, n=15))
```



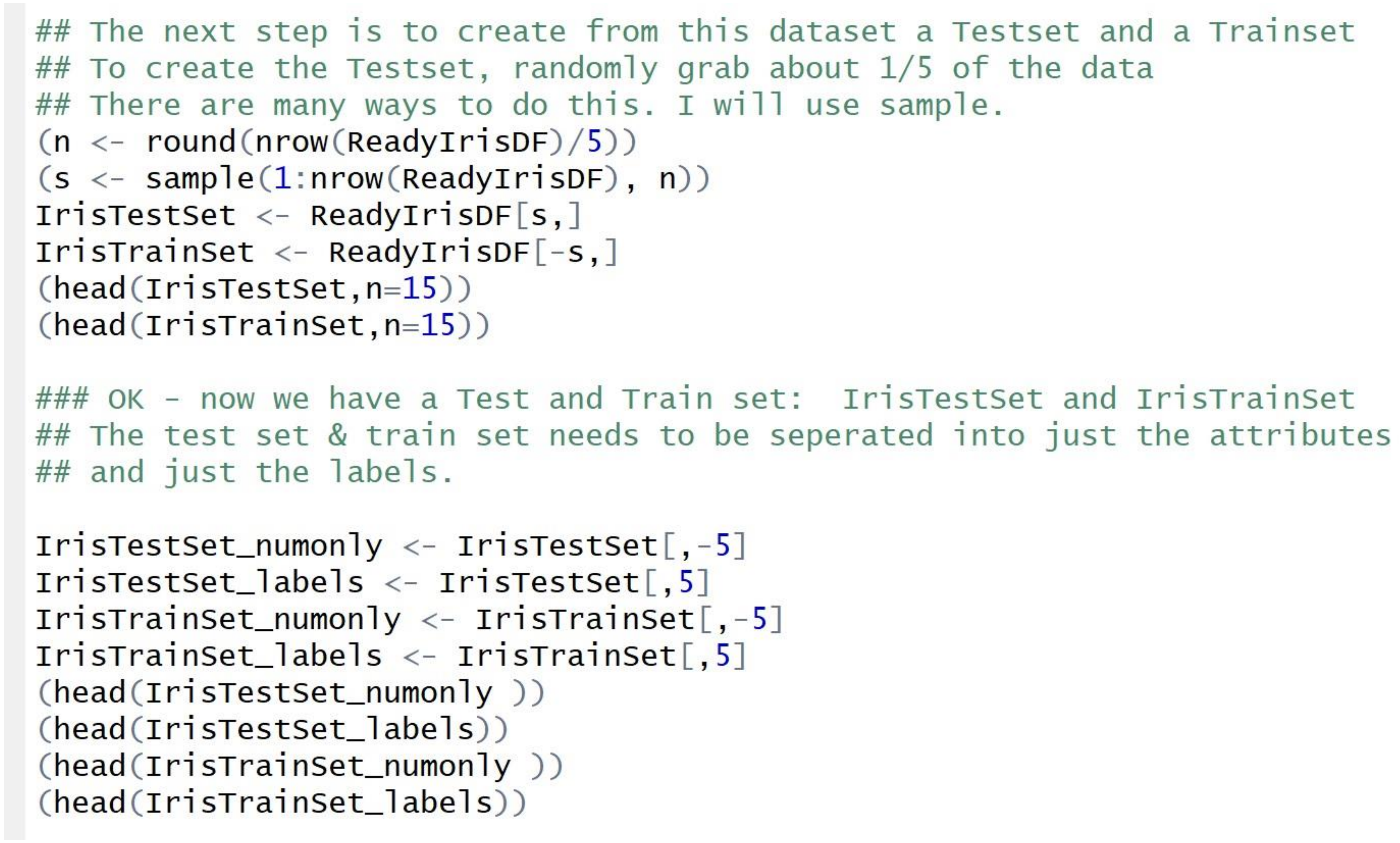
```
> (head(NewIris, n=15))
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
103          7.1         3.0          5.9         2.1  virginica
20           5.1         3.8          1.5         0.3     setosa
63           6.0         2.2          4.0         1.0 versicolor
17           5.4         3.9          1.3         0.4     setosa
83           5.8         2.7          3.9         1.2 versicolor
53           6.9         3.1          4.9         1.5 versicolor
118          7.7         3.8          6.7         2.2  virginica
91           5.5         2.6          4.4         1.2 versicolor
80           5.7         2.6          3.5         1.0 versicolor
43           4.4         3.2          1.3         0.2     setosa
122          5.6         2.8          4.9         2.0  virginica
14           4.3         3.0          1.1         0.1     setosa
135          6.1         2.6          5.6         1.4  virginica
132          7.9         3.8          6.4         2.0  virginica
45           5.1         3.8          1.9         0.4     setosa
> 
```

NORMALIZE THE DATA – THIS IS IMPORTANT



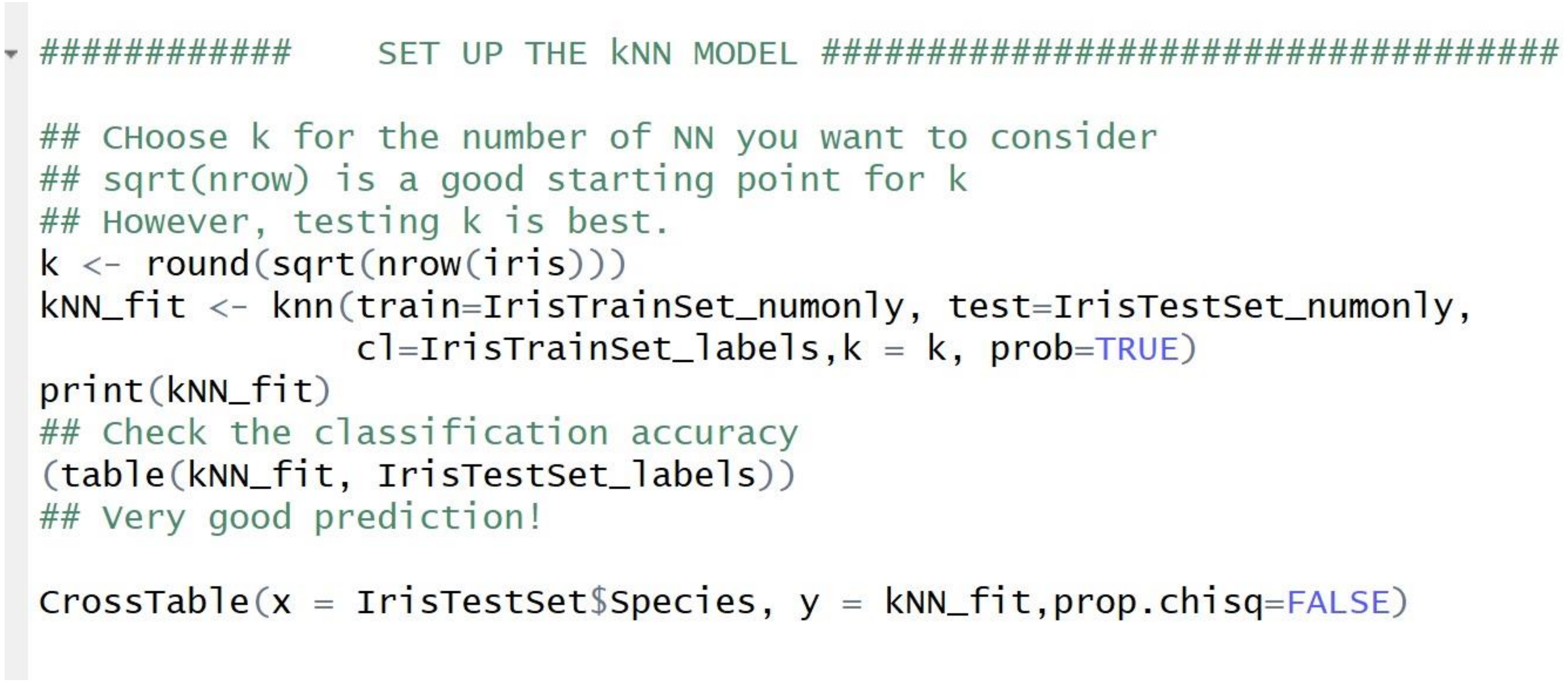
```
## Create a function to use min-max to re-scale/normalize
## the numerical attributes

Min_Max_function <- function(x){
  return(  (x - min(x)) /(max(x) - min(x))    )
}

## Let's check this function first on a few numbers
## By hand:  (1 - 1)/ (3 - 1) = 0, (2-1)/(3-1) = 1/2,  (3-1)/(3-1)=1
(Min_Max_function(c(1,2,3)))
## You can see that the output is correct and is 0, .5, and 1
## So, now we know that the Min_Max_function works.

## Next, apply the Min_Max to all the NewIris data.
Norm_Iris <- as.data.frame(lapply(NewIris[,c(1,2,3,4)], Min_Max_function))
(head(Norm_Iris))
## This looks good!
## Now, let's add back the labels
Spec <- NewIris$Species
(head(ReadyIrisDF <- data.frame(Norm_Iris, Species=Spec)))
```

CREATE THE TRAINING AND TESTING DATA AND LABELS

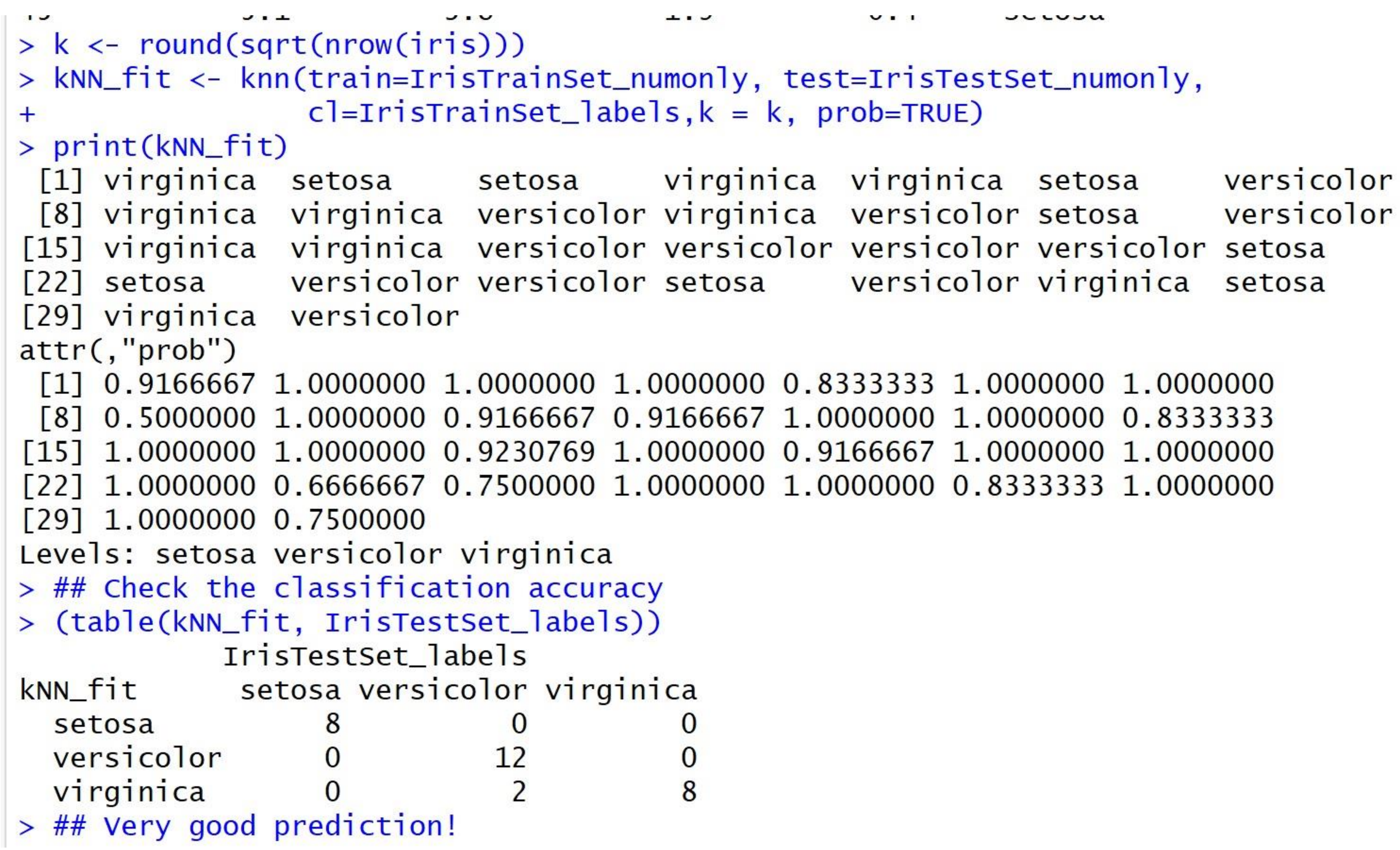


```
## The next step is to create from this dataset a Testset and a Trainset
## To create the Testset, randomly grab about 1/5 of the data
## There are many ways to do this. I will use sample.
(n <- round(nrow(ReadyIrisDF)/5))
(s <- sample(1:nrow(ReadyIrisDF), n))
IrisTestSet <- ReadyIrisDF[s,]
IrisTrainSet <- ReadyIrisDF[-s,]
(head(IrisTestSet,n=15))
(head(IrisTrainSet,n=15))

### OK - now we have a Test and Train set:  IrisTestSet and IrisTrainSet
## The test set & train set needs to be seperated into just the attributes
## and just the labels.

IrisTestSet_numonly <- IrisTestSet[,-5]
IrisTestSet_labels <- IrisTestSet[,5]
IrisTrainSet_numonly <- IrisTrainSet[,-5]
IrisTrainSet_labels <- IrisTrainSet[,5]
(head(IrisTestSet_numonly ))
(head(IrisTestSet_labels))
(head(IrisTrainSet_numonly ))
(head(IrisTrainSet_labels))
```

SET UP THE KNN MODEL

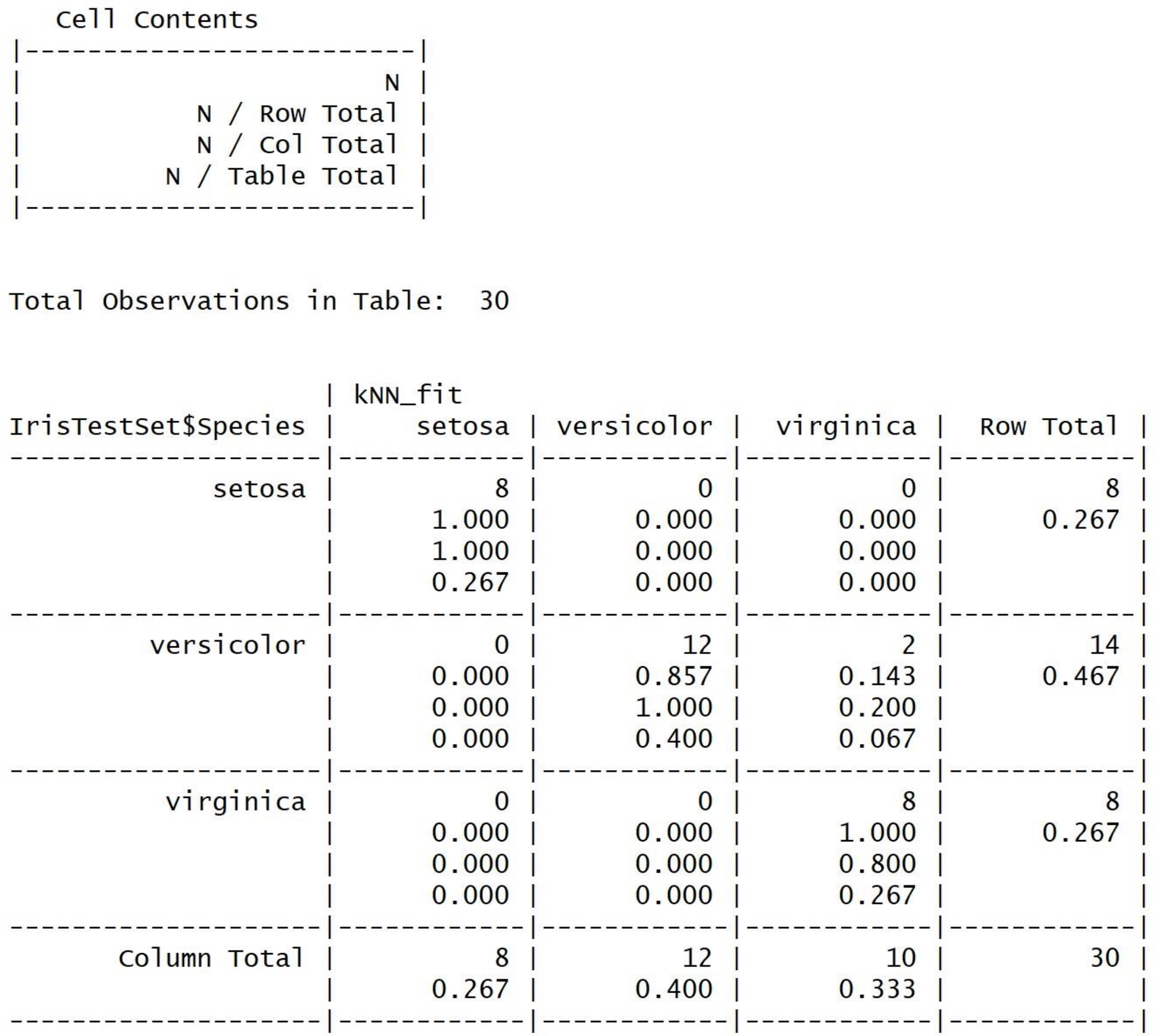

```
############    SET UP THE kNN MODEL ###################################

## CHoose k for the number of NN you want to consider
## sqrt(nrow) is a good starting point for k
## However, testing k is best.
k <- round(sqrt(nrow(iris)))
kNN_fit <- knn(train=IrisTrainSet_numonly, test=IrisTestSet_numonly,
               cl=IrisTrainSet_labels,k = k, prob=TRUE)
print(kNN_fit)
## Check the classification accuracy
(table(kNN_fit, IrisTestSet_labels))
## Very good prediction!

CrossTable(x = IrisTestSet$Species, y = kNN_fit,prop.chisq=FALSE)
```

RESULTS PART 1



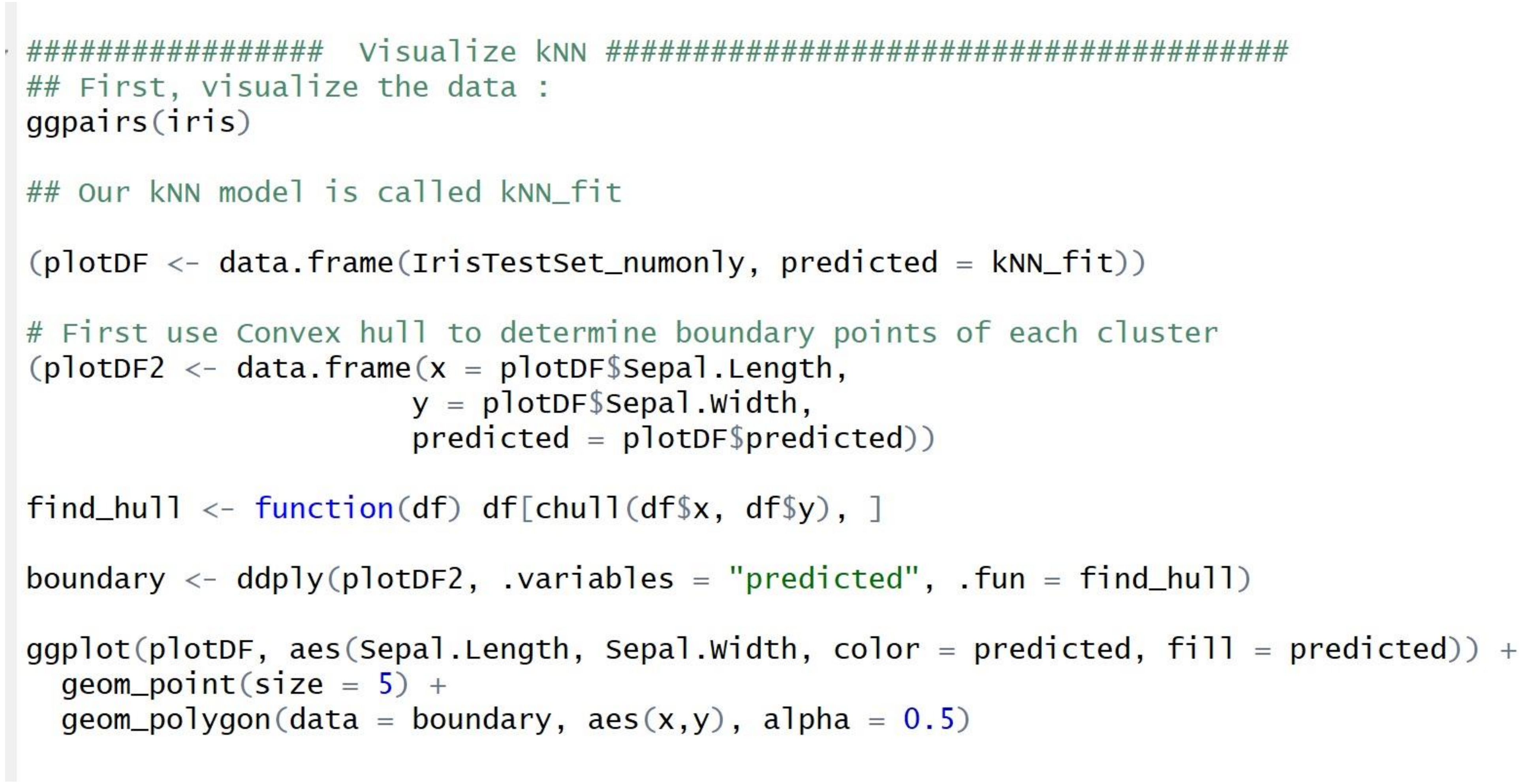
```
> k <- round(sqrt(nrow(iris)))
> kNN_fit <- knn(train=IrisTrainSet_numonly, test=IrisTestSet_numonly,
+                cl=IrisTrainSet_labels,k = k, prob=TRUE)
> print(kNN_fit)
 [1] virginica  setosa     setosa     virginica  virginica  setosa     versicolor
 [8] virginica  virginica  versicolor virginica  versicolor setosa     versicolor
[15] virginica  virginica  versicolor versicolor versicolor versicolor setosa
[22] setosa     versicolor versicolor setosa     versicolor virginica  setosa
[29] virginica  versicolor
attr(,"prob")
 [1] 0.9166667 1.0000000 1.0000000 1.0000000 0.8333333 1.0000000 1.0000000
 [8] 0.5000000 1.0000000 0.9166667 0.9166667 1.0000000 1.0000000 0.8333333
[15] 1.0000000 1.0000000 0.9230769 1.0000000 0.9166667 1.0000000 1.0000000
[22] 1.0000000 0.6666667 0.7500000 1.0000000 1.0000000 0.8333333 1.0000000
[29] 1.0000000 0.7500000
Levels: setosa versicolor virginica
> ## Check the classification accuracy
> (table(kNN_fit, IrisTestSet_labels))
            IrisTestSet_labels
kNN_fit      setosa versicolor virginica
  setosa          8          0         0
  versicolor      0         12         0
  virginica       0          2         8
> ## Very good prediction!
```

RESULTS PART 2



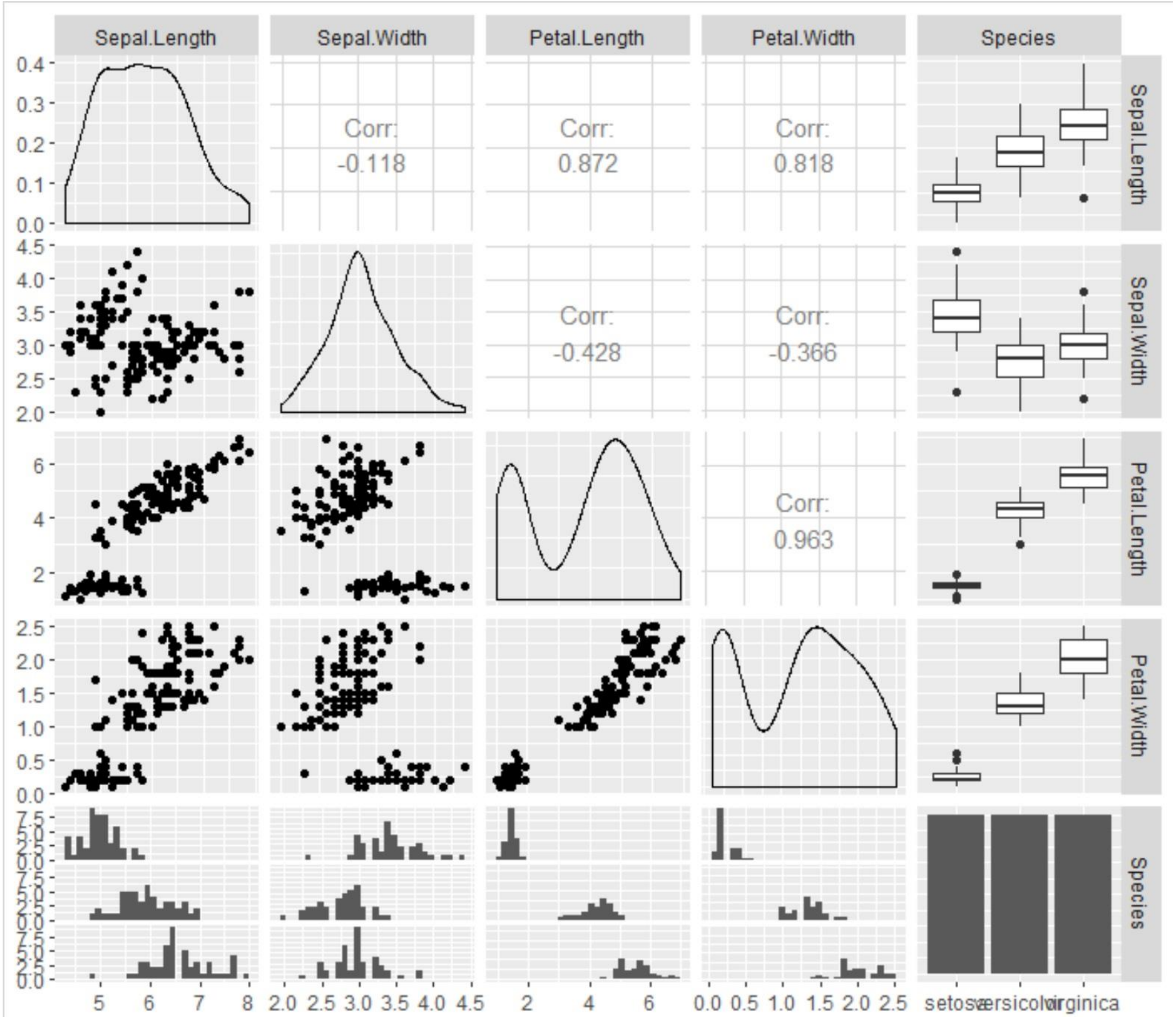
```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  30


                   | kNN_fit 
IrisTestSet$Species |     setosa | versicolor |  virginica |  Row Total | 
--------------------|------------|------------|------------|------------|
             setosa |          8 |          0 |          0 |          8 | 
                    |      1.000 |      0.000 |      0.000 |      0.267 | 
                    |      1.000 |      0.000 |      0.000 |            | 
                    |      0.267 |      0.000 |      0.000 |            | 
--------------------|------------|------------|------------|------------|
         versicolor |          0 |         12 |          2 |         14 | 
                    |      0.000 |      0.857 |      0.143 |      0.467 | 
                    |      0.000 |      1.000 |      0.200 |            | 
                    |      0.000 |      0.400 |      0.067 |            | 
--------------------|------------|------------|------------|------------|
          virginica |          0 |          0 |          8 |          8 | 
                    |      0.000 |      0.000 |      1.000 |      0.267 | 
                    |      0.000 |      0.000 |      0.800 |            | 
                    |      0.000 |      0.000 |      0.267 |            | 
--------------------|------------|------------|------------|------------|
       Column Total |          8 |         12 |         10 |         30 | 
                    |      0.267 |      0.400 |      0.333 |            | 
--------------------|------------|------------|------------|------------|
```

VISUALIZE RESULTS

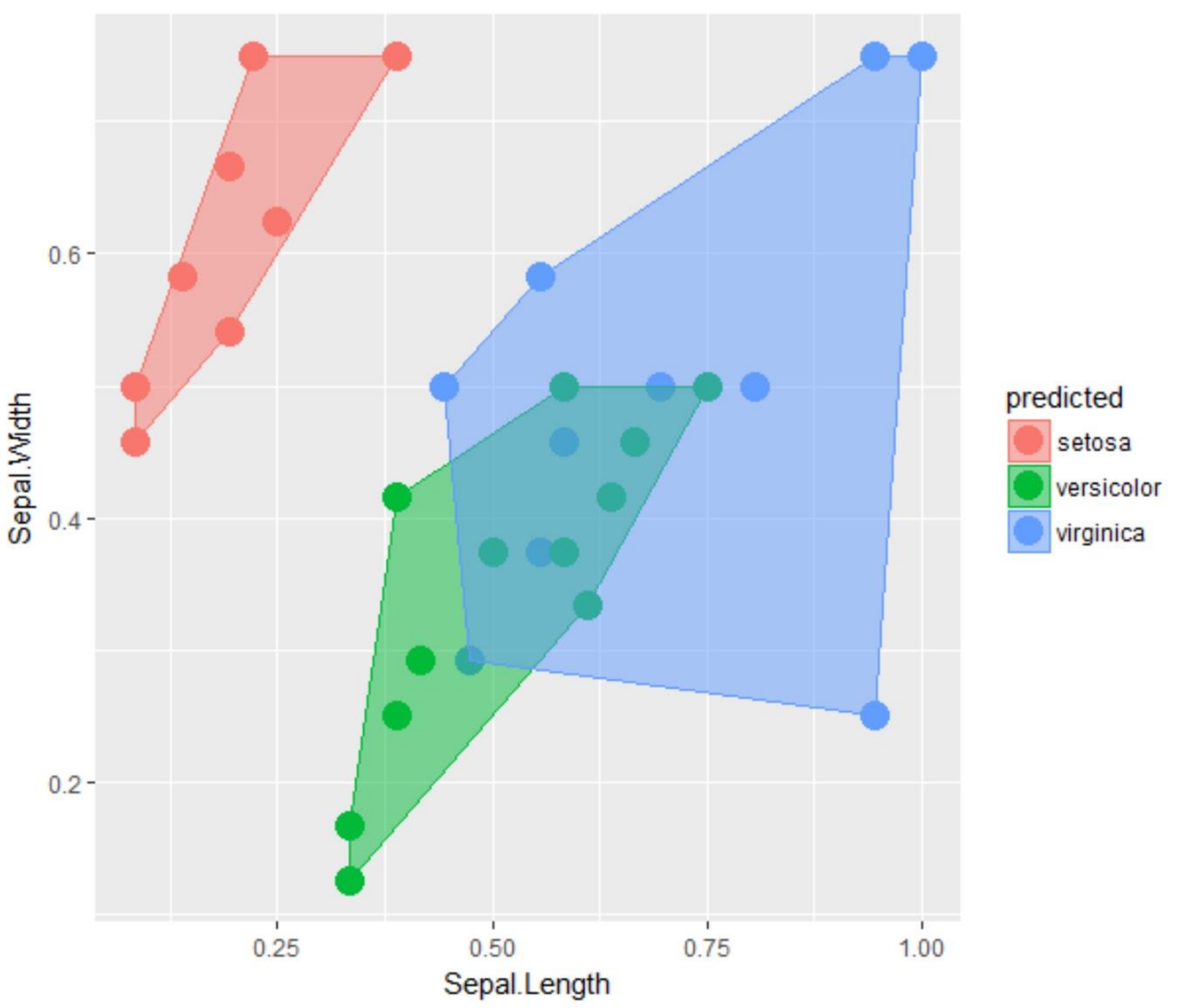


```
#################  Visualize kNN ########################################
## First, visualize the data :
ggpairs(iris)

## Our kNN model is called kNN_fit

(plotDF <- data.frame(IrisTestSet_numonly, predicted = kNN_fit))

# First use Convex hull to determine boundary points of each cluster
(plotDF2 <- data.frame(x = plotDF$Sepal.Length,
                      y = plotDF$Sepal.Width,
                      predicted = plotDF$predicted))

find_hull <- function(df) df[chull(df$x, df$y), ]

boundary <- ddply(plotDF2, .variables = "predicted", .fun = find_hull)

ggplot(plotDF, aes(Sepal.Length, Sepal.Width, color = predicted, fill = predicted)) +
  geom_point(size = 5) +
  geom_polygon(data = boundary, aes(x,y), alpha = 0.5)
```

VIS 1


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Species
Corr: -0.118
Corr: 0.872
Corr: 0.818
Corr: -0.428
Corr: -0.366
Corr: 0.963
setosa
versicolor
virginica

VIS RESULT 2


0.6
0.4
0.2
0.25
0.50
0.75
1.00
Sepal.Length
Sepal.Width
predicted
setosa
versicolor
virginica

NEAREST NEIGHBOR SUMMARY

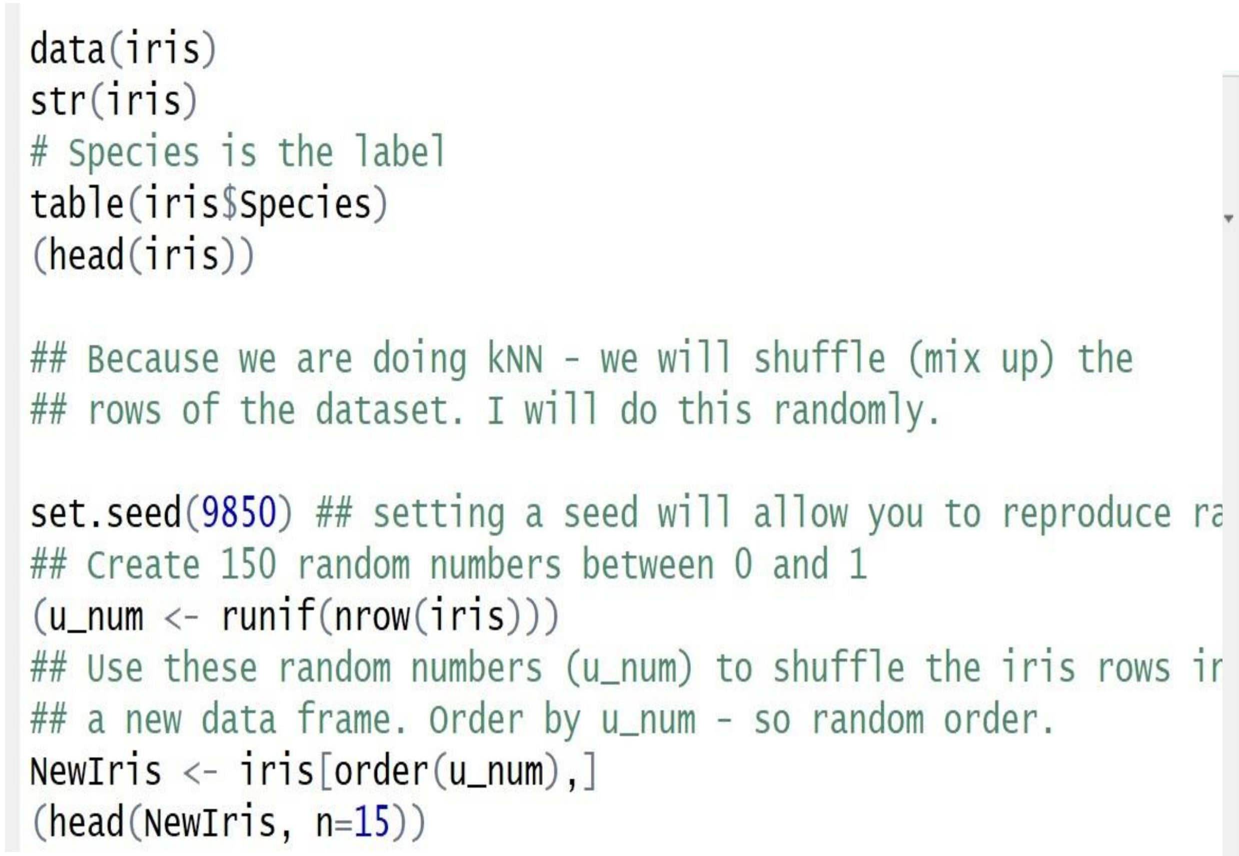
k-NN classifier

- It does not build models explicitly
- Unlike eager learners such as decision tree induction and rule-based systems (lazy learner)
- Classifying unknown records are relatively expensive

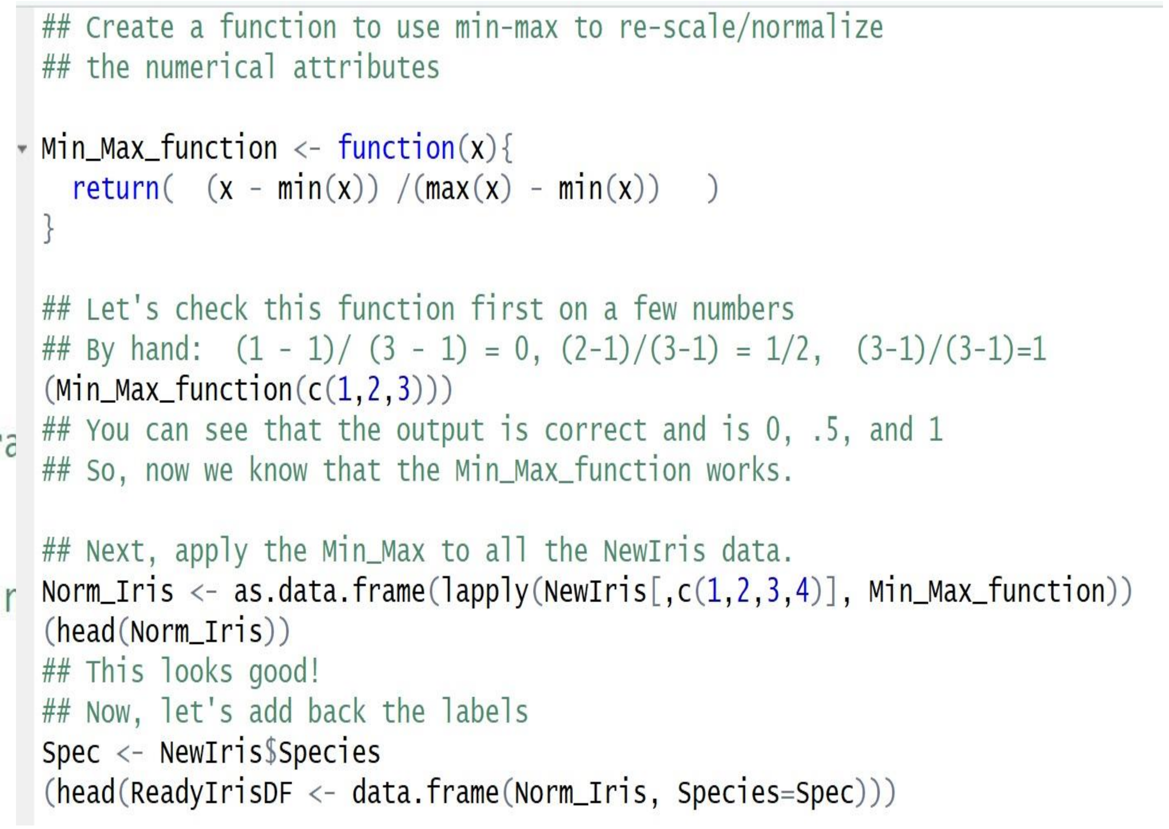
RANDOM FOREST

Gates

THE DATASET AND THE TESTING AND TRAINING SETS: NORMALIZED



```
data(iris)
str(iris)
# Species is the label
table(iris$Species)
(head(iris))

## Because we are doing kNN - we will shuffle (mix up) the
## rows of the dataset. I will do this randomly.

set.seed(9850) ## setting a seed will allow you to reproduce ra
## Create 150 random numbers between 0 and 1
(u_num <- runif(nrow(iris)))
## Use these random numbers (u_num) to shuffle the iris rows ir
## a new data frame. Order by u_num - so random order.
NewIris <- iris[order(u_num),]
(head(NewIris, n=15))
```



```
## Create a function to use min-max to re-scale/normalize
## the numerical attributes

Min_Max_function <- function(x){
  return(  (x - min(x)) /(max(x) - min(x))   )
}

## Let's check this function first on a few numbers
## By hand:  (1 - 1)/ (3 - 1) = 0, (2-1)/(3-1) = 1/2,  (3-1)/(3-1)=1
(Min_Max_function(c(1,2,3)))
## You can see that the output is correct and is 0, .5, and 1
## So, now we know that the Min_Max_function works.

## Next, apply the Min_Max to all the NewIris data.
Norm_Iris <- as.data.frame(lapply(NewIris[,c(1,2,3,4)], Min_Max_function))
(head(Norm_Iris))
## This looks good!
## Now, let's add back the labels
Spec <- NewIris$Species
(head(ReadyIrisDF <- data.frame(Norm_Iris, Species=Spec)))
```

## ABOUT RF

## Ensemble Learning is a type of Supervised Learning Technique.

## We generate multiple Models on a training dataset and

## combine (average) their Output Rules

## to generate a stronger Model

## RF is an emsemble of Decision Trees (DT)

## Averaging the DTs helps to reduce the variance

## and improve the performance of

## This also helps to avoid overfitting.

## We will again use the training and testing set form above.

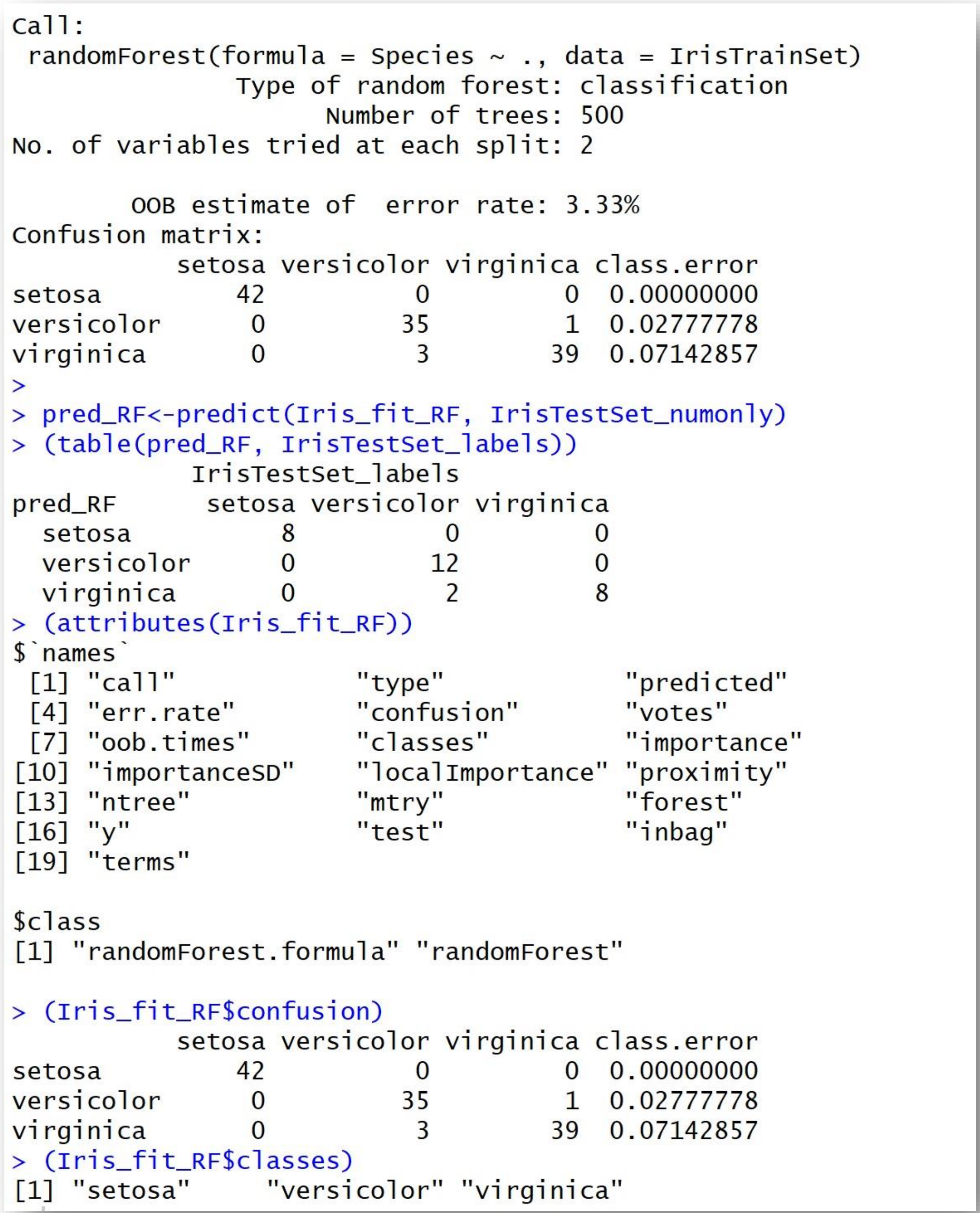
## To save time, when you plan to try out many ML methods

## such as SVM, kNN, RF, DT, etc....create a clean Training

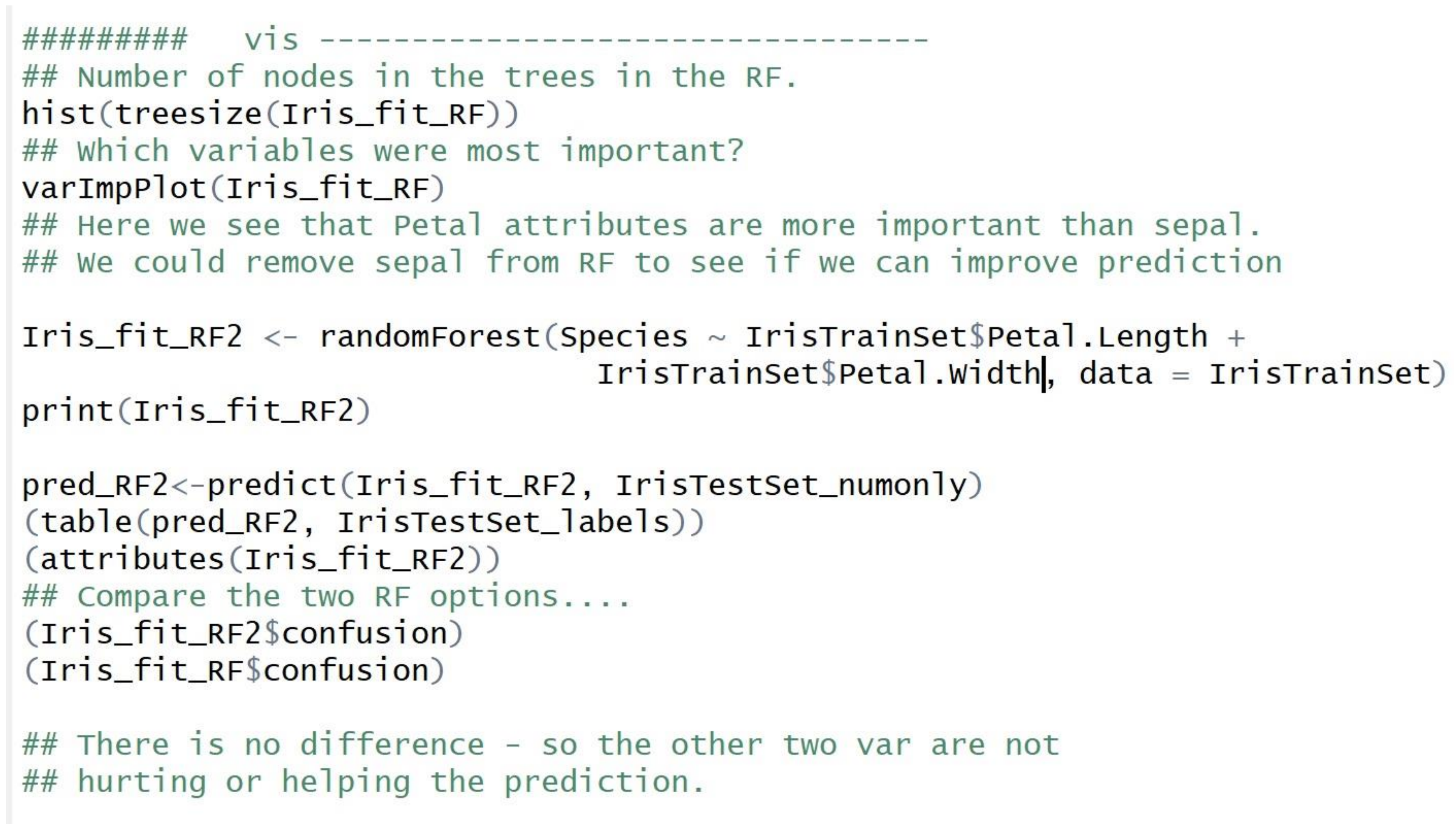
## and Testing set and then you can use the same sets to experiment

## with all the methods.

```
#######  Set up Random Forest ------------------

Iris_fit_RF <- randomForest(Species ~ . , data = IrisTrainSet)
print(Iris_fit_RF)

pred_RF<-predict(Iris_fit_RF, IrisTestSet_numonly)
(table(pred_RF, IrisTestSet_labels))
(attributes(Iris_fit_RF))
(Iris_fit_RF$confusion)
(Iris_fit_RF$classes)
```

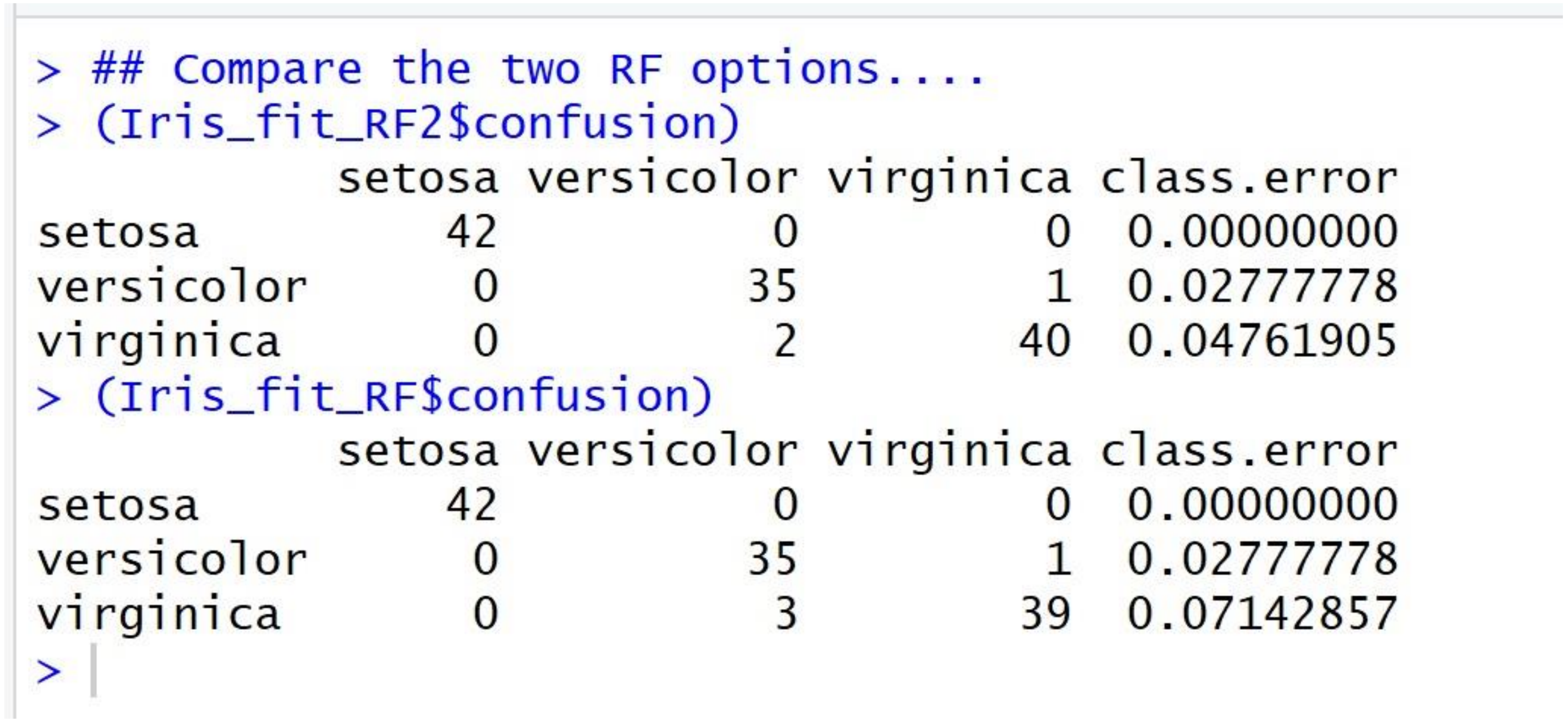


```
Call:
 randomForest(formula = Species ~ ., data = IrisTrainSet)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.33%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         42          0         0  0.00000000
versicolor      0         35         1  0.02777778
virginica       0          3        39  0.07142857
>
> pred_RF<-predict(Iris_fit_RF, IrisTestSet_numonly)
> (table(pred_RF, IrisTestSet_labels))
            IrisTestSet_labels
pred_RF      setosa versicolor virginica
  setosa          8          0         0
  versicolor      0         12         0
  virginica       0          2         8
> (attributes(Iris_fit_RF))
$`names`
 [1] "call"            "type"            "predicted"
 [4] "err.rate"        "confusion"       "votes"
 [7] "oob.times"       "classes"         "importance"
[10] "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"
[16] "y"               "test"            "inbag"
[19] "terms"

$class
[1] "randomForest.formula" "randomForest"

> (Iris_fit_RF$confusion)
           setosa versicolor virginica class.error
setosa         42          0         0  0.00000000
versicolor      0         35         1  0.02777778
virginica       0          3        39  0.07142857
> (Iris_fit_RF$classes)
[1] "setosa"     "versicolor" "virginica"
```



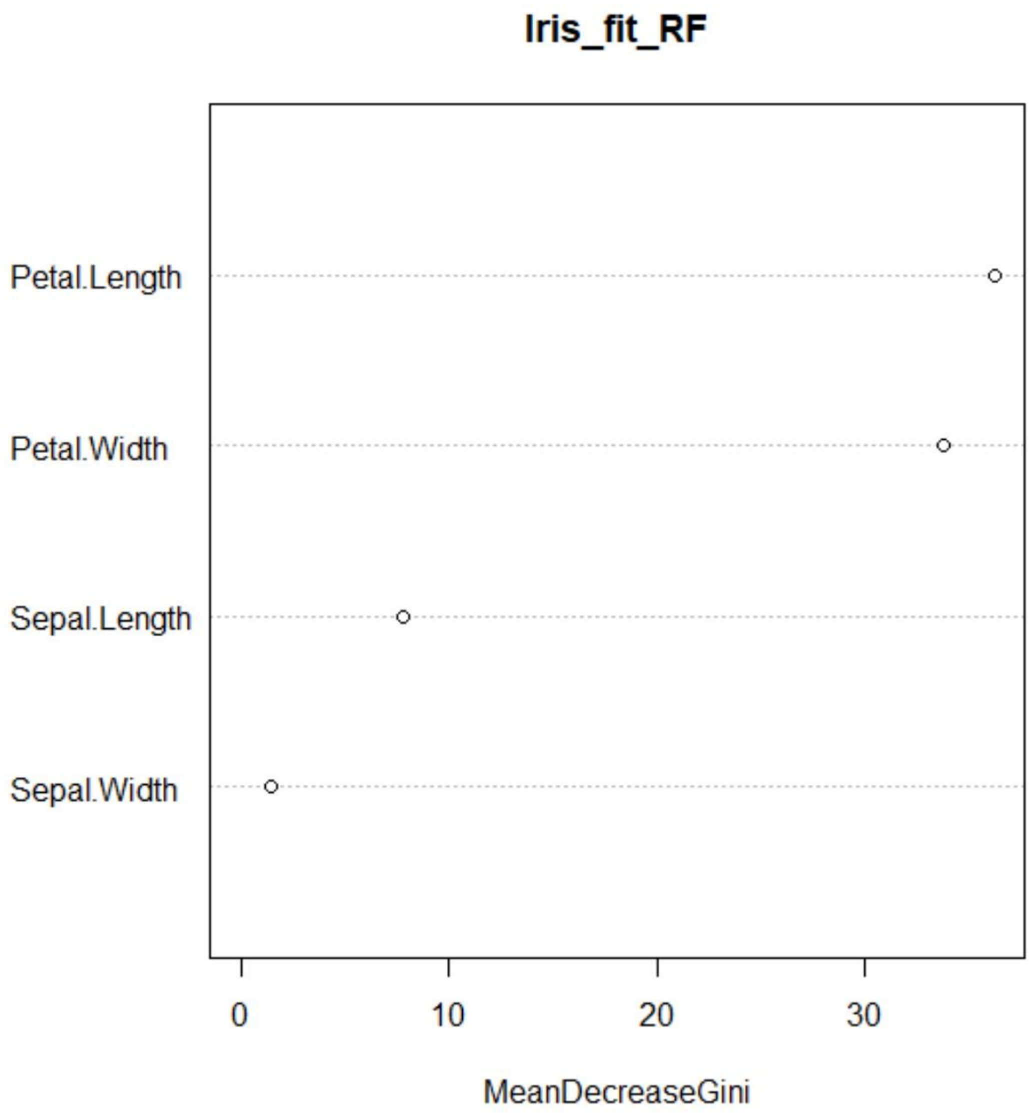
```
#########   vis ---------------------------------
## Number of nodes in the trees in the RF.
hist(treesize(Iris_fit_RF))
## Which variables were most important?
varImpPlot(Iris_fit_RF)
## Here we see that Petal attributes are more important than sepal.
## We could remove sepal from RF to see if we can improve prediction

Iris_fit_RF2 <- randomForest(Species ~ IrisTrainSet$Petal.Length +
                               IrisTrainSet$Petal.Width, data = IrisTrainSet)
print(Iris_fit_RF2)

pred_RF2<-predict(Iris_fit_RF2, IrisTestSet_numonly)
(table(pred_RF2, IrisTestSet_labels))
(attributes(Iris_fit_RF2))
## Compare the two RF options....
(Iris_fit_RF2$confusion)
(Iris_fit_RF$confusion)

## There is no difference - so the other two var are not
## hurting or helping the prediction.
```

RESULTS 1



```
> ## Compare the two RF options....
> (Iris_fit_RF2$confusion)
           setosa versicolor virginica class.error
setosa         42          0         0  0.00000000
versicolor      0         35         1  0.02777778
virginica       0          2        40  0.04761905
> (Iris_fit_RF$confusion)
           setosa versicolor virginica class.error
setosa         42          0         0  0.00000000
versicolor      0         35         1  0.02777778
virginica       0          3        39  0.07142857
> 
```

RESULTS 2


Iris_fit_RF
Petal.Length
Petal.Width
Sepal.Length
Sepal.Width
0
10
20
30
MeanDecreaseGini

PERFORMANCE ANALYSIS

Gates

METRICS FOR PERFORMANCE EVALUATION: CONFUSION MATRIX

Confusion Matrix:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	True Positive	False Negative
	Class=No	False Positive	True Negative

IS ACCURACY ALWAYS A GOOD MEASURE?
CAN YOU THINK OF AN EXAMPLE WHEN IT IS NOT?

	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

COST-SENSITIVE MEASURES

$$\text{Precision (p)} = \frac{TP}{TP+FN}$$

Measure of Sensitivity

$$\text{Recall (r)} = \frac{TP}{TP+FP}$$

Measure of Specificity

$$\text{F-measure (F)} = \frac{2rp}{r+p} = \frac{2TP}{2TP+FP+TN}$$

DECISION TREES

ML Topic 1

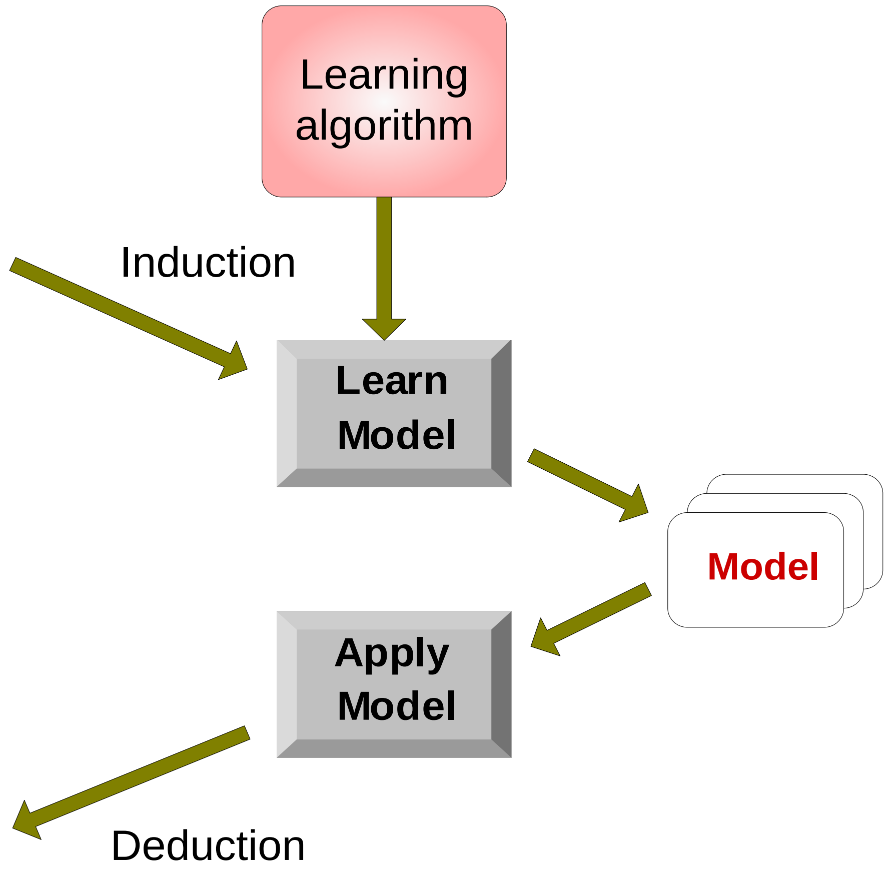
ILLUSTRATING A CLASSIFICATION TASK

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



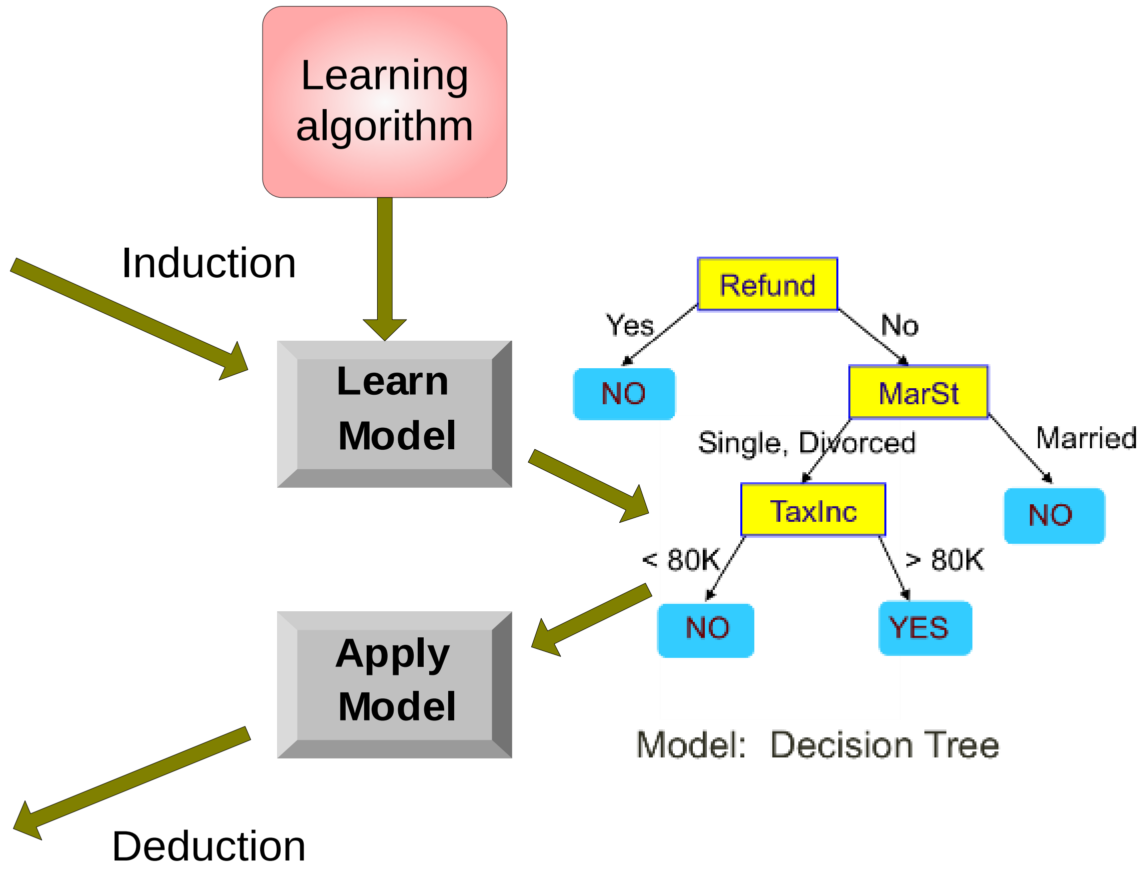
EXAMPLE – Decision Tree: Is Someone Cheating on Their Taxes?

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

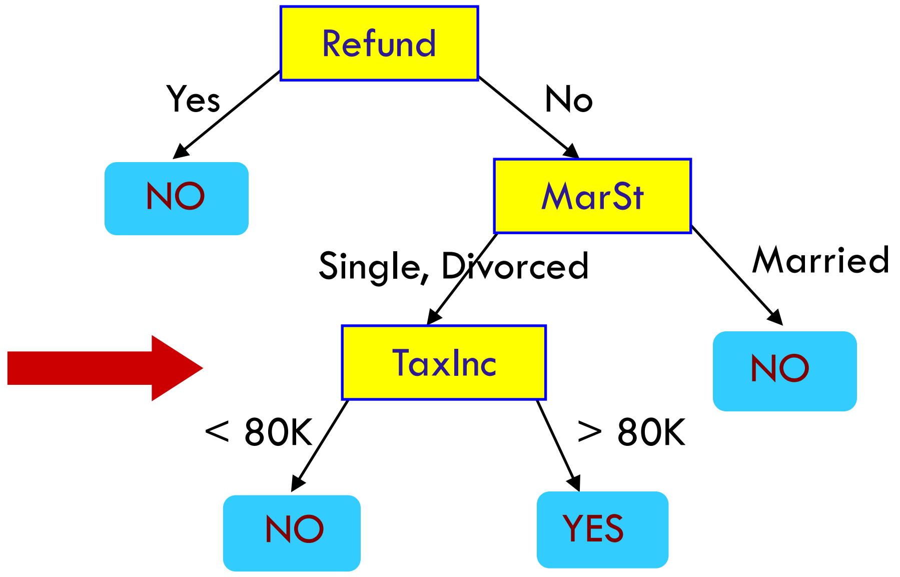
Test Set



EXAMPLE OF TRAINING DATA AND DECISION TREE MODEL

Tid	Refund categorical	Marital Status categorical	Taxable Income continuous	Cheat class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

DECISION TREE OVERVIEW

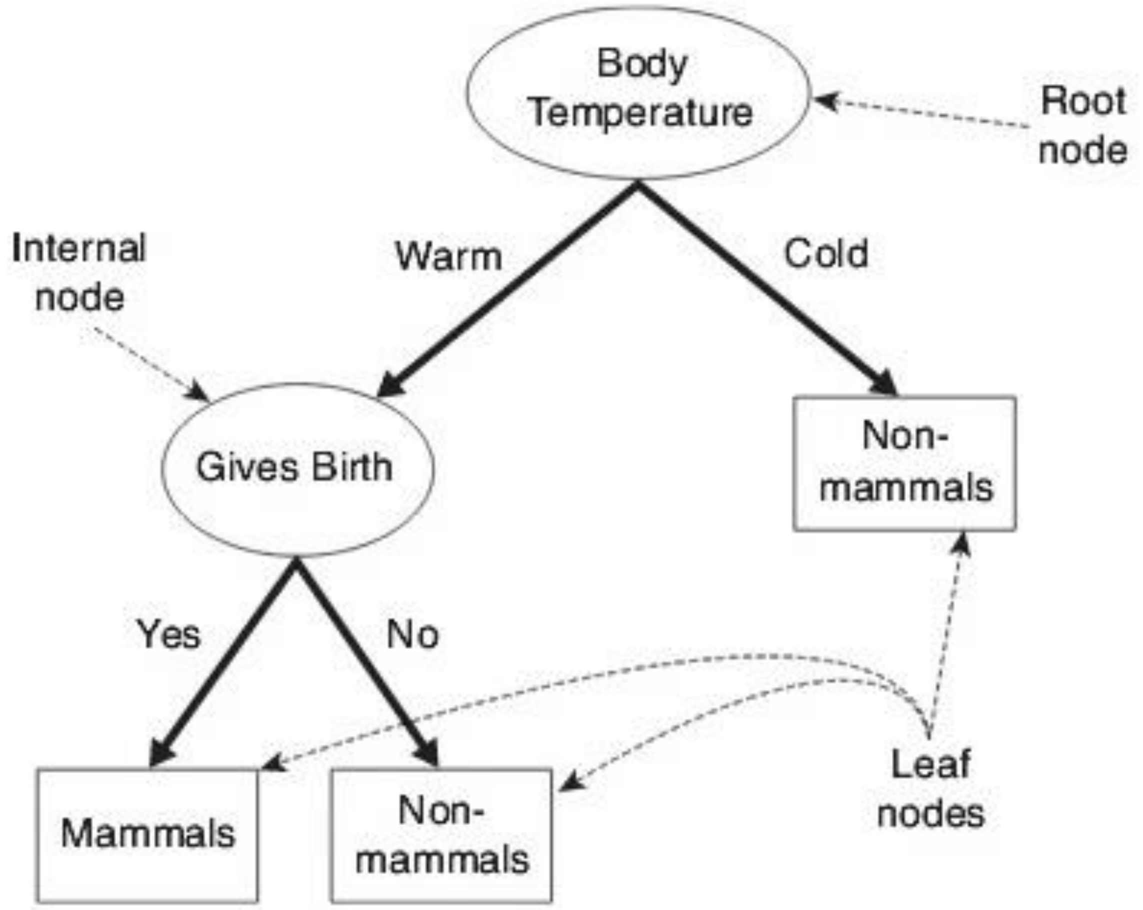
Build a **classifier that is a directional tree structure.**

The tree has

- **Root Node**: no incoming edges and zero or more outgoing edges. (contains attribute test condition(s))
- **Internal Nodes**: Exactly ONE incoming edge and TWO or more outgoing. (contains attribute test condition(s))
- **Leaf/terminal Nodes**: ONE incoming, no outgoing.

Each leaf node is assigned a class label.

EXAMPLE



http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf, page 7

BUILDING A DECISION TREE

There are an **infinite** number of possible decision trees that can be constructed from a set of attributes.

Finding the **optimal tree** is an **intractable** problem as the search space is exponential.

Algorithms can find "good" decision trees using the **Greedy** approach – they make a series of **locally optimal** decisions.

Example: **Hunt's Algorithm**

- **Hunt is the basis of ID3, C4.5, and CART**

HUNT’S ALGORITHM: DECISION TREE

Assumptions:

Let D_t be the set of **training records**, associated with node t in the tree.

Let **xi** be record i such that yi is the class label.

All training records are: {**x**1, **x**2, …, **x**n} with associated class labels

{ y1, y2, …, yn}

Method: The tree is created in a **recursive** fashion by continuing to partition the training records (**x**) into purer subsets.

Steps:

1) If all records in D_t belong to the same class y_t, then t is a leaf node labeled as y_t.
2) If D_t contains records that belong to MORE THAN one class, an **attribute test condition** is selected to partition into smaller subsets.
3) The above is recursively repeated

PRACTICE: PREDICT WHETHER A LOAN APPLICANT WILL REPAY THEIR LOAN:

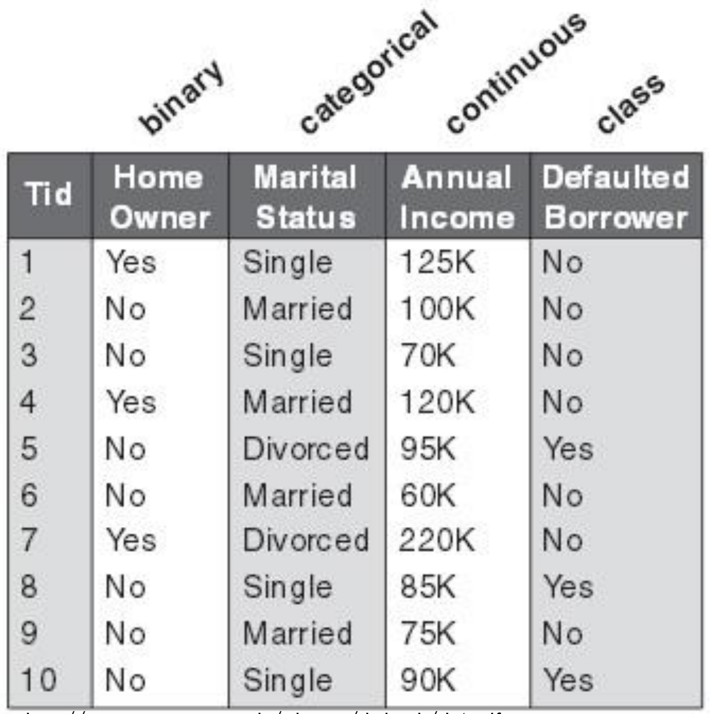
Class label 1
Defaulted = No
Class label 2
Defaulted = Yes

Next, examine a known **training set.**

What are the attributes of each record?

What is the label of each?

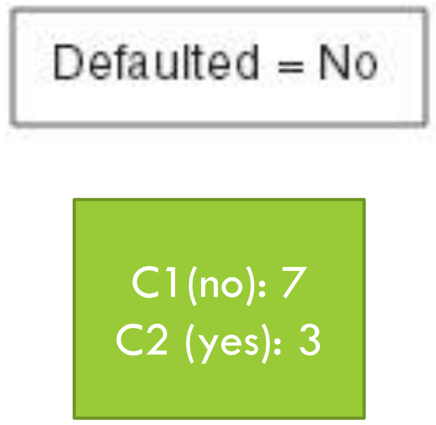
Build a Decision Tree…


binary
categorical
continuous
class

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf

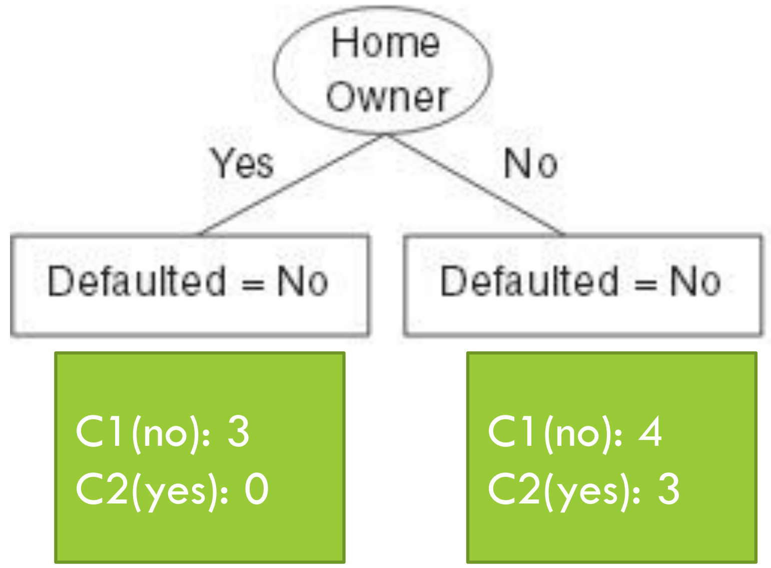
STEP 1



The initial tree is a single node that represents the fact that most borrows did pay and so the majority is default=no.

However, **this current node contains records from both classes** and so **must be further refined (split).**

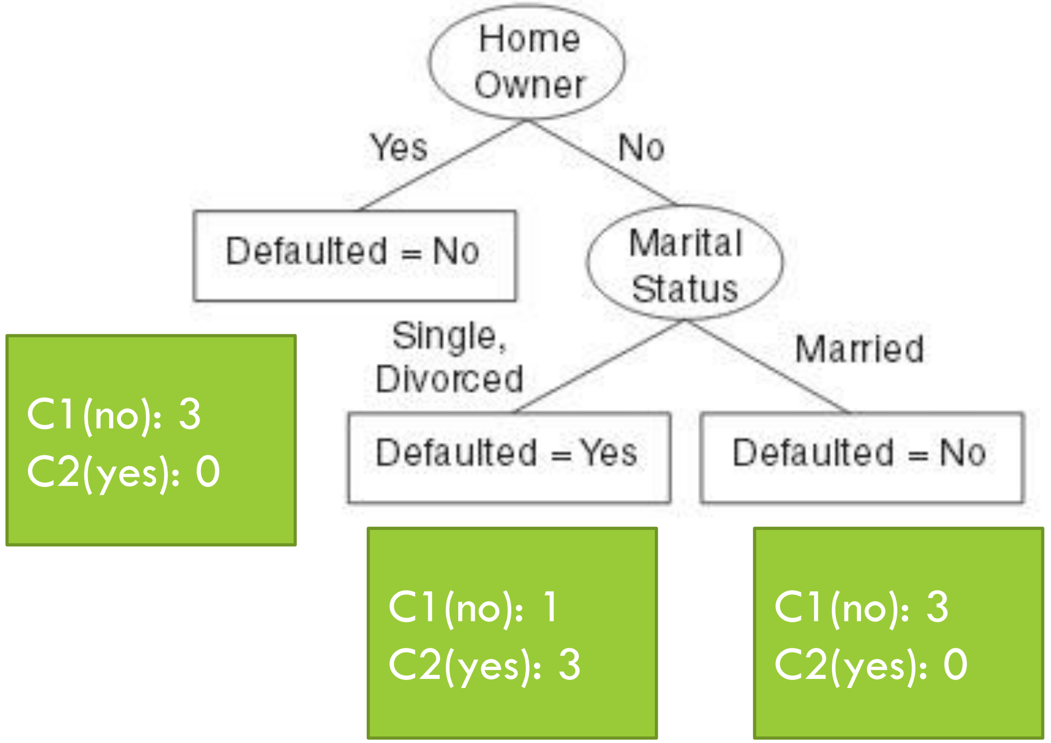
STEP 2



Using the attribute condition test of "Home Owner" still results one mixed class, with the majority of each labeled as Default=NO.

This must be further refined until the node is **pure**.

STEP 3



All "Home Owners" are class: Default=NO. Therefore, that node is pure and does not require further partitioning.

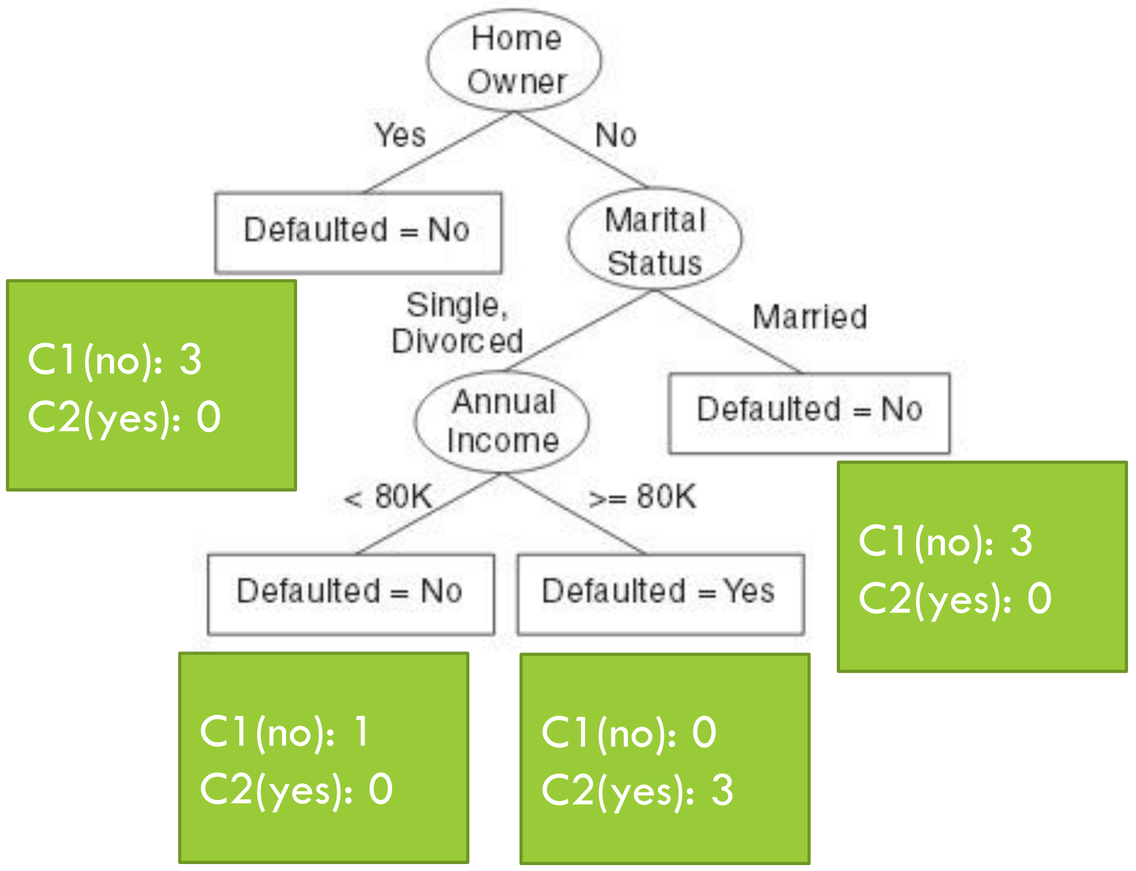
If the borrower is not a home owner, they are further partitioned by Marital Status.

The only node that is still not pure here is "Single/Divorces" AND "not home owner". Another attribute condition must be added.

STEP 4

"Annual Income" is used as an attribute condition with <80K, or >80K.

Now, all nodes are pure and the leaf nodes contain the classification.



Test it!

Suppose a non-married person with 75K per year who does not own a home gets a loan – **<u>will they pay it back?</u>**

ABOUT HUNT

Hunt works well if the training set contains every combination of all possible attributes.

What happens if it does not?

DESIGN ISSUES WITH DECISION TREES

How should training records be split?

- How can the “best” attribute test condition be selected?

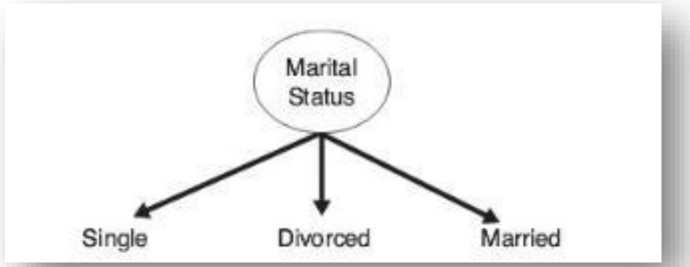
How should the splitting stop?

- One option is to expand a node until all records are in the same class or have identical attribute values.

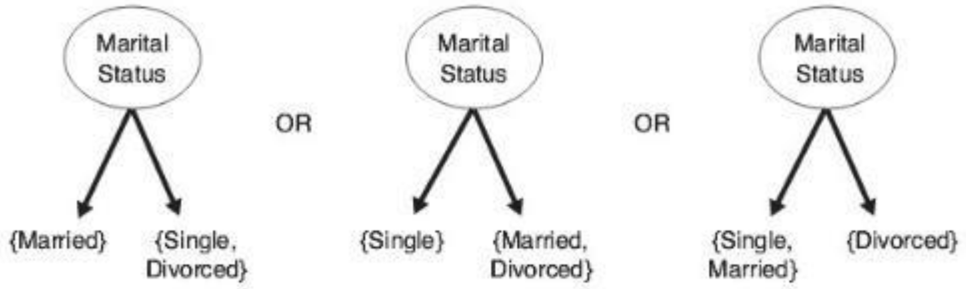
EXPRESSING ATTRIBUTE TEST CONDITIONS:

- **Binary attributes:** two possible outcomes such as married or not married.
- **Nominal Attributes:**
  - Multi-split – one node for each attribute name
  - binary split (CART does this) determined by investigating the best of the $2^k - 1$ options for splitting. (k is the number of attributes)

Multi-split



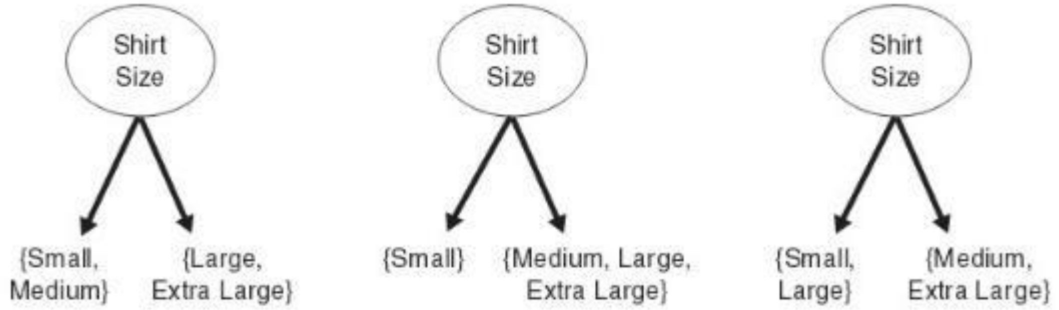
http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf



Binary split options for 3 attributes

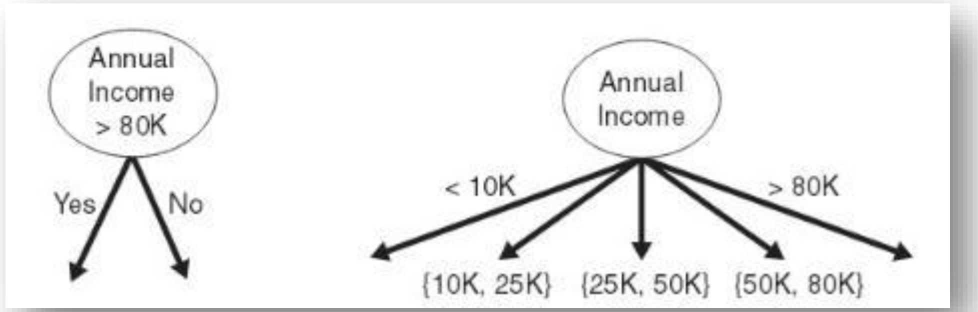
EXPRESSING ATTRIBUTE TEST CONDITIONS:

- **Ordinal attributes:** can also be split using binary or multi.
  - **Why is the last option here not as good?**



Continuous Attributes:

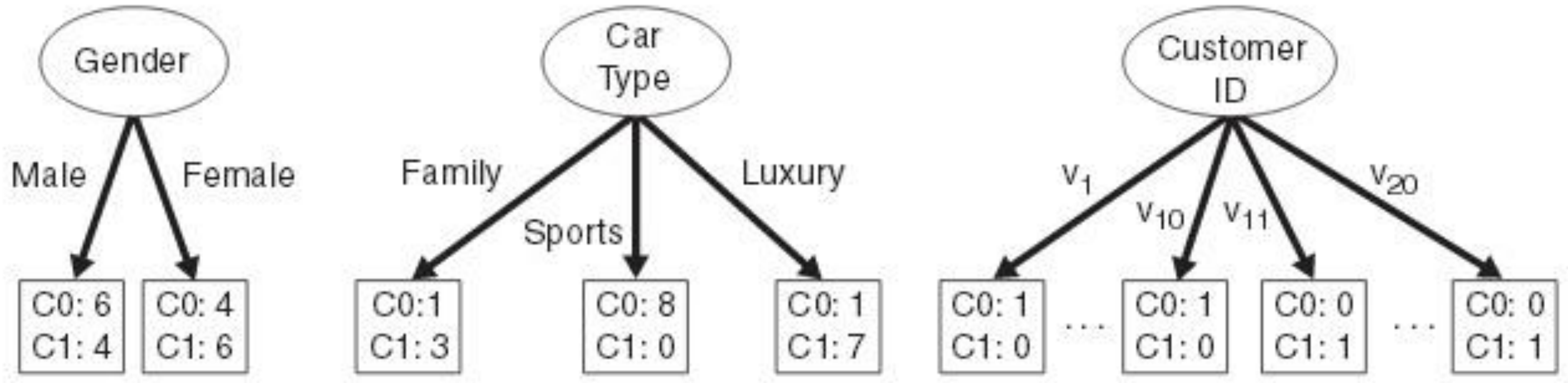
- Can use **comparison set:** (A < x) OR (A >= x)
- Can use a range of options (for mult):



COMPARING SPLITS

Note: C0:6 means that there are 6 records of class "0" in the partition.

Which split created purer classes?



METHODS FOR MEASURING "BEST" SPLIT

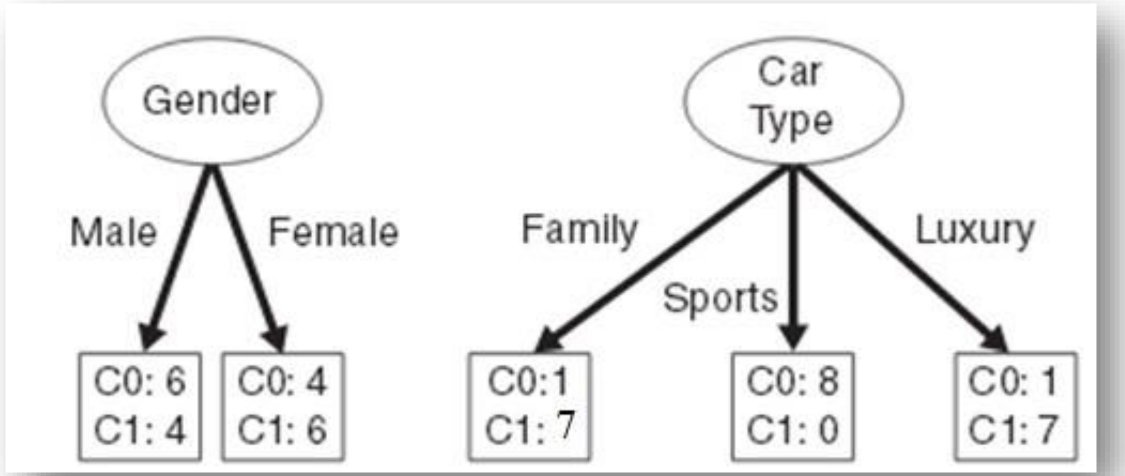
$$\begin{aligned}
\text{Entropy}(t) &= -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t),\\
\text{Gini}(t) &= 1-\sum_{i=0}^{c-1}[p(i|t)]^2,\\
\text{Classification error}(t) &= 1-\max_i[p(i|t)],
\end{aligned}$$

STEP 1: CLASS NODE PROBABILITY

Let p(i|t) be the fraction of records belonging to class i at a given node t.

For the Gender Partition:
p(C0|Male) = 6/10
p(C1|Female) = 6/10

For the Car Type partition
p(C0|Family) = 1/8



CALCULATING ENTROPY

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t),$$

Node N_2	Count
Class=0	1
Class=1	5

p(i|t) = p(class 0 | node N2) = 1/6 (recall that i is the class)
p(i|t) = p(class 1 | node N2) = 5/6

Negative sum over all classes:

entropy = -(1/6)log2(1/6) – (5/6)log2(5/6) = .65

Entropy ranges from 0 to 1, where 0 is pure and 1 is the worst case.

COMPARISON

Node N_1	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$$
$$\text{Entropy} = -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$$
$$\text{Error} = 1 - \max[0/6, 6/6] = 0$$

Node N_2	Count
Class=0	1
Class=1	5

$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$
$$\text{Entropy} = -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$$
$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$$
$$\text{Entropy} = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$$
$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

1) which has lowest impurity?
2) Which has highest impurity?

INTUITION:

If a partition results in p(i|t) = .5, it is very poor. Why?

If a partition results in p(i|t) = 0, it is pure. Why?

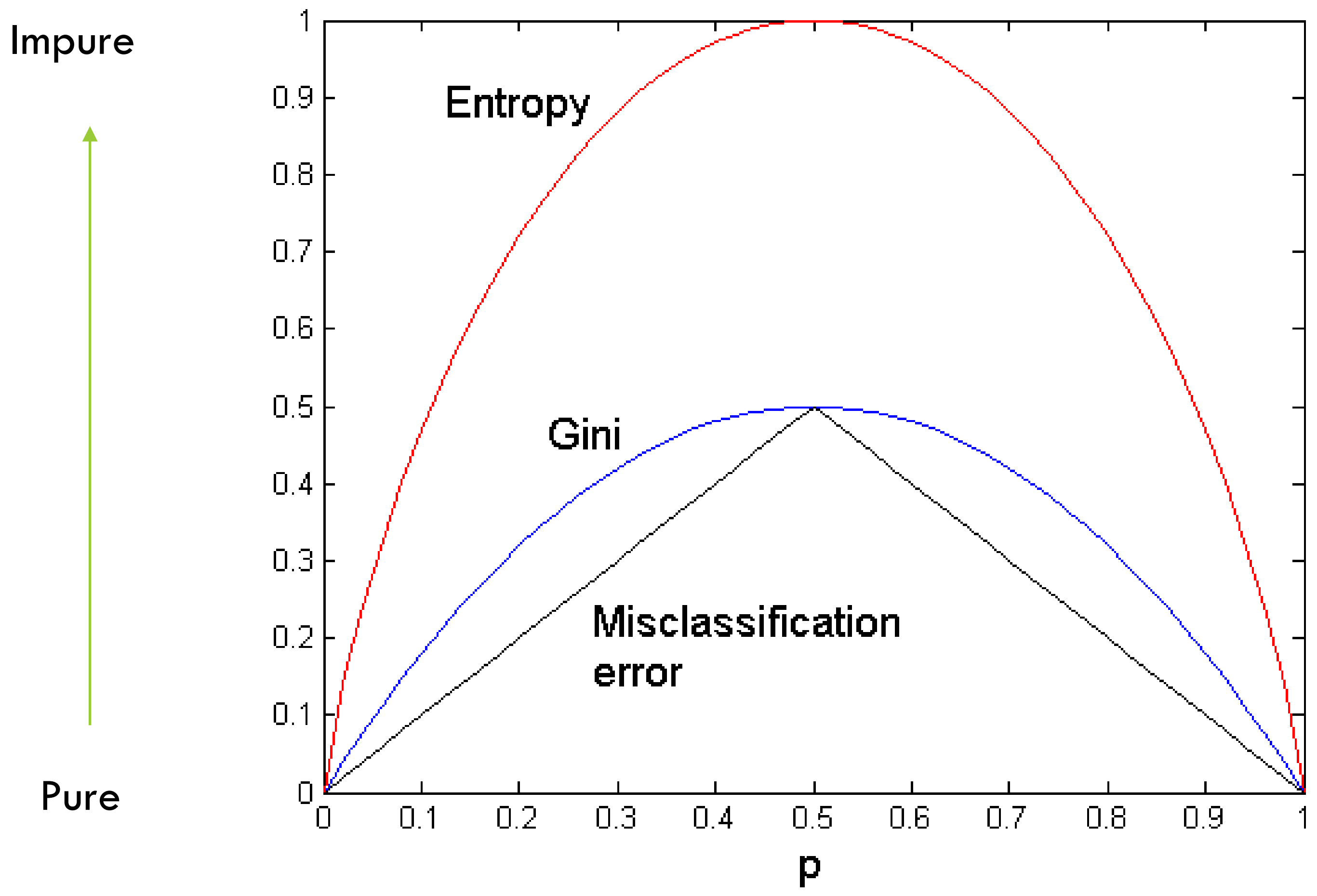
The lower the impurity of the partition (so the more pure it is), the more skewed the class distribution. Why?

A node with class distribution of C1:0, C2:10 is skewed and pure.

A node distribution with C1:5 and C2:5 is has the highest impurity an no skew.

COMPARISON AMONG SPLITTING CRITERIA

For a 2-class problem:



INFORMATION GAIN

To determine the **strength of a partition** – compare purity of parent node (before split) to child nodes (after split).

The greater the difference – the better the partition condition.

The **Gain** (Δ) is a measure for **goodness of split.**

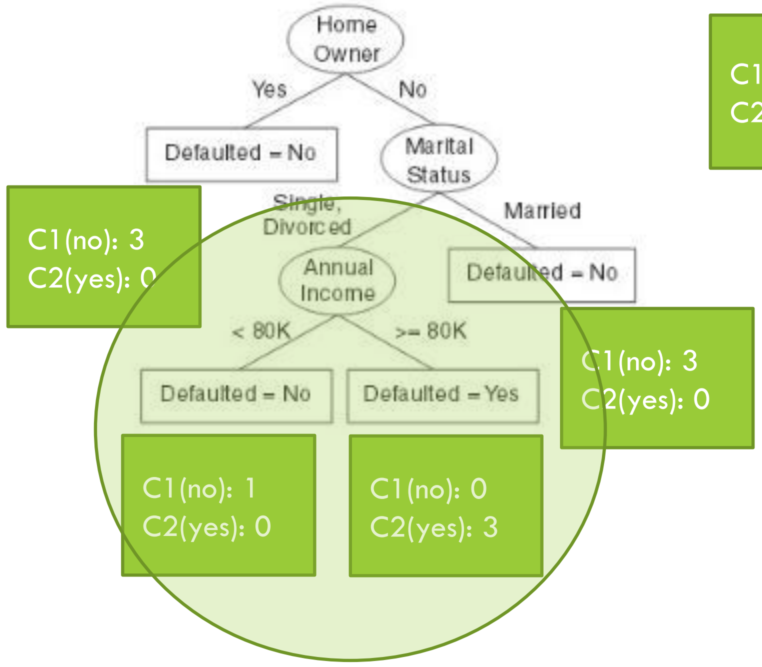
I is the **impurity measure** of a node, N is the number of records at parent node, k is the number of attribute values. N(vj) is the number of records in child vj.

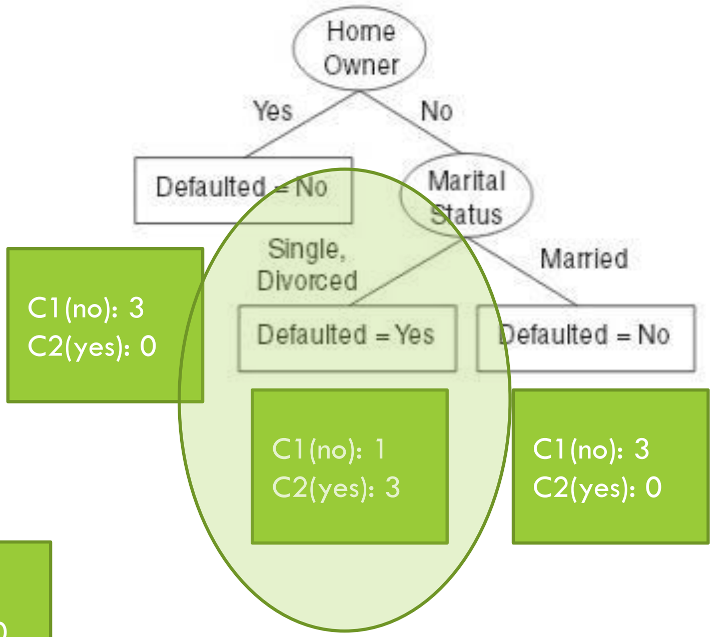
If entropy is used as the impurity measure, the difference in entropy is the **information gain.**

This method is used in **ID3**

$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j),$$

EXAMPLE: INFORMATION GAIN USING ENTROPY AS THE PURITY MEASURE





$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j),$$

CALCULATIONS FOR INFORMATION GAIN USING ENTROPY

Entropy for Parent =

-(1/4)log(1/4) - (3/4)log(3/4) =

-(1/4)(-2) - (1/4)(-.415)= .604

Entropy for left node =

-(1/1)log(1/1) - (0/1)log(0/1) =

0 - 0 = 0

Entropy for right node =

-(0/3)log(0/3) - (3/3)log(3/3) =

0 - 0 = 0

Information GAIN:
I(Parent) - sum over all children N(v)/N * I(v) =
.604 - (1)/(4) * 0 - (3)/(4)*0 = .604
This is the max possible difference and so is the best partition.
N is the num records at parent
k is the num attribute values (ours has two possible values)
N(v) is the num of records in child
I in this case is the entropy

The greater the difference between I(Parent) and children – the better the partition condition.

SPLITTING BASED ON INFORMATION GAIN

Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions

n_i is the number of records in partition i

- Used in C4.5

DECISION TREE BASED CLASSIFICATION

Advantages:

- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees
- Robust for missing values
- Redundant attributes do not adversely affect accuracy of prediction
- Accuracy is comparable to other classification techniques for many simple data sets

DECISION TREE ISSUES

Choosing Splitting Attributes

Ordering of Splitting Attributes

Tree Structure

Stopping Criteria

Training Data

Pruning

OCCAM'S RAZOR

Given two models of similar generalization errors, one should **prefer the simpler model over the more complex model**

For complex models, there is a greater chance that it was fitted accidentally by errors in data

Therefore, one should include model complexity when evaluating a model

LINK TO CODE:

https://drive.google.com/drive/folders/1rXm4jTHMTTjFvHfJ3daCCWNmOdF9tNPI?
usp=sharing