# Machine Learning

Ajay Ohri
Dec 2017

# Big Data: Hadoop Stack with Mahout

https://mahout.apache.org/

## The Apache Mahout™ project's goal is to build an environment for quickly creating scalable performant machine learning applications.

**Apache Mahout Samsara Environment includes**

- Distributed Algebraic optimizer
- R-Like DSL Scala API
- Linear algebra operations
- Ops are extensions to Scala
- IScala REPL based interactive shell
- Integrates with compatible libraries like MLLib
- Runs on distributed Spark, H2O, and Flink

**Apache Mahout Samsara Algorithms included**

- Stochastic Singular Value Decomposition (ssvd, dssvd)
- Stochastic Principal Component Analysis (spca, dspca)

# Machine Learning

Machine learning concerns the construction and study of systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between spam and non-spam messages

**Supervised learning** is the machine learning task of inferring a function from labeled training data.[1] The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*).

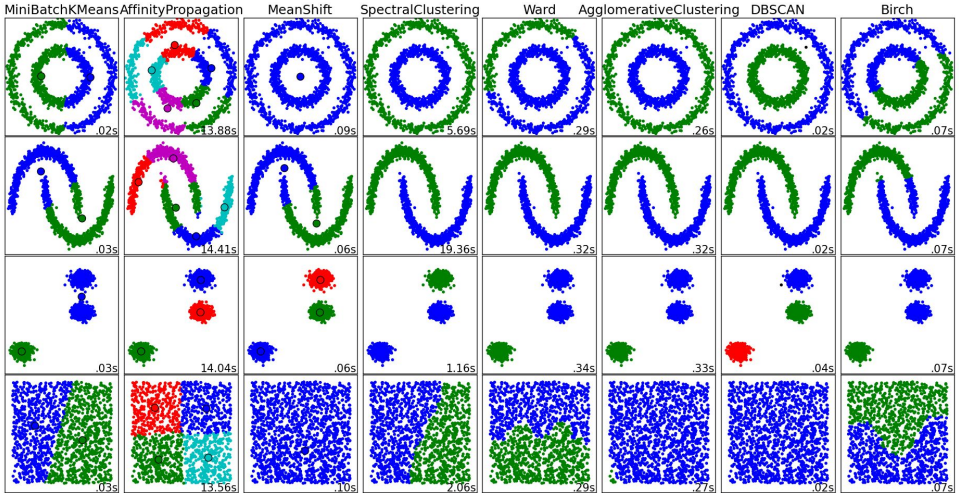
In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available.

In machine learning, the problem of **unsupervised learning** is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning

# Sci-kit learn

http://scikit-learn.org/stable/

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.




# Supervised Learning

**The problem solved in supervised learning**

Supervised learning consists in learning the link between two datasets: the observed data `X` and an external variable `y` that we are trying to predict, usually called “target” or “labels”. Most often, `y` is a 1D array of length `n_samples`.

All supervised estimators in scikit-learn implement a `fit(X, y)` method to fit the model and a `predict(X)` method that, given unlabeled observations `X`, returns the predicted labels `y`

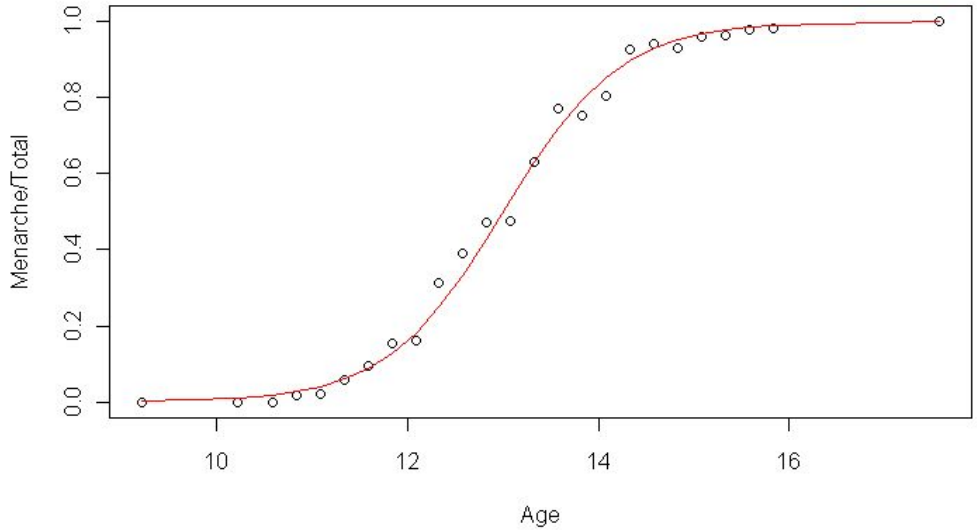
**Vocabulary: classification and regression**

If the prediction task is to classify the observations in a set of finite labels, in other words to “name” the objects observed, the task is said to be a **classification** task. On the other hand, if the goal is to predict a continuous target variable, it is said to be a **regression** task.When doing classification in scikit-learn, `y` is a vector of integers or strings.

http://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html

# Logistic Regression Models

Source https://ww2.coastal.edu/kingw/statistics/R-tutorials/logistic.html



# Logistic Regression Models

Logistic Regression $\ln(p/1-p) = a + bX$

Reading

https://ww2.coastal.edu/kingw/statistics/R-tutorials/logistic.html

https://cran.r-project.org/web/packages/HSAUR/vignettes/Ch_logistic_regression_glm.pdf

# Logistic Regression Models

Logistic Regression  ln(p/1-p) =a +bX

http://www.statmethods.net/advstats/glm.html

**glm(***formula*, **family=***familytype*(**link=***linkfunction*), **data=)**

| Family | Default Link Function |
|---|---|
| binomial | (link = "logit") |
| gaussian | (link = "identity") |
| Gamma | (link = "inverse") |
| inverse.gaussian | (link = "1/mu^2") |
| poisson | (link = "log") |
| quasi | (link = "identity", variance = "constant") |
| quasibinomial | (link = "logit") |
| quasipoisson | (link = "log") |

See **help(glm)** for other modeling options. See **help(family)** for other allowable link functions for each family. Three subtypes of generalized linear models will be covered here: logistic regression, poisson regression, and survival analysis.

## Logistic Regression

Logistic regression is useful when you are predicting a binary outcome from a set of continuous predictor variables. It is frequently preferred over discriminant function analysis because of its less restrictive assumptions.

```
# Logistic Regression
# where F is a binary factor and
# x1-x3 are continuous predictors
fit <- glm(F~x1+x2+x3,data=mydata,family=binomial())
summary(fit) # display results
confint(fit) # 95% CI for the coefficients
exp(coef(fit)) # exponentiated coefficients
exp(confint(fit)) # 95% CI for exponentiated coefficients
predict(fit, type="response") # predicted values
residuals(fit, type="deviance") # residuals
```

# Case Study

## Menarche

```
library("MASS")
 data(menarche)
summary(menarche)
glm.out = glm(cbind(Menarche, Total-Menarche) ~ Age, family=binomial(logit),
  data=menarche)
```

https://ww2.coastal.edu/kingw/statistics/R-tutorials/logistic.html

# Case Study

## Menarche

plot(Menarche/Total ~ Age, data=menarche)

lines(menarche$Age, glm.out$fitted, type="l", col="red")

title(main="Menarche Data with Fitted Logistic Regression Line")

summary(glm.out)

AIC - the smaller the AIC or BIC, the better the fit.

https://ww2.coastal.edu/kingw/statistics/R-tutorials/logistic.html

# Case Study

## College Admissions

```
library(aod)
  library(ggplot2)
  library(Rcpp)

mydata <- read.csv("http://www.ats.ucla.edu/stat/data/binary.csv")

head(mydata)

summary(mydata)
```

http://www.ats.ucla.edu/stat/r/dae/logit.htm

# Case Study

## College Admissions

```
sapply(mydata, sd)

xtabs(~ admit + rank, data = mydata)

mydata$rank <- factor(mydata$rank)
  mylogit <- glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")
```

http://www.ats.ucla.edu/stat/r/dae/logit.htm

# Case Study

## mtcars

```
am.glm = glm(formula=am ~ hp + wt,
+            data=mtcars,
+            family=binomial)
summary(am.glm)
newdata = data.frame(hp=120, wt=2.8)
predict(am.glm, newdata, type="response")
```

p-values of the hp and wt variables are both less than 0.05, neither hp or wt is insignificant in the logistic regression model.

http://www.r-tutor.com/elementary-statistics/logistic-regression/estimated-logistic-regression-equation

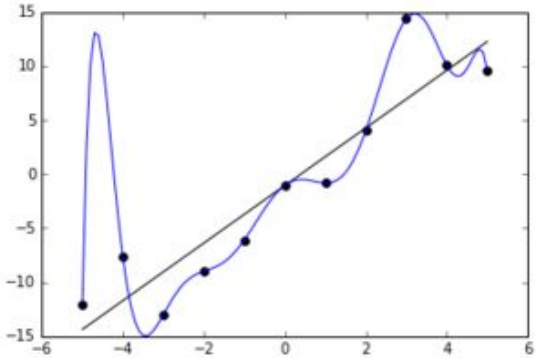
help(predict.glm)

# Some Additional Concepts

Odds ratio =  p/1-p where p is probability of success

Logit = log (odds ratio)

Overfitting -It occurs when the model is closer to sample data than real data and it occurs due to excessive noise. It is avoided by splitting the data into test and training, and then building the model on one part of data



# Some Additional Concepts

ROC Curve and AUC

The ROC is a curve generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings while the AUC is the area under the ROC curve. As a rule of thumb, a model with good predictive ability should have an AUC closer to 1 (1 is ideal) than to 0.5.

Confusion Matrix helps determine classifier

| n=165 | **Predicted: NO** | **Predicted: YES** |
|---|---|---|
| **Actual: NO** | 50 | 10 |
| **Actual: YES** | 5 | 100 |

# Logistic Regression in R


## CLASS EXERCISE - German Credit

file:///C:/Users/KOGENTIX/Desktop/training%20offsite/pythonfordatascience_logistic%20regression%20-%20script%20for%20ppt.R%20at%20master%20%C2%B7%20decisionstats_pythonfordatascience.html

# Logistic Regression in R

122 lines (84 sloc) | 4.32 KB Raw Blame History

```
1  library(caret)
2  data(GermanCredit)
3  head(GermanCredit)
4  str(GermanCredit)
5  summary(GermanCredit)
6
7  # Converting Class variable from Bad/Good to 0/1
8
9  GermanCredit$Class=ifelse(GermanCredit$Class=="Bad",0,1)
10 GermanCredit$Class[1:5]
11
12
13
14
15 # We will split our dataset into Train and Test data.
16 #The Train dataset will be used to build the model and
17 #the Test dataset will used to validate how well our model is performing.
18
19 set.seed(1234)
20 inTrain=createDataPartition(GermanCredit$Class, p=0.7, list=FALSE)
21 inTrain
22 Training=GermanCredit[inTrain,]
23 Testing=GermanCredit[-inTrain,]
24 dim(Training)
25 dim(Testing)
26
27
28 summary(Training$Class)
29 sum(Training$Class)
30 sum(Testing$Class)
```

# kNN

http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

In pattern recognition, the ***k*-Nearest Neighbors algorithm** (or ***k*-NN** for short) is a non-parametric method used for Classification and regression.[1] In both cases, the input consists of the *k* closest training examples in the feature space. The output
depends on whether *k*-NN is used for classification or regression:

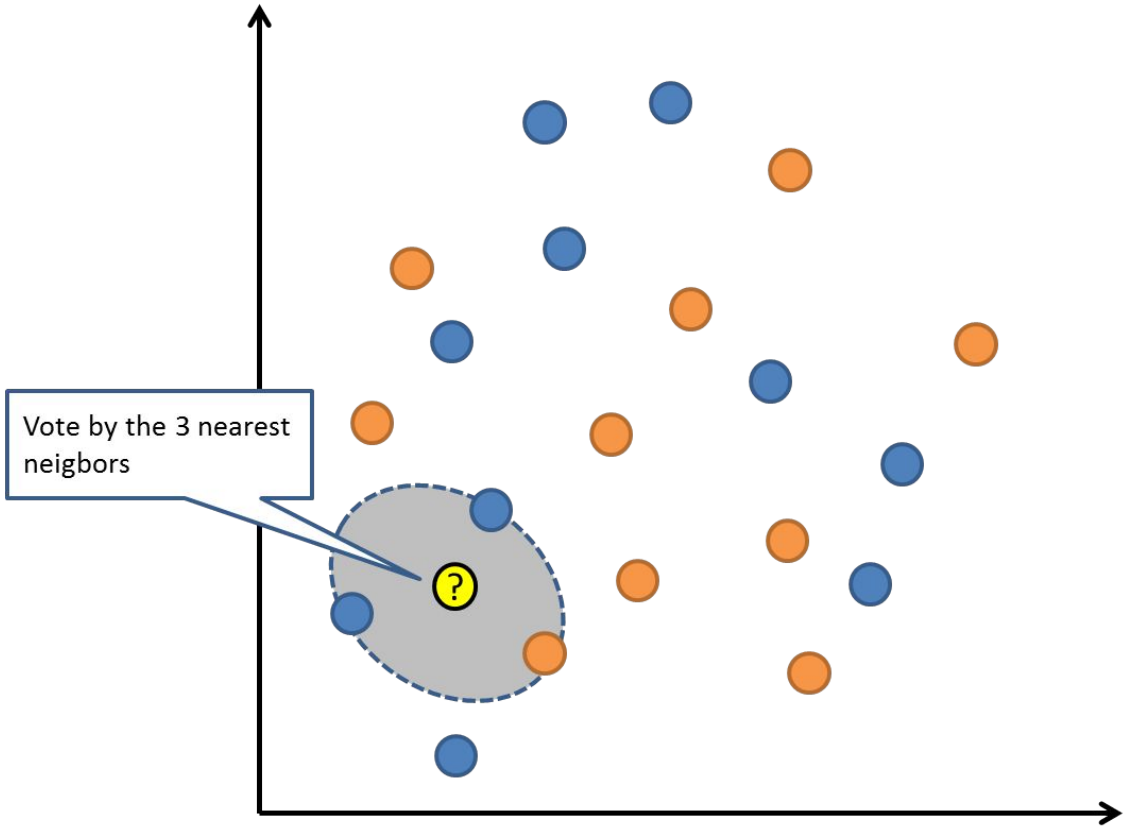
- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If*k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

knn() function in the class package in R
http://cran.r-project.org/web/packages/class/index.html

In Python The **sklearn.neighbors** module implements the k-nearest neighbors algorithm. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these.

# kNN



# kNN

```
In [53]: type(iris_y_train)

Out[53]: numpy.ndarray

In [18]: # Create and fit a nearest-neighbor classifier
         from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier()
         knn.fit(iris_X_train, iris_y_train)

Out[18]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                    weights='uniform')

In [20]: knn.predict(iris_X_test)

Out[20]: array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])

In [21]: iris_y_test

Out[21]: array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

# Naive Bayes

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}.$$

A **naive Bayes classifier** is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions.  In R

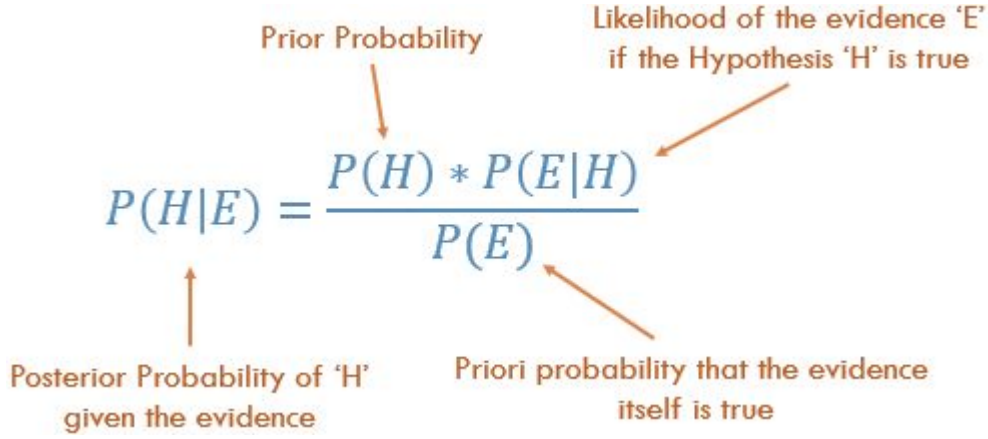
http://cran.r-project.org/web/packages/e1071/index.html naiveBayes() function is used

http://cran.r-project.org/web/packages/klaR/index.html NaiveBayes() function is used

In Python

http://scikit-learn.org/stable/modules/naive_bayes.html

We use sklearn.naive_bayes



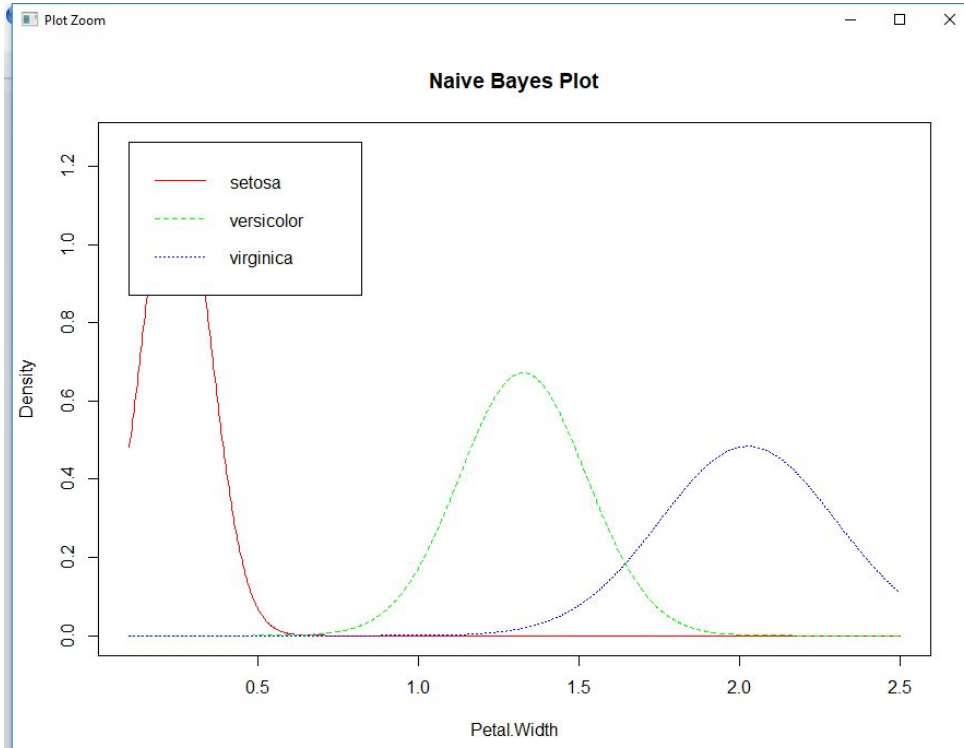
# Naive Bayes

```
rm(list=ls())

library(caret)

data(iris)

x=iris[1:4]

y=iris$Species

model = train(x,y,'nb',trControl=trainControl(method='cv',number=10))

model

str(model)

predict(model$finalModel,x)

table(predict(model$finalModel,x)$class,y)
```

```
>
> table(predict(model$finalModel,x)$class,y)
            y
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          3        47
>
```

# Naive Bayes

```
naive_iris <- NaiveBayes(iris$Species ~ ., data = iris)

plot(naive_iris)
```



In [12]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

In [17]:
```
gnb.fit(iris.data, iris.target)
```

Out[17]:
```
GaussianNB(priors=None)
```

In [18]:
```
print(gnb.fit)
```

```
<bound method GaussianNB.fit of GaussianNB(priors=None)>
```

In [14]:
```
y_pred = gnb.fit(iris.data, iris.target).predict(iris.data)
y_pred
```

Out[14]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [22]:
```
import numpy as np
import pandas as pd
```

In [24]:
```
dict(pd.Series(iris.target).value_counts())
```

Out[24]:
```
{0: 50, 1: 50, 2: 50}
```

In [23]:
```
dict(pd.Series(y_pred).value_counts())
```

Out[23]:
```
{0: 50, 1: 50, 2: 50}
```

In [26]:
```
iris.data.shape
```

Out[26]: (150, 4)

In [13]:
```
print("Number of mislabeled points out of a total %d points : %d"
%(iris.data.shape[0],(iris.target != y_pred).sum()))
```

```
Number of mislabeled points out of a total 150 points : 6
```

In [40]:
```
from sklearn.metrics import confusion_matrix
```

In [41]:
```
cnf_matrix = confusion_matrix(iris.target, y_pred)
```

In [42]:
```
cnf_matrix
```

Out[42]:
```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  3, 47]])
```

In [48]:
```
from sklearn.preprocessing import label_binarize
```

In [53]:
```
# Binarize the output
y = label_binarize(y, classes=[0, 1, 2])
y
```

Out[53]:
```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
```

# Neural Networks

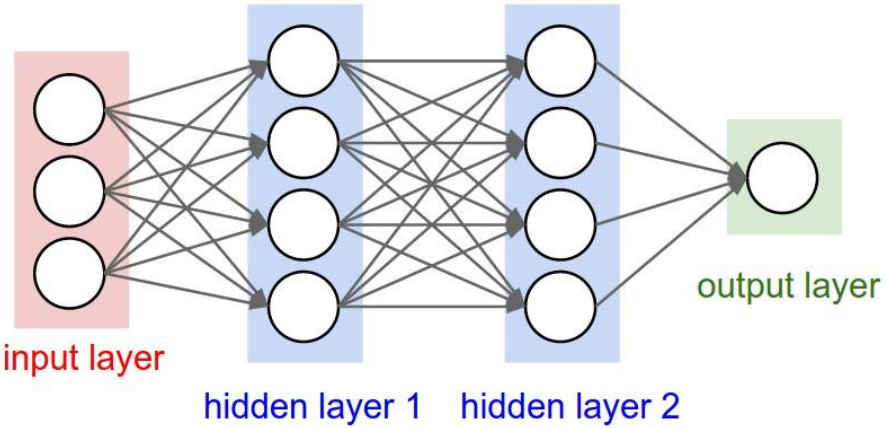
**artificial neural networks** are computational models inspired by animals' central nervous systems (in particular the brain) that are capable of machine learning and pattern recognition. They are usually presented as systems of interconnected "neurons" that can compute values from inputs by feeding information through the network.

In R

neuralnet() from neuralnet package http://cran.r-project.org/web/packa

In Python

We use pybrain http://pybrain.org/docs/quickstart/network.html



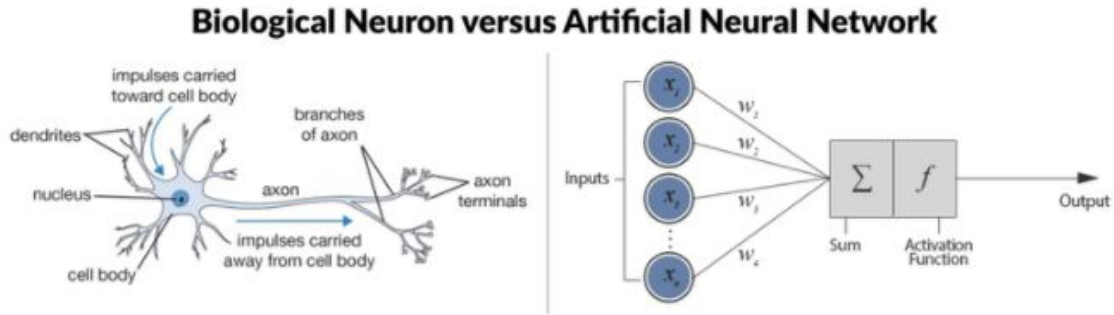
Deep Learning

# Deep Learning

deep learning, a subfield of machine learning that is a set of algorithms that is inspired by the structure and function of the brain. These algorithms are usually called Artificial Neural Networks (ANN). Deep learning is one of the hottest fields in data science with many case studies with marvelous results in robotics, image recognition and Artificial Intelligence (AI).

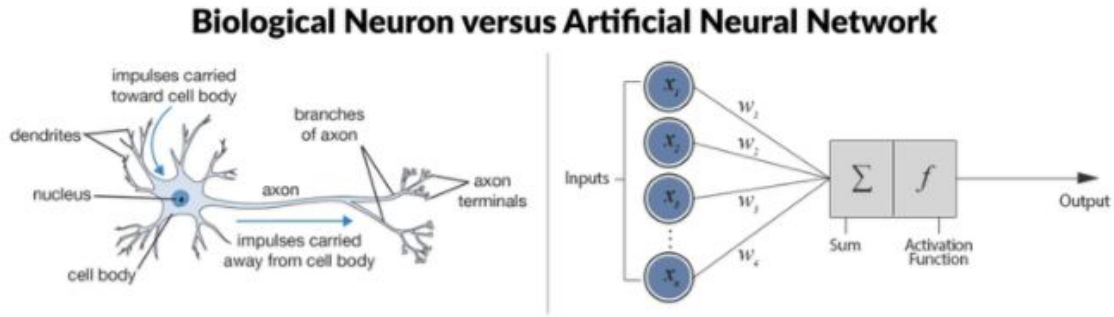
Deep Learning

deep learning, a subfield of machine learning that is a set of algorithms that is inspired by the structure and function of the brain. These algorithms are usually called Artificial Neural Networks (ANN). Deep learning is one of the hottest fields in data science with many case studies with marvelous results in robotics, image recognition and Artificial Intelligence (AI).



Deep Learning

https://www.datacamp.com/community/tutorials/deep-learning-python



# Tensor  Flow

## About TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

# Tensor Flow

```
devtools::install_github("rstudio/tfestimators")
install_tensorflow(version = "1.3.0")
install.packages("modelr")

library(tfestimators)
library(modelr)
response <- function() "Species"
features <- function() setdiff(names(iris), response())

# split into train, test datasets
set.seed(123)
partitions <- modelr::resample_partition(iris, c(test = 0.2, train = 0.8))
iris_train <- as.data.frame(partitions$train)
iris_test  <- as.data.frame(partitions$test)

# construct feature columns
feature_columns <- feature_columns(
  column_numeric(features())
)
```

# Tensor Flow

```
# construct classifier
classifier <- dnn_classifier(
  feature_columns = feature_columns,
  hidden_units = c(10, 20, 10),
  n_classes = 3
)

# construct input function
iris_input_fn <- function(data) {
  input_fn(data, features = features(), response = response())
}

# train classifier with training dataset
train(classifier, input_fn = iris_input_fn(iris_train))

# valuate with test dataset
predictions <- predict(classifier, input_fn = iris_input_fn(iris_test))
evaluation <- evaluate(classifier, input_fn = iris_input_fn(iris_test))
```

# Support Vector Machines

**support vector machines** (**SVMs**, also

**support vector networks**[1]) are supervised learning

models

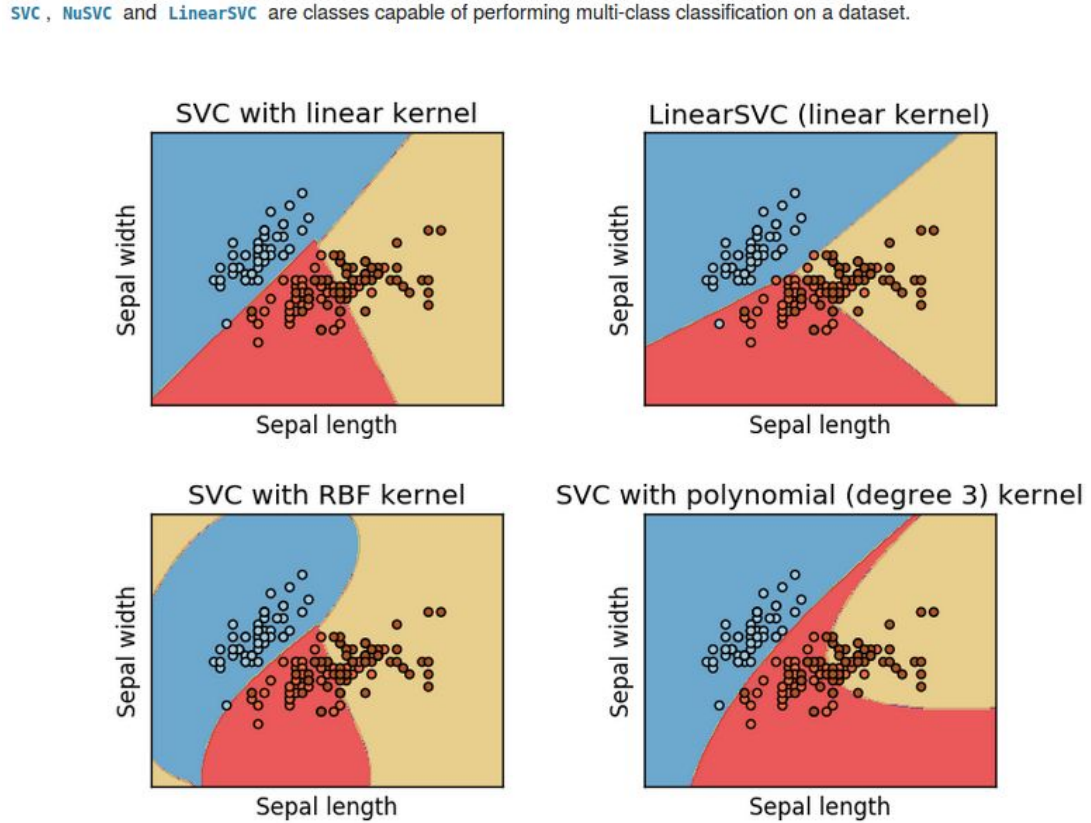
An SVM model is a representation of the examples as

points in space, mapped so that the examples of the

separate categories are divided by a clear gap

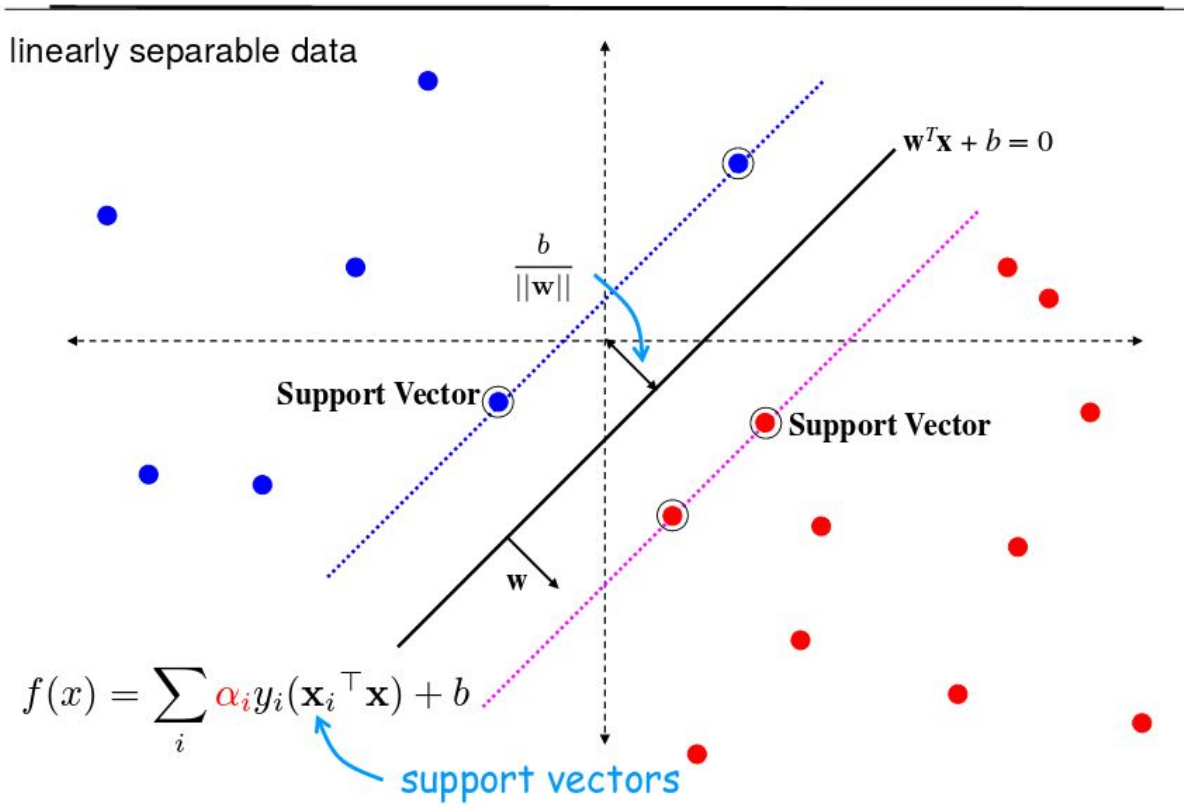
(hyperplane) that is

as wide as possible. New examples are then mapped



# Support Vector Machines

http://axon.cs.byu.edu/Dan/678/miscellaneous/SVM.example.pdf



# Decision Trees

**Decision tree learning** uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. More descriptive names for such tree models are **classification trees** or **regression trees**. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making

In R

C5.0 Algorithm using C5.0() function from the C50 package http://cran.r-project.org/web/packages/C50/C50.pdf

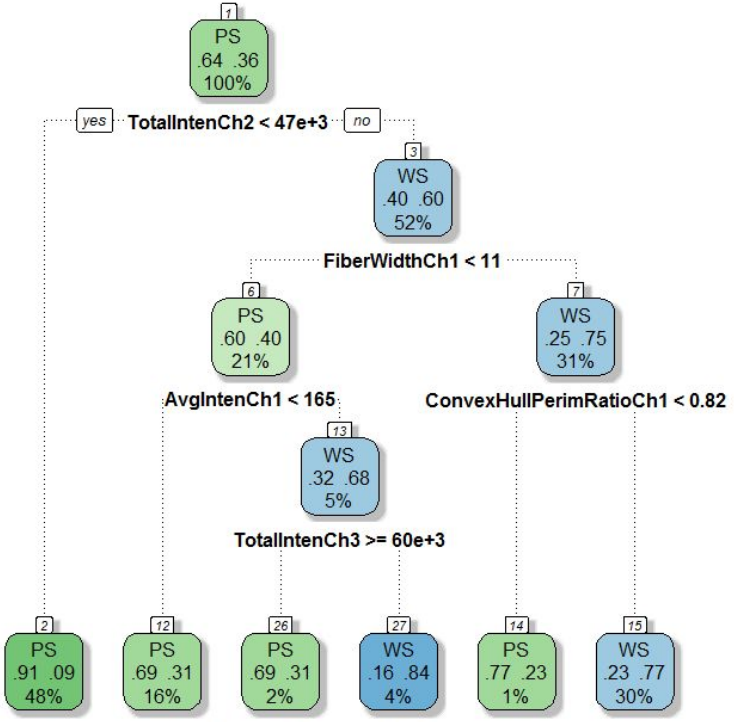
OneR function from RWeka Package http://cran.r-project.org/web/packages/RWeka/index.html

In Python http://scikit-learn.org/stable/modules/tree.html **DecisionTreeClassifier** is a class capable of performing multi-class

classification. .Decision trees can also be applied to regression problems, using the **DecisionTreeRegressor** class.

# Decision Trees ::rattle

http://blog.revolutionanalytics.com/2013/06/plotting-classification-and-regression-trees-with-plotrpart.html



# Decision Trees::rpart

http://www.statmethods.net/advstats/cart.html

### 1. Grow the Tree

To grow a tree, use

**rpart(***formula*, **data=**, **method=,control=)** where

| | |
|---|---|
| ***formula*** | is in the format<br>*outcome ~ predictor1+predictor2+predictor3*+ect. |
| **data=** | specifies the data frame |
| **method=** | **"class"** for a classification tree<br>**"anova"** for a regression tree |
| **control=** | optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001) requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted. |

# Decision Trees::rpart

```
11  library(rpart)
12  data(iris)
13  tree=rpart(Species~.,data=iris)
14  print(tree)
15  library(rattle)
16  fancyRpartPlot(tree)
```

9:13 (Top Level) R Script

Global Environment

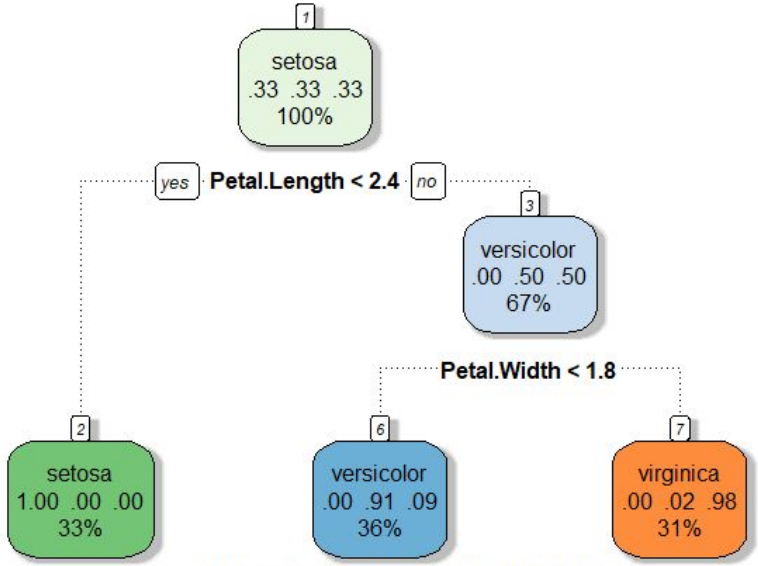
| | |
|---|---|
| partitions | List of 2 |
| sample.space | chr [1:2] "Heads" "Tails" |
| theta | 0.5 |
| tree | List of 14 |

Console ~/

```
Loading required package: sandwich

Attaching package: 'party'

The following object is masked _by_ '.GlobalEnv':

    response

> data(iris)
> library(rpart)
> tree=rpart(Species~.,data=iris)
> library(rattle)
Rattle: A free graphical interface for data science with R.
Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
Type 'rattle()' to shake, rattle, and roll your data.
> print(tree)
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
  2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
    6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259) *
    7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *
> fancyRpartPlot(tree)
```

Files Plots Packages Help Viewer

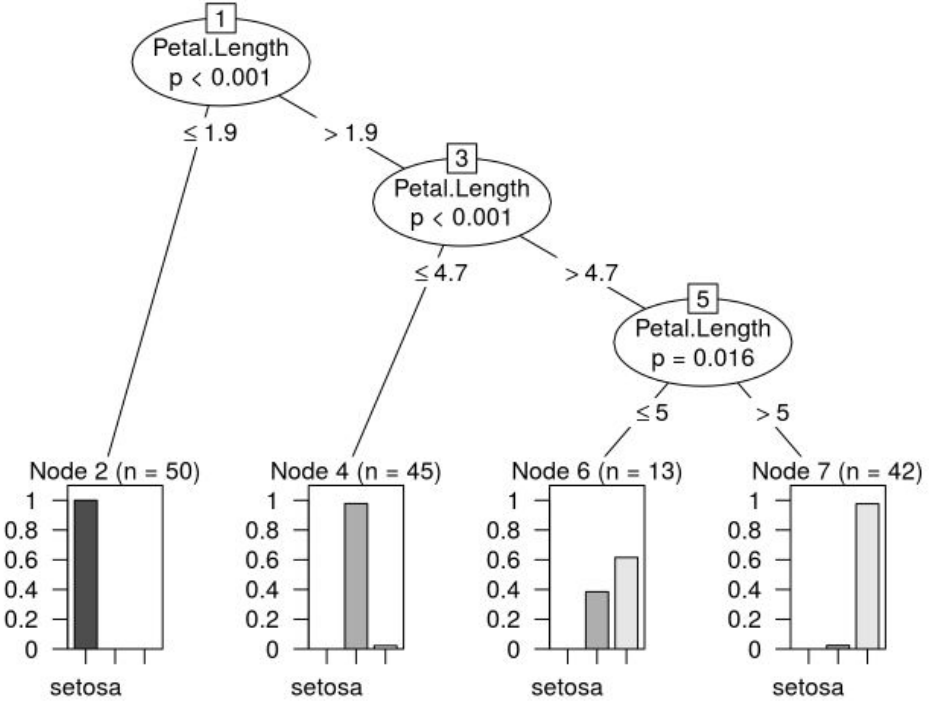
Zoom Export Publish



# Decision Trees :: ctree

```
library(party)
data("iris")
names(iris)
fit2 <- ctree(Species ~ Sepal.Length + Petal.Length + Sepal.Width ,
        data=iris)
plot(fit2)
print(fit2)
nodes(fit2,1)
nodes(fit2,3)
table(Predict(fit2), iris$Species)
```

```
fit2 <- ctree(Species ~ Sepal.Length + Petal.Length + Sepal.Width ,
              data=iris)
plot(fit2)
```



# Decision Trees :: party

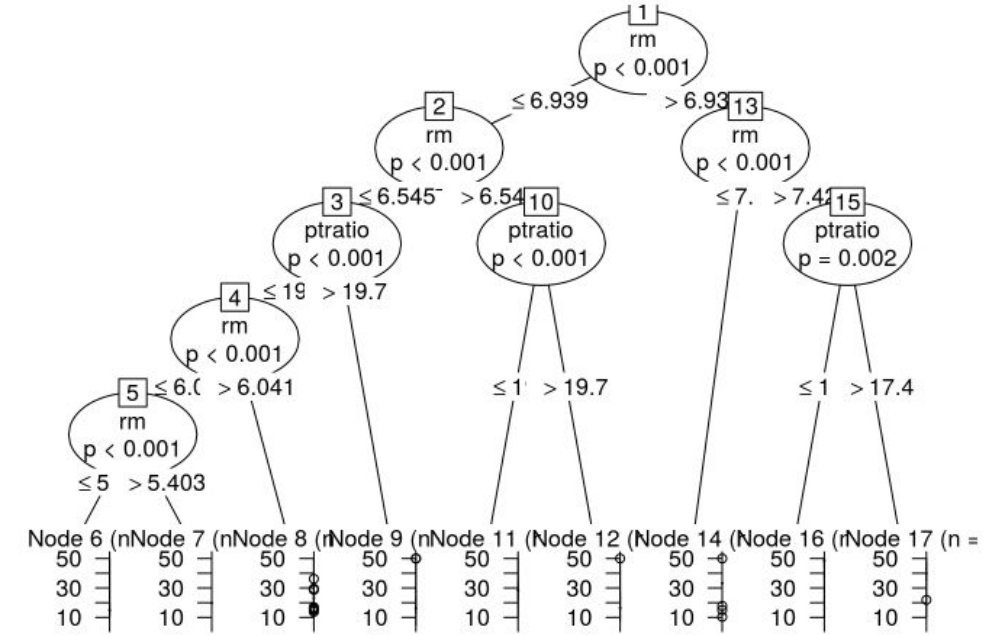
```
library(party)
data("iris")
names(iris)
fit2 <- ctree(Species ~ Sepal.Length + Petal.Length + Sepal.Width ,
        data=iris)
plot(fit2)
print(fit2)
nodes(fit2,1)
nodes(fit2,3)
table(Predict(fit2), iris$Species)
```

```
> print(fit2)

        Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Petal.Length, Sepal.Width
Number of observations:  150

1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
  2)*  weights = 50
1) Petal.Length > 1.9
  3) Petal.Length <= 4.7; criterion = 1, statistic = 61.228
    4)*  weights = 45
  3) Petal.Length > 4.7
    5) Petal.Length <= 5; criterion = 0.984, statistic = 7.701
      6)*  weights = 13
    5) Petal.Length > 5
      7)*  weights = 42
> table(Predict(fit2), iris$Species)

            setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         44         1
  virginica       0          6        49
```

# Decision Trees :: party

https://cran.r-project.org/web/packages/party/vignettes/party.pdf

```
fit <- ctree(medv ~ rm + ptratio ,
             data=Boston)
plot(fit)
```



# Decision Trees in Python

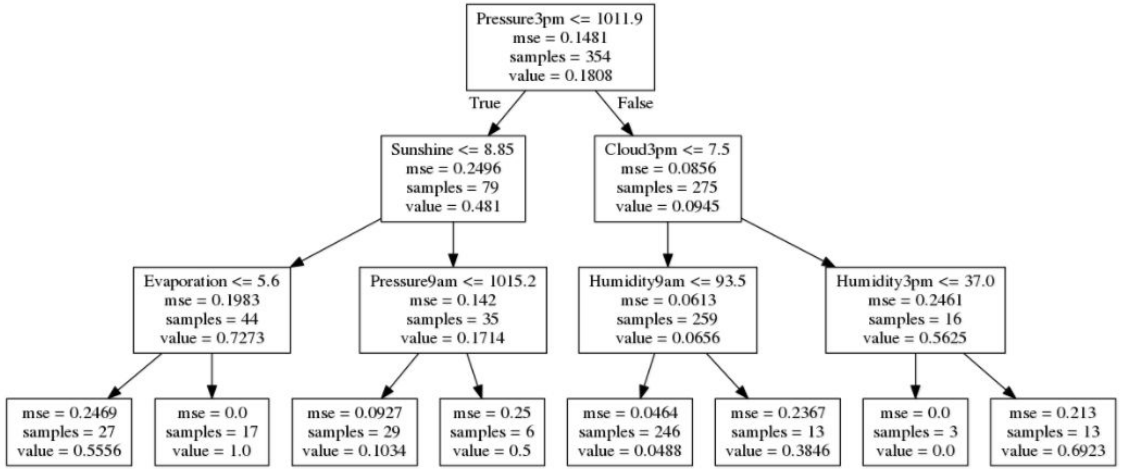
Weather Dataset

In [304]:
```
tree.export_graphviz(dtr, out_file=dot_data,feature_names=names[:-1])
```

In [305]:
```
graph = pydot.graph_from_dot_data(dot_data.getvalue())
```

In [306]:
```
Image(graph.create_png())
```

Out[306]:




In [307]:
```
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
```

In [308]:
```
model
```

# Random Forest

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

In the original paper on random forests, it was shown that the forest error rate depends on two things:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#intro

# Random Forest

```
library(randomForest)
fit3 <- randomForest(Species ~ Sepal.Length + Petal.Length + Sepal.Width ,
        data=iris)
print(fit3)
importance(fit3)
```

```
> library(randomForest)
> fit3 <- randomForest(Species ~ Sepal.Length + Petal.Length + Sepal.Width ,
+                 data=iris)
> print(fit3)

Call:
 randomForest(formula = Species ~ Sepal.Length + Petal.Length +      Sepal.Width, data = iris)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 6.67%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         50          0         0        0.00
versicolor      0         44         6        0.12
virginica       0          4        46        0.08
> importance(fit3)
             MeanDecreaseGini
Sepal.Length         26.77110
Petal.Length         53.82144
Sepal.Width          15.72355
>
```

# **Evaluating Results** Confusion Matrix

In the field of machine learning a **confusion matrix**, also known as an error matrix, [i]s a specific table layout that allows visualization of the performance of an algorithm,

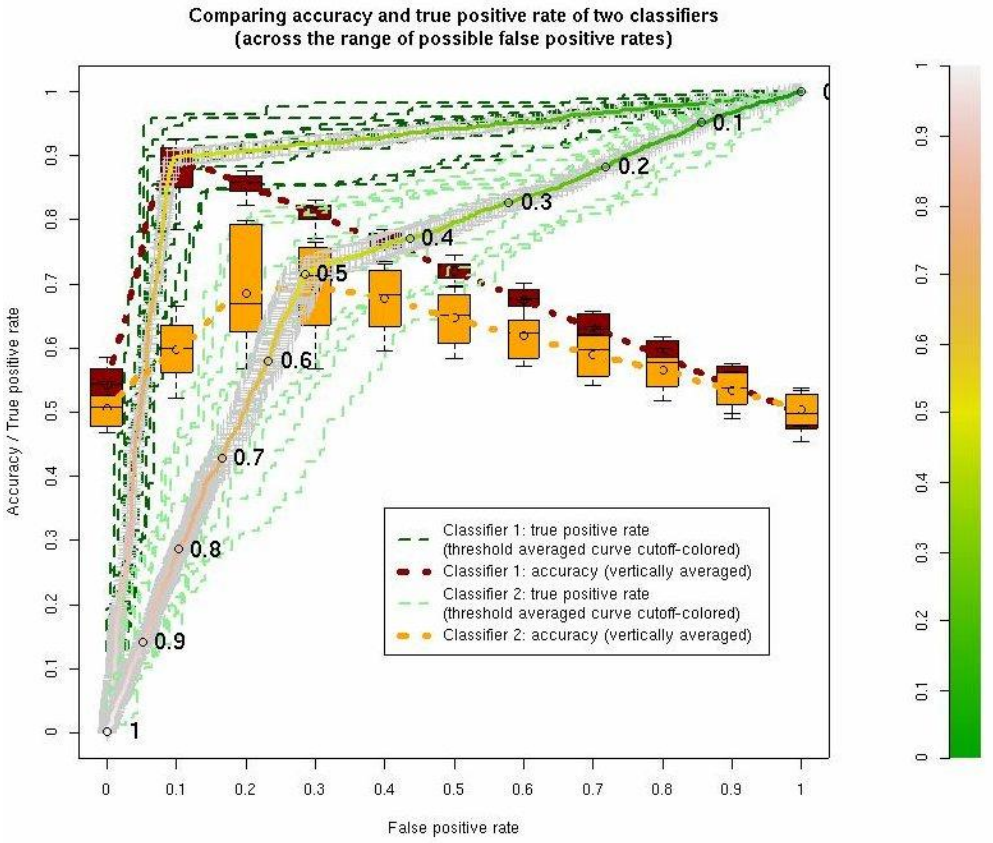
Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice versa)

| | | **Predicted condition** | | |
|---|---|---|---|---|
| | Total population | Predicted Condition positive | Predicted Condition negative | Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$ |
| **True condition** | condition positive | **True positive** | **False Negative** (Type II error) | True positive rate (TPR), Sensitivity, Recall = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$ | False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$ |
| | condition negative | **False Positive** (Type I error) | **True negative** | False positive rate (FPR), Fall-out = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$ | True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$ |
| | Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$ | Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$ | False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$ | Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$ | Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$ |
| | | False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$ | Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$ | Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$ | |

| | | **Predicted** | | |
|---|---|---|---|---|
| | | **Cat** | **Dog** | **Rabbit** |
| **Actual class** | **Cat** | 5 | 3 | 0 |
| | **Dog** | 2 | 3 | 1 |
| | **Rabbit** | 0 | 2 | 11 |

# **Evaluating Results** ROC Curve

a **receiver operating characteristic** (**ROC**), or **ROC curve**, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings



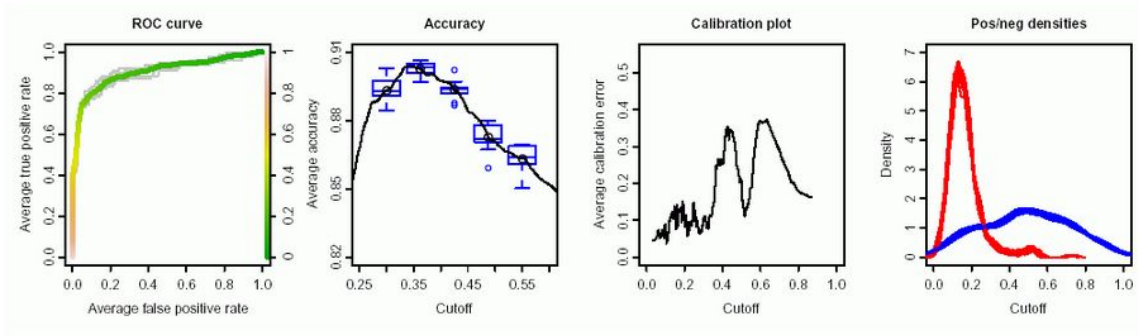
http://rocr.bioinf.mpi-sb.mpg.de/

Example

ROCR

prediction - performance - plot

[visualizing classifier performance in R, with only 3 commands]




**Performance measures that ROCR knows:**

*tive rate, false positive rate, true negative rate, false negative rate, sensitivity, ictive value, negative predictive value, precision, fallout, miss, phi correlation tion coefficient, mutual information, chi square statistic, odds ratio, lift value, OC convex hull, area under the ROC curve, precision/recall break-even point, s-entropy, root mean squared error, SAR measure, expected cost, explicit cost.*

**ROCR features:**

*ROC curves, precision/recall plots, lift charts, cost curves, custom curves by f measure for the x axis and one for the y axis, handling of data from cross-vali averaging (vertically, horizontally, or by threshold), standard error bars, box by cutoff, printing threshold values on the curve, tight integration with Rs plo adjust plots or to combine multiple plots), fully customizable, easy to use (onl*

http://farmacokratia.blogspot.in/2011/09/example-of-roc-curves-plotting-with.html

- Using ROCR's 3 commands to produce a simple ROC plot:
  pred <- prediction(predictions, labels)
  perf <- performance(pred, measure = "tpr", x.measure = "fpr")
  plot(perf, col=rainbow(10))
- Gallery

# Cross Validation

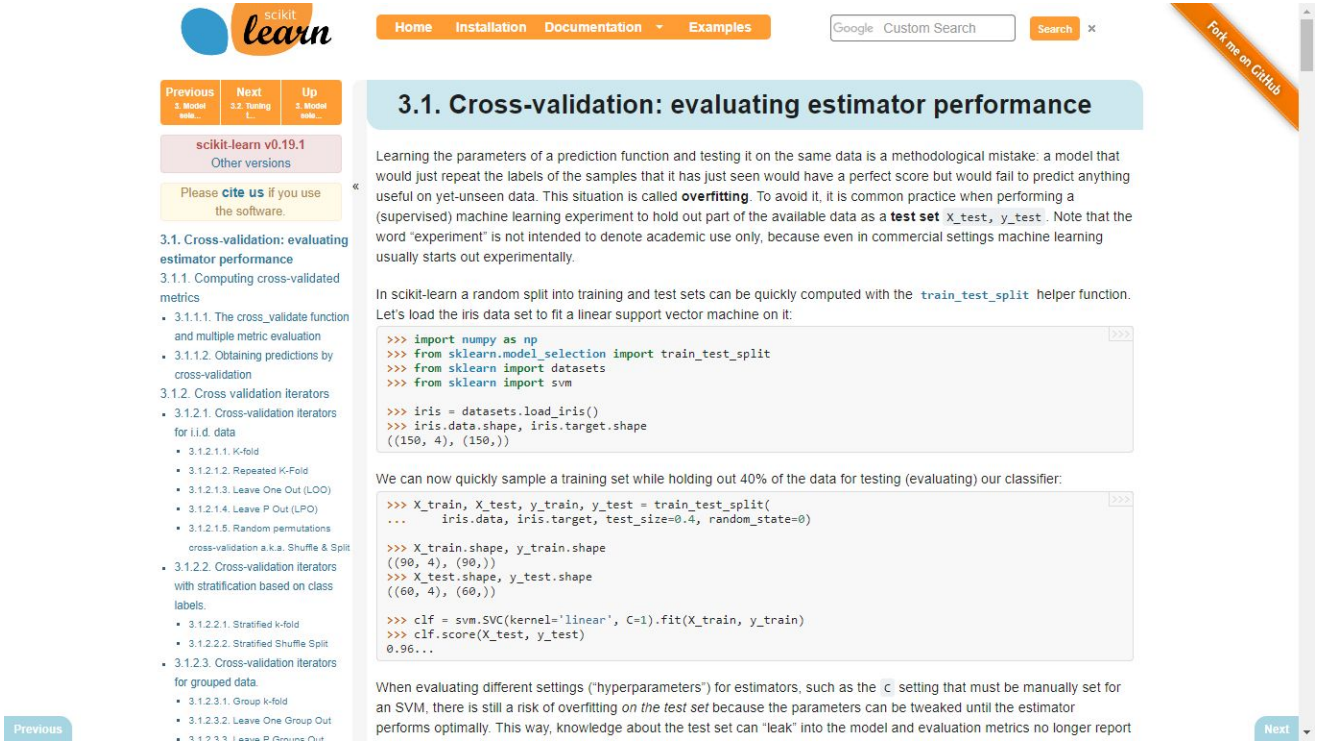
$k$

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called **overfitting**. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set** `X_test, y_test`

”:

- A model is trained using  of the folds as training data;
- the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

# Cross Validation



# Ensemble Methods

Ensemble learning involves combining multiple predictions derived by different techniques in order to create a stronger overall prediction. For example, the predictions of a random forest, a support vector machine, and a simple linear model may be combined to create a stronger final prediction set. The key to creating a powerful ensemble is model diversity.

An ensemble with two techniques that are very similar in nature will perform more poorly than a more diverse model set.

http://www.vikparuchuri.com/blog/intro-to-ensemble-learning-in-r/

# Bagging

Bagging, aka bootstrap aggregation, is a relatively simple way to increase the power of a predictive statistical model by taking multiple random samples(with replacement) from your training data set, and using each of these samples to construct a separate model and separate predictions for your test set. These predictions are then averaged to create a, hopefully more accurate, final prediction value.

http://www.vikparuchuri.com/blog/build-your-own-bagging-function-in-r/

# Boosting

**Boosting** is one of several classic methods for creating ensemble models, along with bagging, random forests, and so forth. *Boosting* means that each tree is dependent on prior trees, and learns by fitting the residual of the trees that preceded it. Thus, boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage.

XGBoost is a library designed and optimized for boosting trees algorithms. XGBoost is used in more than half of the winning solutions in machine learning challenges hosted at Kaggle.

http://xgboost.readthedocs.io/en/latest/model.html#

And http://dmlc.ml/rstats/2016/03/10/xgboost.html

# Unsupervised Learning

unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization* (Click here to go to the Scikit-Learn unsupervised learning page).

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

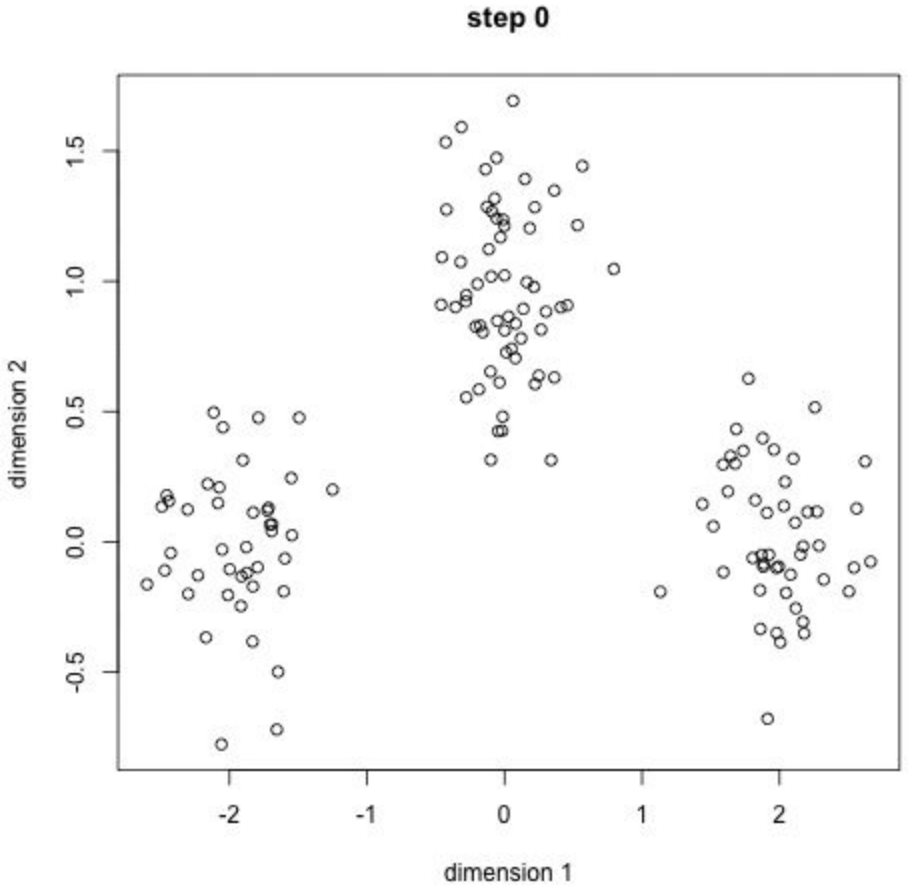
# Clustering

**Cluster analysis** or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (**clusters**).

**k-means clustering** aims to partition n observations into **k clusters** in which each observation belongs to the **cluster** with the nearest **mean**, serving as a prototype of the**cluster**. This results in a partitioning of the data space into Voronoi cells

http://shabal.in/visuals/kmeans/1.html

# Clustering



In [128]:
```
y.head()
```

Out[128]:

| | winetype |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |

In [ ]:
```
from sklearn.cluster import KMeans
import sklearn.metrics as sm
```

In [129]:
```
# K Means Cluster
model = KMeans(n_clusters=3)
model.fit(x)
```

Out[129]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [130]:
```
model.labels_
```

Out[130]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2, 1, 2, 2, 1, 2, 2, 1,
       1, 1, 2, 2, 0, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2,
       2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2,
       2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 2,
       2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2,
       1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2])
```

In [131]:
```
pd.value_counts(y['winetype'])
```

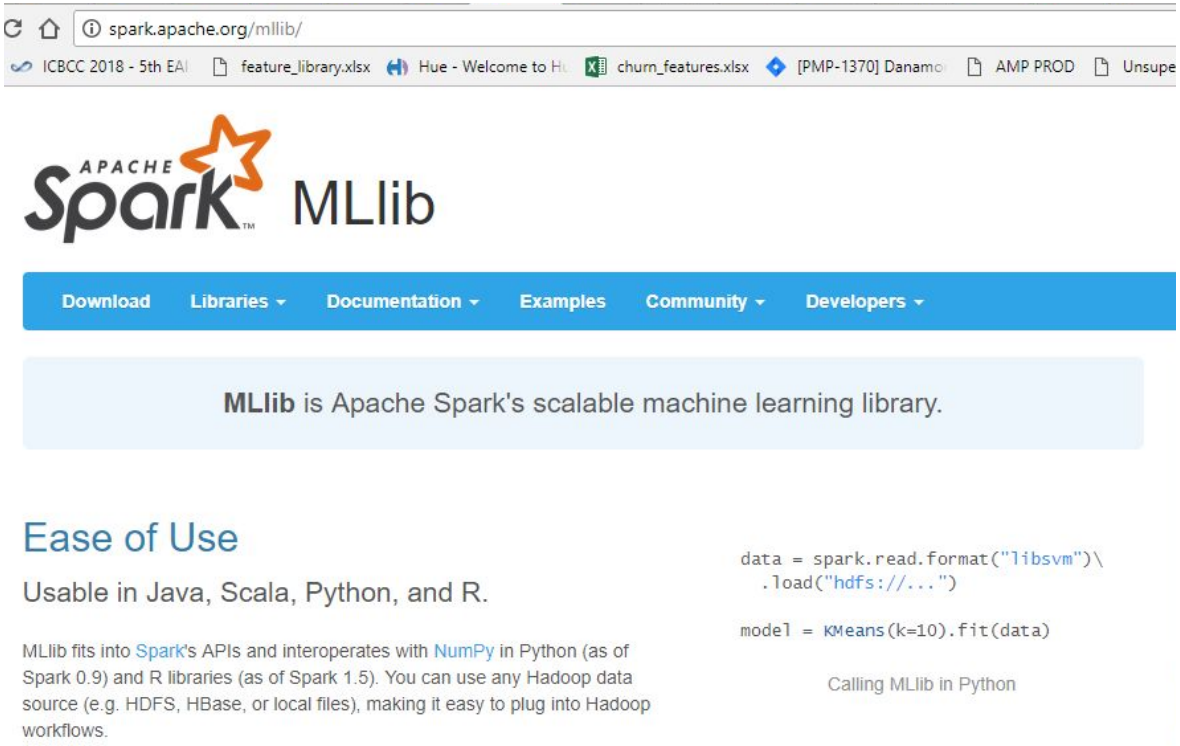
Out[131]:
```
2    71
1    59
3    48
```

# Clustering

**hierarchical clustering** (also called **hierarchical cluster analysis** or **HCA**) is a method of cluster analysis which seeks to build ahierarchy of clusters. Strategies for hierarchical clustering generally fall into two types: [1]

- **Agglomerative**: This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- **Divisive**: This is a "top down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

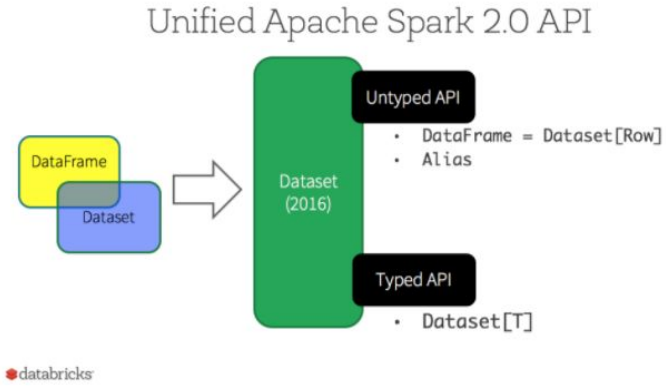
# MLLib -Machine Learning on Spark



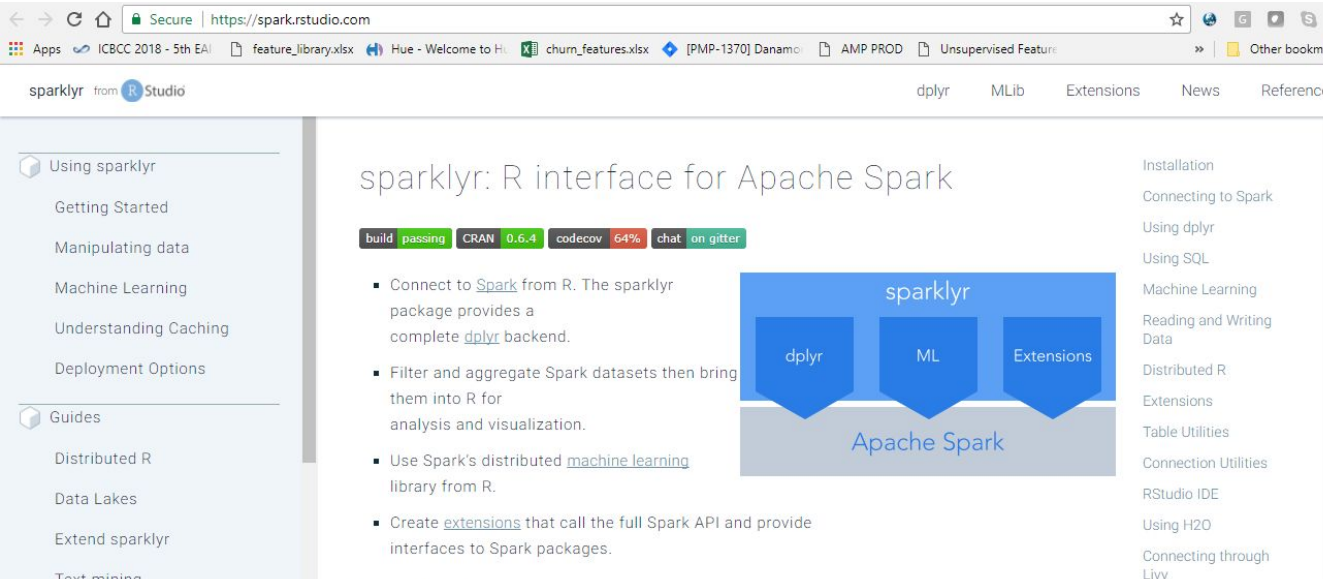
# 3 types of Data Objects in Spark

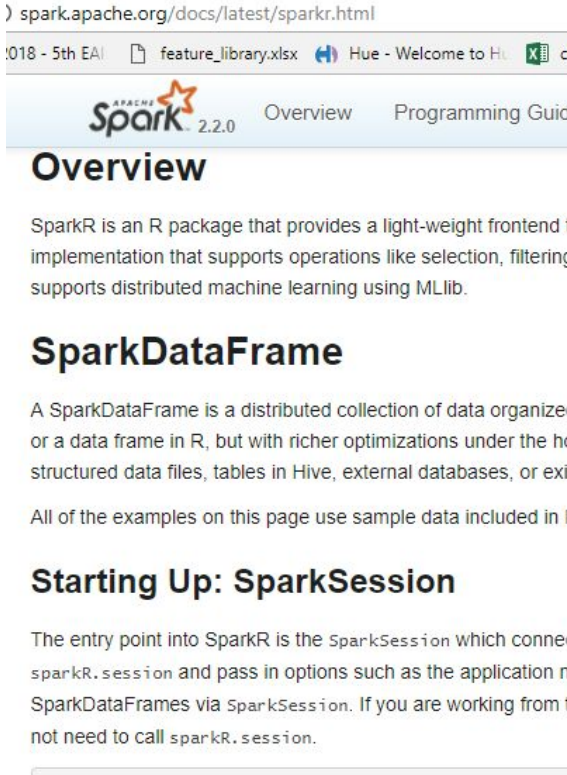
https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

RDD
Dataset
Dataframe

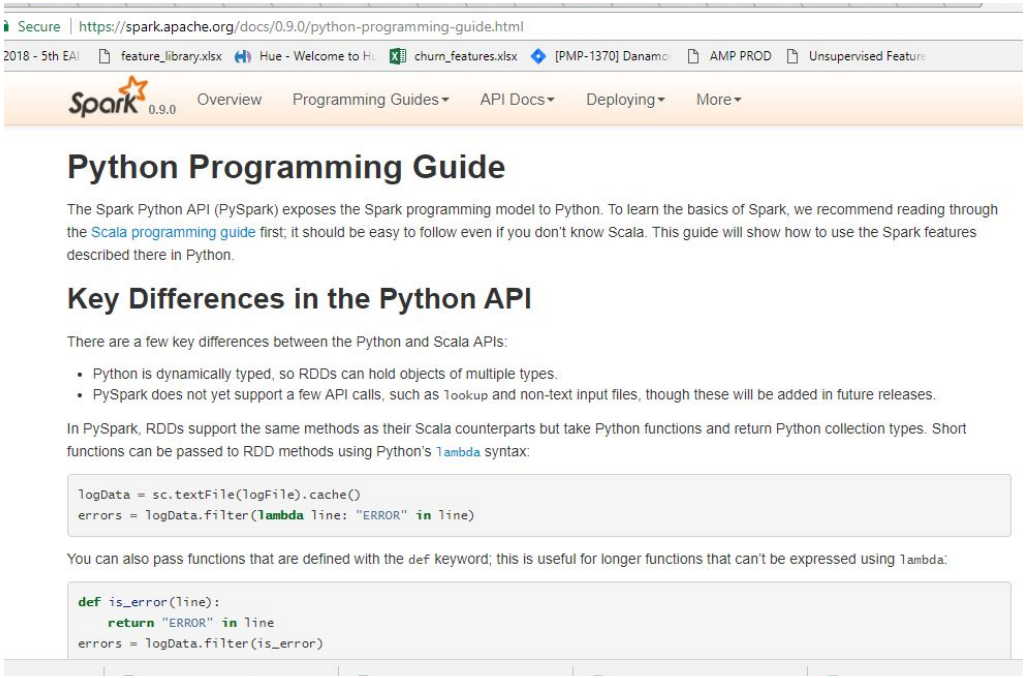


# Spark Machine Learning in R and Python



# Spark Machine Learning in R and Python



# Revision

Datasets- Case Studies
Visualization
Quiz 3-4-5-6

# Revision

What affects car price?

What kind of model would predict price?

Visualize cars data

# Revision

Take FBI data Table 1

Visualize it in ggplot (hint you can use a GUI)

Visualize is in seaboorn

What conclusion can you draw about crime ?
Write 3 conclusions

# Revision

Do Quiz 3

Do Quiz 4

Do Quiz 5

Do Quiz 6

# Revision

Take weather data and do a decision tree model for rain tommorow in R (hint use GUI)

Now make any model in Python for Weather

# Revision

Take German Credit Data . Make a regression model in R using steps shown