> 谷歌董事长施密特曾说过：虽然谷歌的无人驾驶汽车和机器人受到了许多媒体关注，但是这家公司真正的未来在于机器学习，一种让计算机更聪明、更个性化的技术。

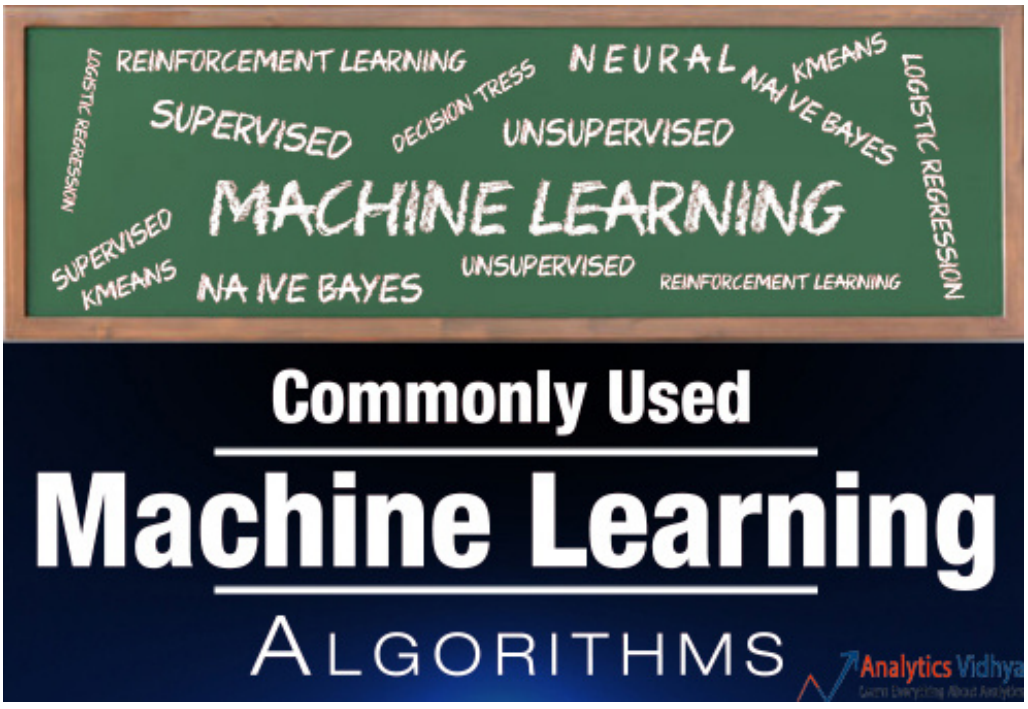
也许我们生活在人类历史上最关键的时期：从使用大型计算机，到个人电脑，再到现在的云计算。关键的不是过去发生了什么，而是将来会有什么发生。

工具和技术的民主化，让像我这样的人对这个时期兴奋不已。计算的蓬勃发展也是一样。如今，作为一名数据科学家，用复杂的算法建立数据处理机器一小时能赚到好几美金。但能做到这个程度可并不简单！我也曾有过无数黑暗的日日夜夜。

谁能从这篇指南里受益最多？

> 我今天所给出的，也许是我这辈子写下的最有价值的指南。

这篇指南的目的，是为那些有追求的数据科学家和机器学习狂热者们，简化学习旅途。这篇指南会让你动手解决机器学习的问题，并从实践中获得真知。我提供的是几个机器学习算法的高水平理解，以及运行这些算法的 R 和 Python 代码。这些应该足以让你亲自试一试了。



我特地跳过了这些技术背后的数据，因为一开始你并不需要理解这些。如果你想从数据层面上理解这些算法，你应该去别处找找。但如果你想要在开始一个机器学习项目之前做些准备，你会喜欢这篇文章的。

广义来说，有三种机器学习算法

1、 监督式学习

工作机制：这个算法由一个目标变量或结果变量（或因变量）组成。这些变量由已知的一系列预示变量（自变量）预测而来。利用这一系列变量，我们生成一个将输入值映射到期望输出值的函数。这个训练过程会一直持续，直到模型在训练数据上获得期望的精确度。监督式学习的例子有：回归、决策树、随机森林、K - 近邻算法、逻辑回归等。

2、非监督式学习

工作机制：在这个算法中，没有任何目标变量或结果变量要预测或估计。这个算法用在不同的组内聚类分析。这种分析方式被广泛地用来细分客户，根据干预的方式分为不同的用户组。非监督式学习的例子有：关联算法和 K - 均值算法。

3、强化学习

工作机制：这个算法训练机器进行决策。它是这样工作的：机器被放在一个能让它通过反复试错来训练自己的环境中。机器从过去的经验中进行学习，并且尝试利用了解最透彻的知识作出精确的商业判断。 强化学习的例子有马尔可夫决策过程。

常见机器学习算法名单

这里是一个常用的机器学习算法名单。这些算法几乎可以用在所有的数据问题上：

1. 线性回归
2. 逻辑回归
3. 决策树
4. SVM
5. 朴素贝叶斯
6. K最近邻算法
7. K均值算法
8. 随机森林算法
9. 降维算法
10. Gradient Boost 和 Adaboost 算法

1、线性回归

线性回归通常用于根据连续变量估计实际数值（房价、呼叫次数、总销售额等）。我们通过拟合最佳直线来建立自变量和因变量的关系。这条最佳直线叫做回归线，并且用 Y= a *X + b 这条线性等式来表示。

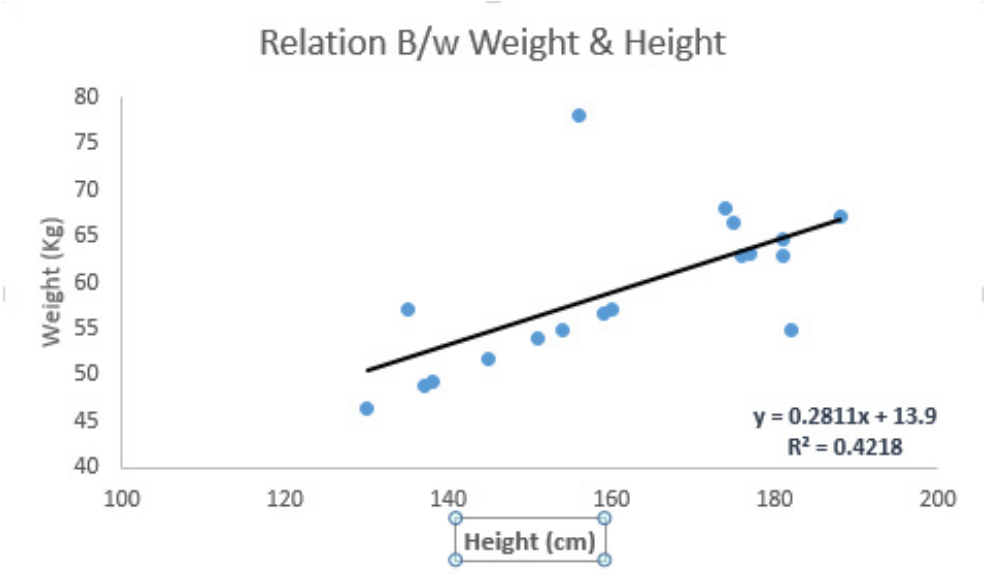
理解线性回归的最好办法是回顾一下童年。假设在不问对方体重的情况下，让一个五年级的孩子按体重从轻到重的顺序对班上的同学排序，你觉得这个孩子会怎么做？他（她）很可能会目测人们的身高和体型，综合这些可见的参数来排列他们。这是现实生活中使用线性回归的例子。实际上，这个孩子发现了身高和体型与体重有一定的关系，这个关系看起来很像上面的等式。

在这个等式中：

- Y：因变量
- a：斜率
- x：自变量
- b ：截距

系数 a 和 b 可以通过最小二乘法获得。

参见下例。我们找出最佳拟合直线 y=0.2811x+13.9。已知人的身高，我们可以通过这条等式求出体重。



线性回归的两种主要类型是一元线性回归和多元线性回归。一元线性回归的特点是只有一个自变量。多元线性回归的特点正如其名，存在多个自变量。找最佳拟合直线的时候，你可以拟合到多项或者曲线回归。这些就被叫做多项或曲线回归。

Python 代码

```python
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import linear_model

#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_datasets
x_test=input_variables_values_test_datasets

# Create linear regression object
linear = linear_model.LinearRegression()

# Train the model using the training sets and check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)

#Equation coefficient and Intercept
print('Coefficient: n', linear.coef_)
print('Intercept: n', linear.intercept_)

#Predict Output
predicted= linear.predict(x_test)
```

R代码

```r
#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train <- input_variables_values_training_datasets
y_train <- target_variables_values_training_datasets
x_test <- input_variables_values_test_datasets
x <- cbind(x_train,y_train)

# Train the model using the training sets and check score
linear <- lm(y_train ~ ., data = x)
summary(linear)

#Predict Output
predicted= predict(linear,x_test)
```

2、逻辑回归

别被它的名字迷惑了！这是一个分类算法而不是一个回归算法。该算法可根据已知的一系列因变

量估计离散数值（比方说二进制数值 0 或 1 ，是或否，真或假）。简单来说，它通过将数据拟合进一个逻辑函数来预估一个事件出现的概率。因此，它也被叫做逻辑回归。因为它预估的是概率，所以它的输出值大小在 0 和 1 之间（正如所预计的一样）。

让我们再次通过一个简单的例子来理解这个算法。

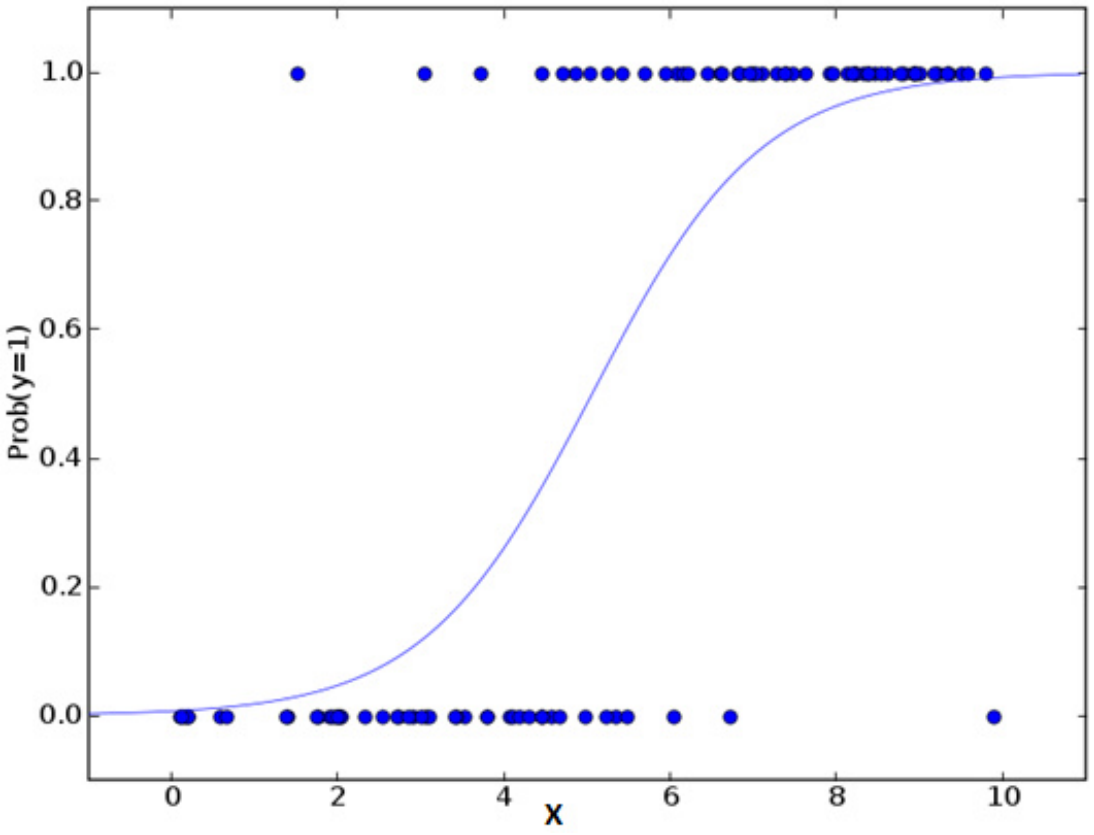
假设你的朋友让你解开一个谜题。这只会有两个结果：你解开了或是你没有解开。想象你要解答很多道题来找出你所擅长的主题。这个研究的结果就会像是这样：假设题目是一道十年级的三角函数题，你有 70%的可能会解开这道题。然而，若题目是个五年级的历史题，你只有30%的可能性回答正确。这就是逻辑回归能提供给你的信息。

从数学上看，在结果中，几率的对数使用的是预测变量的线性组合模型。

```
odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence
ln(odds) = ln(p/(1-p))
logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk
```

在上面的式子里，p 是我们感兴趣的特征出现的概率。它选用使观察样本值的可能性最大化的值作为参数，而不是通过计算误差平方和的最小值（就如一般的回归分析用到的一样）。

现在你也许要问了，为什么我们要求出对数呢？简而言之，这种方法是复制一个阶梯函数的最佳方法之一。我本可以更详细地讲述，但那就违背本篇指南的主旨了。



Python代码

```
#Import Library
from sklearn.linear_model import LogisticRegression
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of te
# Create logistic regression object
model = LogisticRegression()

# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)

#Equation coefficient and Intercept
print('Coefficient: n', model.coef_)
print('Intercept: n', model.intercept_)
```

```
#Predict Output
predicted= model.predict(x_test)
```

R代码

```
x <- cbind(x_train,y_train)
# Train the model using the training sets and check score
logistic <- glm(y_train ~ ., data = x,family='binomial')
summary(logistic)

#Predict Output
predicted= predict(logistic,x_test)
```

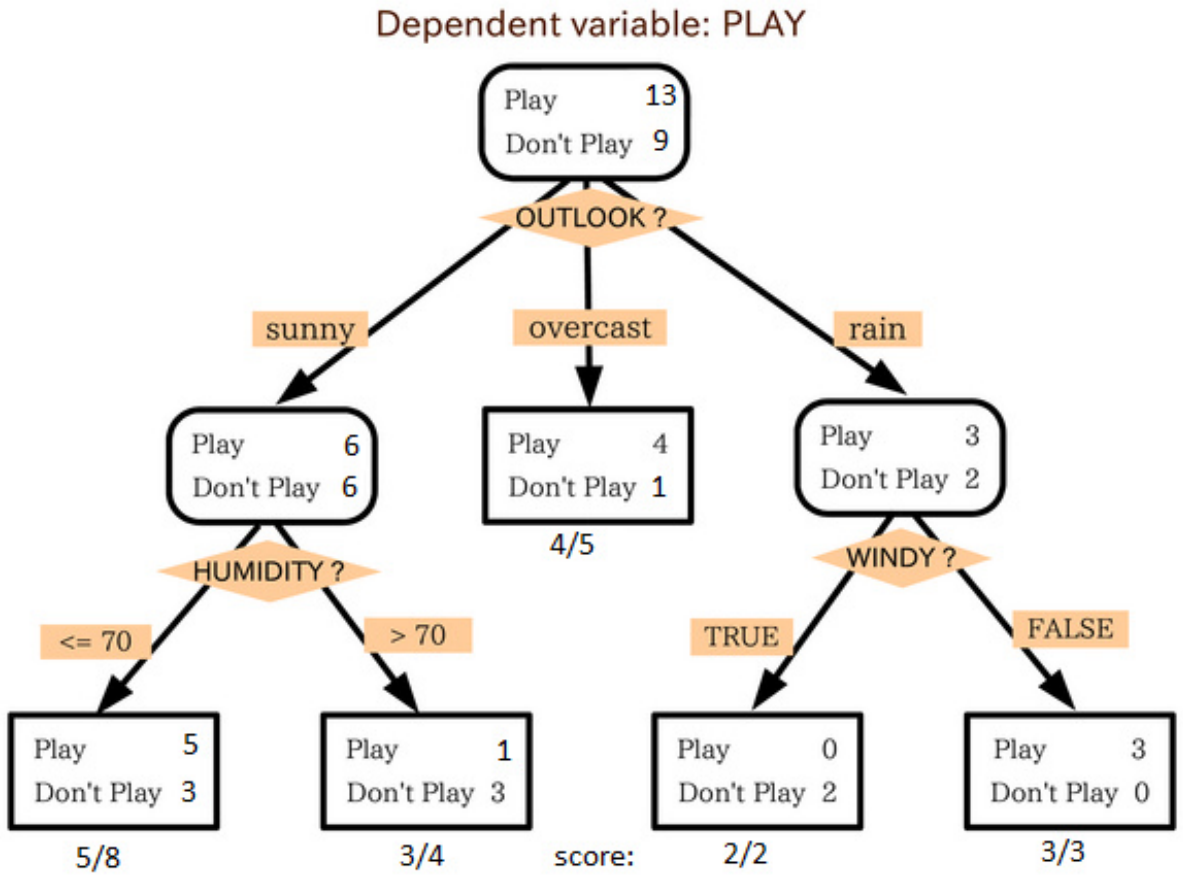
更进一步：

你可以尝试更多的方法来改进这个模型：

- 加入交互项
- 精简模型特性
- 使用正则化方法
- 使用非线性模型

3、决策树

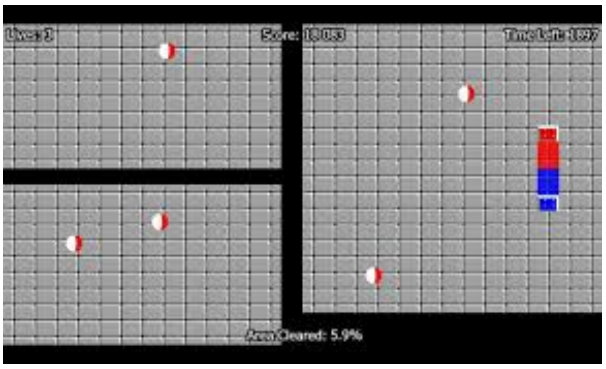
这是我最喜爱也是最频繁使用的算法之一。这个监督式学习算法通常被用于分类问题。令人惊奇的是，它同时适用于分类变量和连续因变量。在这个算法中，我们将总体分成两个或更多的同类群。这是根据最重要的属性或者自变量来分成尽可能不同的组别。想要知道更多，可以阅读：[简化决策树](#)。



来源： statsexchange

在上图中你可以看到，根据多种属性，人群被分成了不同的四个小组，来判断 “他们会不会去玩”。为了把总体分成不同组别，需要用到许多技术，比如说 Gini、Information Gain、Chi-square、entropy。

理解决策树工作机制的最好方式是玩Jezzball，一个微软的经典游戏（见下图）。这个游戏的最终目的，是在一个可以移动墙壁的房间里，通过造墙来分割出没有小球的、尽量大的空间。



因此，每一次你用墙壁来分隔房间时，都是在尝试着在同一间房里创建两个不同的总体。相似地，决策树也在把总体尽量分割到不同的组里去。

更多信息请见：[决策树算法的简化]

Python代码

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import tree

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of te
# Create tree object
model = tree.DecisionTreeClassifier(criterion='gini')
# for classification, here you can change the algorithm as gini or entropy (information gain) by

# model = tree.DecisionTreeRegressor() for regression
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)

#Predict Output
predicted= model.predict(x_test)
```

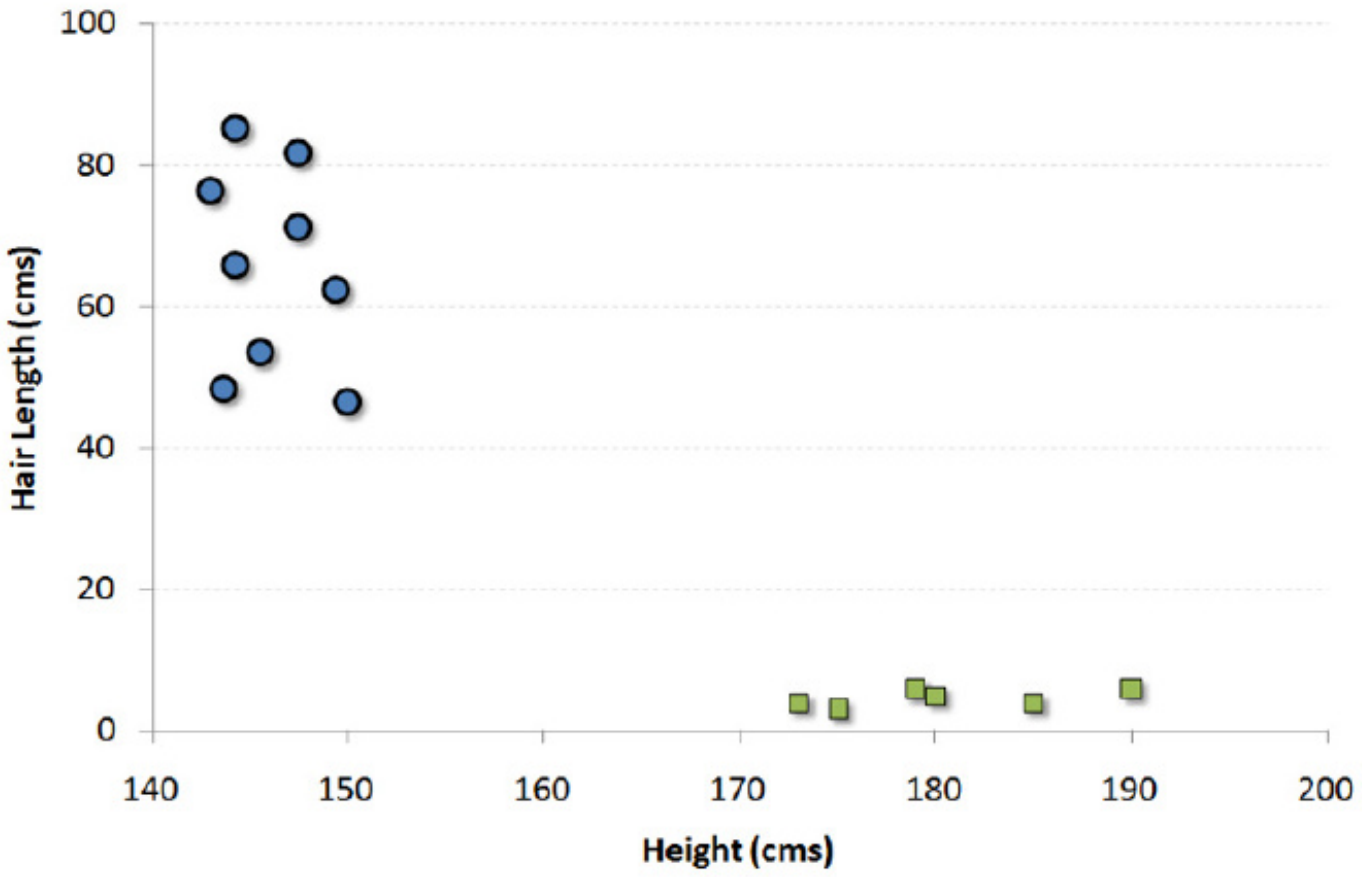
R代码

```
library(rpart)
x <- cbind(x_train,y_train)

# grow tree
fit <- rpart(y_train ~ ., data = x,method="class")
summary(fit)

#Predict Output
predicted= predict(fit,x_test)
```

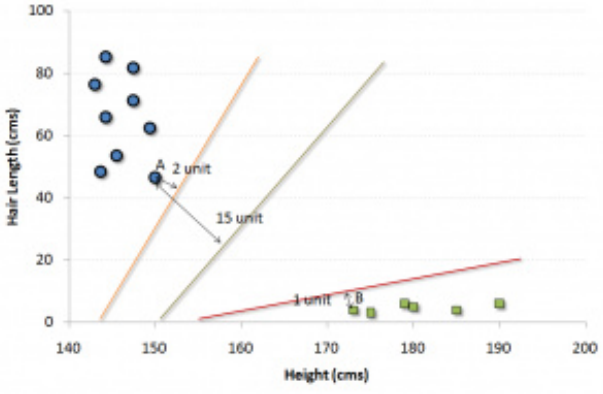
4、支持向量机

这是一种分类方法。在这个算法中，我们将每个数据在N维空间中用点标出（N是你所有的特征总数），每个特征的值是一个坐标的值。

举个例子，如果我们只有身高和头发长度两个特征，我们会在二维空间中标出这两个变量，每个点有两个坐标（这些坐标叫做支持向量）。



现在，我们会找到将两组不同数据分开的一条直线。两个分组中距离最近的两个点到这条线的距离同时最优化。



上面示例中的黑线将数据分类优化成两个小组，两组中距离最近的点（图中A、B点）到达黑线的距离满足最优条件。这条直线就是我们的分割线。接下来，测试数据落到直线的哪一边，我们就将它分到哪一类去。

更多请见：[支持向量机的简化]

将这个算法想作是在一个 N 维空间玩 JezzBall。需要对游戏做一些小变动：

- 比起之前只能在水平方向或者竖直方向画直线，现在你可以在任意角度画线或平面。
- 游戏的目的变成把不同颜色的球分割在不同的空间里。
- 球的位置不会改变。

Python代码

```
#Import Library
from sklearn import svm

#Assumed you have, X (predic
tor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create SVM classification object
model = svm.svc()
# there is various option associated with it, this is simple for classification. You can refer li
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)

#Predict Output
predicted= model.predict(x_test)
```

R代码

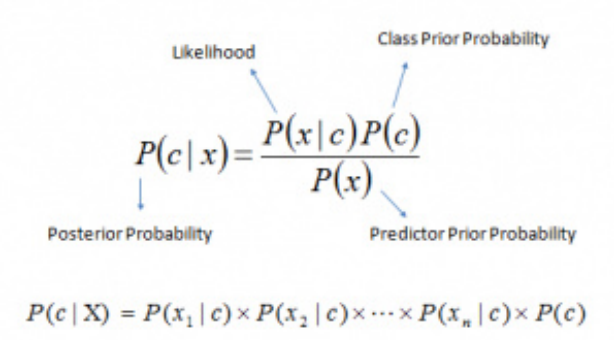
```
library(e1071)
x <- cbind(x_train,y_train)

# Fitting model
fit <-svm(y_train ~ ., data = x)
summary(fit)

#Predict Output
predicted= predict(fit,x_test)
```

5、朴素贝叶斯

在预示变量间相互独立的前提下，根据贝叶斯定理可以得到朴素贝叶斯这个分类方法。用更简单的话来说，一个朴素贝叶斯分类器假设一个分类的特性与该分类的其它特性不相关。举个例子，如果一个水果又圆又红，并且直径大约是 3 英寸，那么这个水果可能会是苹果。即便这些特性互相依赖，或者依赖于别的特性的存在，朴素贝叶斯分类器还是会假设这些特性分别独立地暗示这个水果是个苹果。

朴素贝叶斯模型易于建造，且对于大型数据集非常有用。虽然简单，但是朴素贝叶斯的表现却超越了非常复杂的分类方法。

贝叶斯定理提供了一种从P(c)、P(x)和P(x|c) 计算后验概率 P(c|x) 的方法。请看以下等式：



$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

在这里，

- P(c|x) 是已知预示变量（属性）的前提下，类（目标）的后验概率
- P(c) 是类的先验概率
- P(x|c) 是可能性，即已知类的前提下，预示变量的概率
- P(x) 是预示变量的先验概率

例子：让我们用一个例子来理解这个概念。在下面，我有一个天气的训练集和对应的目标变量“Play”。现在，我们需要根据天气情况，将会“玩”和“不玩”的参与者进行分类。让我们执行以下步骤。

步骤1：把数据集转换成频率表。

步骤2：利用类似“当Overcast可能性为0.29时，玩耍的可能性为0.64”这样的概率，创造 Likelihood 表格。

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

步骤3：现在，使用朴素贝叶斯等式来计算每一类的后验概率。后验概率最大的类就是预测的结果。

问题：如果天气晴朗，参与者就能玩耍。这个陈述正确吗？

我们可以使用讨论过的方法解决这个问题。于是 P（会玩 | 晴朗）= P（晴朗 | 会玩）* P（会玩）/ P （晴朗）

我们有 P （晴朗 |会玩）= 3/9 = 0.33，P（晴朗） = 5/14 = 0.36，P（会玩）= 9/14 = 0.64

现在，P(会玩 | 晴朗）= 0.33 * 0.64 / 0.36 = 0.60，有更大的概率。

朴素贝叶斯使用了一个相似的方法，通过不同属性来预测不同类别的概率。这个算法通常被用于文本分类，以及涉及到多个类的问题。

Python代码

```python
#Import Library
from sklearn.naive_bayes import GaussianNB

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of te
# Create SVM classification object model = GaussianNB() # there is other distribution for multinc
# Train the model using the training sets and check score
model.fit(X, y)

#Predict Output
predicted= model.predict(x_test)
```

R代码

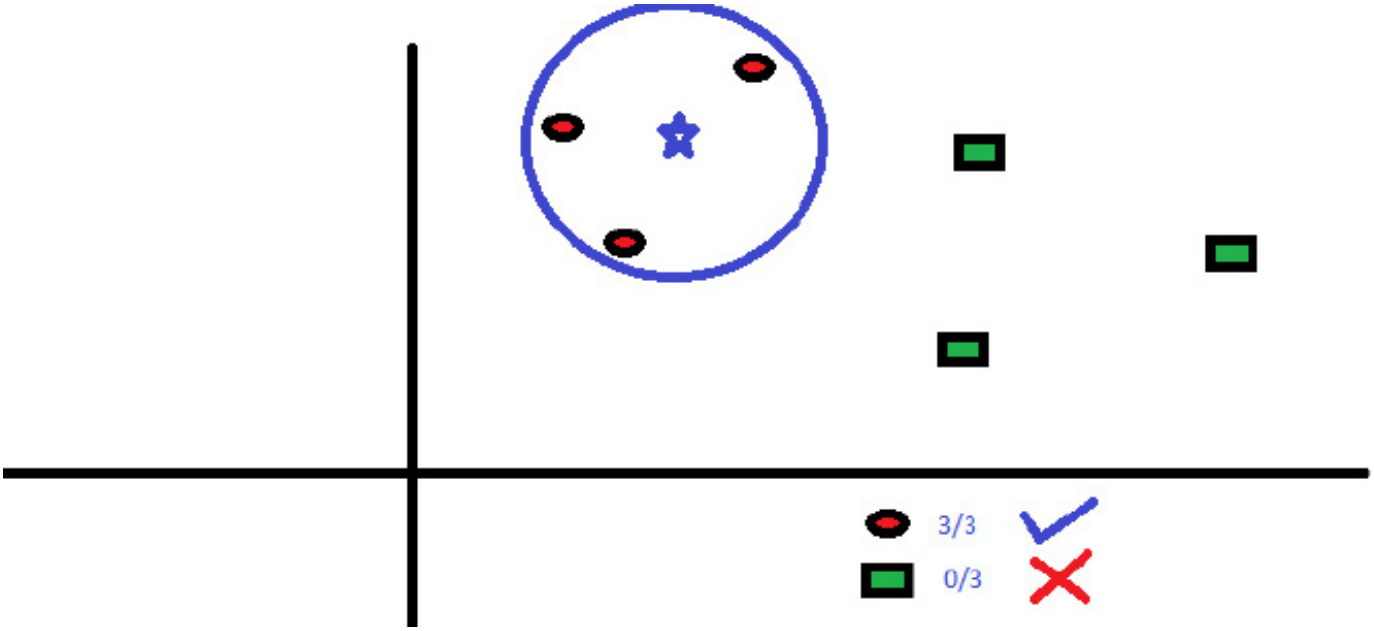
```r
library(e1071)
x <- cbind(x_train,y_train)

# Fitting model
fit <-naiveBayes(y_train ~ ., data = x)
summary(fit)

#Predict Output
predicted= predict(fit,x_test)
```

6、KNN（K – 最近邻算法）

该算法可用于分类问题和回归问题。然而，在业界内，K – 最近邻算法更常用于分类问题。K – 最近邻算法是一个简单的算法。它储存所有的案例，通过周围k个案例中的大多数情况划分新的案例。根据一个距离函数，新案例会被分配到它的 K 个近邻中最普遍的类别中去。

这些距离函数可以是欧式距离、曼哈顿距离、明式距离或者是汉明距离。前三个距离函数用于连续函数，第四个函数（汉明函数）则被用于分类变量。如果 K=1，新案例就直接被分到离其最近的案例所属的类别中。有时候，使用 KNN 建模时，选择 K 的取值是一个挑战。

更多信息：K – 最近邻算法入门（简化版）



我们可以很容易地在现实生活中应用到 KNN。如果想要了解一个完全陌生的人，你也许想要去找他的好朋友们或者他的圈子来获得他的信息。

在选择使用 KNN 之前，你需要考虑的事情：

- KNN 的计算成本很高。
- 变量应该先标准化（normalized），不然会被更高范围的变量偏倚。
- 在使用KNN之前，要在野值去除和噪音去除等前期处理多花功夫。

Python代码

```
#Import Library
from sklearn.neighbors import KNeighborsClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of te
# Create KNeighbors classifier object model
KNeighborsClassifier(n_neighbors=6)
# default value for n_neighbors is 5

# Train the model using the training sets and check score
model.fit(X, y)

#Predict Output
predicted= model.predict(x_test)
```

R代码

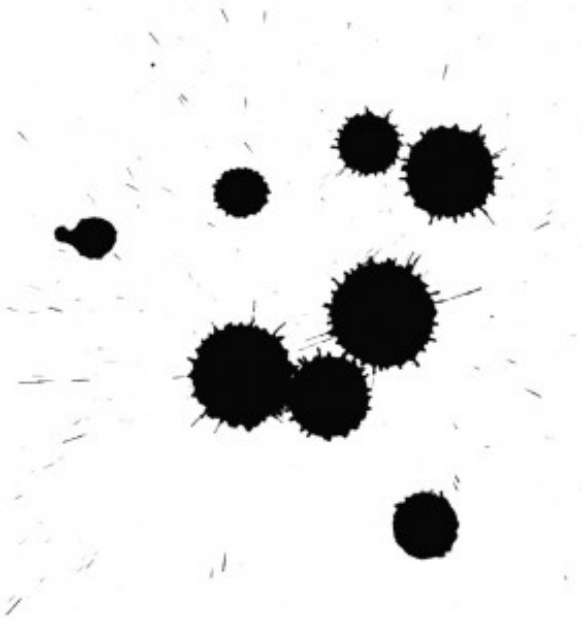
```
library(knn)
x <- cbind(x_train,y_train)

# Fitting model
fit <-knn(y_train ~ ., data = x,k=5)
summary(fit)
```

```
7
8  #Predict Output
9  predicted= predict(fit,x_test)
```

7、K 均值算法

K - 均值算法是一种非监督式学习算法，它能解决聚类问题。使用 K - 均值算法来将一个数据归入一定数量的集群（假设有 k 个集群）的过程是简单的。一个集群内的数据点是均匀齐次的，并且异于别的集群。

还记得从墨水渍里找出形状的活动吗？K - 均值算法在某方面类似于这个活动。观察形状，并延伸想象来找出到底有多少种集群或者总体。



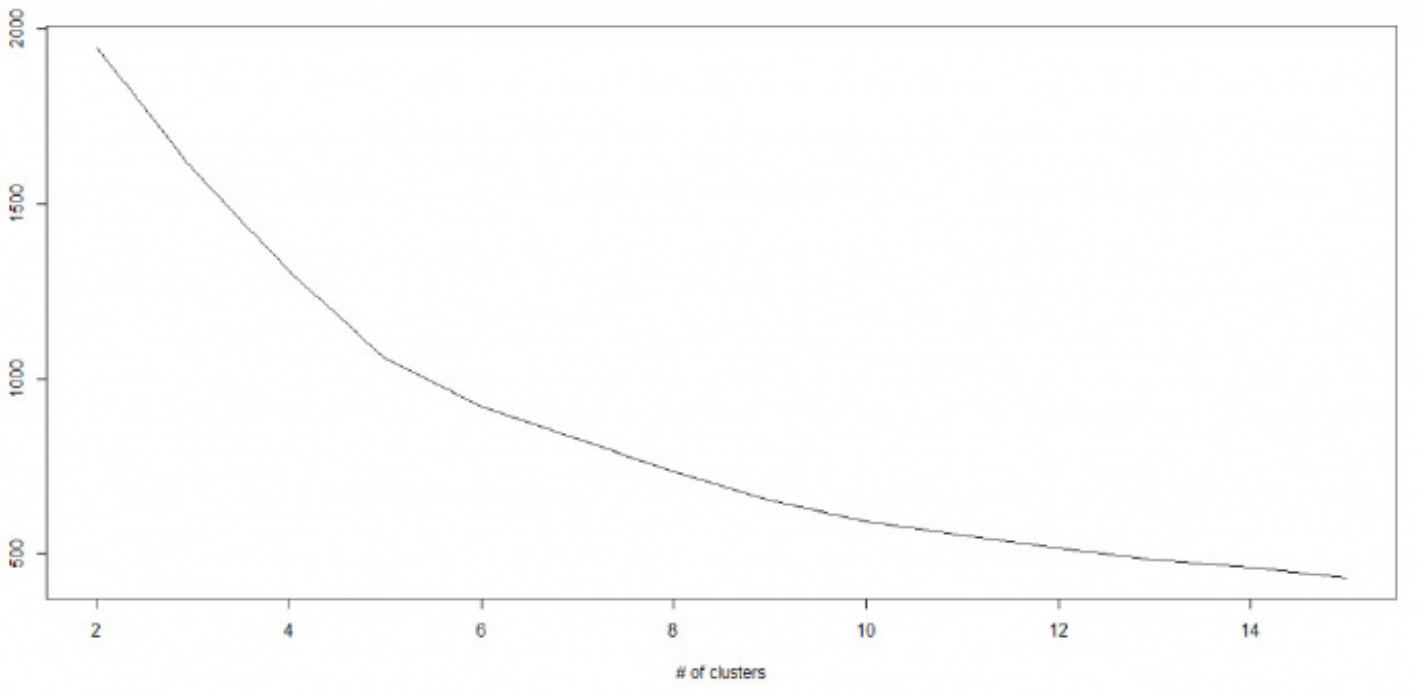
K - 均值算法怎样形成集群：

1. K - 均值算法给每个集群选择k个点。这些点称作为质心。
2. 每一个数据点与距离最近的质心形成一个集群，也就是 k 个集群。
3. 根据现有的类别成员，找出每个类别的质心。现在我们有了新质心。
4. 当我们有新质心后，重复步骤 2 和步骤 3。找到距离每个数据点最近的质心，并与新的k集群联系起来。重复这个过程，直到数据都收敛了，也就是当质心不再改变。

如何决定 K 值：

K - 均值算法涉及到集群，每个集群有自己的质心。一个集群内的质心和各数据点之间距离的平方和形成了这个集群的平方值之和。同时，当所有集群的平方值之和加起来的时候，就组成了集群方案的平方值之和。

我们知道，当集群的数量增加时，K值会持续下降。但是，如果你将结果用图表来表示，你会看到距离的平方总和快速减少。到某个值 k 之后，减少的速度就大大下降了。在此，我们可以找到集群数量的最优值。



Python代码

```python
#Import Library
from sklearn.cluster import KMeans

#Assumed you have, X (attributes) for training data set and x_test(attributes) of test_dataset
# Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)

# Train the model using the training sets and check score
model.fit(X)

#Predict Output
predicted= model.predict(x_test)
```

R代码

```r
library(cluster)
fit <- kmeans(X, 3) # 5 cluster solution
```

8、随机森林

随机森林是表示决策树总体的一个专有名词。在随机森林算法中，我们有一系列的决策树（因此又名“森林”）。为了根据一个新对象的属性将其分类，每一个决策树有一个分类，称之为这个决策树“投票”给该分类。这个森林选择获得森林里（在所有树中）获得票数最多的分类。

每棵树是像这样种植养成的：

1. 如果训练集的案例数是 N，则从 N 个案例中用重置抽样法随机抽取样本。这个样本将作为“养育”树的训练集。
2. 假如有 M 个输入变量，则定义一个数字 m<<M。m 表示，从 M 中随机选中 m 个变量，这 m 个变量中最好的切分会被用来切分该节点。在种植森林的过程中，m 的值保持不变。
3. 尽可能大地种植每一棵树，全程不剪枝。

若想了解这个算法的更多细节，比较决策树以及优化模型参数，我建议你阅读以下文章：

1. 随机森林入门—简化版
2. 将 CART 模型与随机森林比较（上）
3. 将随机森林与 CART 模型比较（下）

4. 调整你的随机森林模型参数

Python

```
#Import Library
from sklearn.ensemble import RandomForestClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of te
# Create Random Forest object
model= RandomForestClassifier()

# Train the model using the training sets and check score
model.fit(X, y)

#Predict Output
predicted= model.predict(x_test)
```

R代码

```
library(randomForest)
x <- cbind(x_train,y_train)

# Fitting model
fit <- randomForest(Species ~ ., x,ntree=500)
summary(fit)

#Predict Output
predicted= predict(fit,x_test)
```

9、降维算法

在过去的 4 到 5 年里，在每一个可能的阶段，信息捕捉都呈指数增长。公司、政府机构、研究组织在应对着新资源以外，还捕捉详尽的信息。

举个例子：电子商务公司更详细地捕捉关于顾客的资料：个人信息、网络浏览记录、他们的喜恶、购买记录、反馈以及别的许多信息，比你身边的杂货店售货员更加关注你。

作为一个数据科学家，我们提供的数据包含许多特点。这听起来给建立一个经得起考研的模型提供了很好材料，但有一个挑战：如何从 1000 或者 2000 里分辨出最重要的变量呢？在这种情况下，降维算法和别的一些算法（比如决策树、随机森林、PCA、因子分析）帮助我们根据相关矩阵，缺失的值的比例和别的要素来找出这些重要变量。

想要知道更多关于该算法的信息，可以阅读《降维算法的初学者指南》。

Python代码

```
#Import Library
from sklearn import decomposition

#Assumed you have training and test data set as train and test
# Create PCA obeject pca= decomposition.PCA(n_components=k) #default value of k =min(n_sample, n_
# For Factor analysis
#fa= decomposition.FactorAnalysis()
# Reduced the dimension of training dataset using PCA
train_reduced = pca.fit_transform(train)

#Reduced the dimension of test dataset
test_reduced = pca.transform(test)

#For more detail on this, please refer  this link.
```

R Code

```
library(stats)
```

```
pca <- princomp(train, cor = TRUE)
train_reduced  <- predict(pca,train)
test_reduced  <- predict(pca,test)
```

10、Gradient Boosting 和 AdaBoost 算法

当我们要处理很多数据来做一个有高预测能力的预测时，我们会用到 GBM 和 AdaBoost 这两种 boosting 算法。boosting 算法是一种集成学习算法。它结合了建立在多个基础估计值基础上的预测结果，来增进单个估计值的可靠程度。这些 boosting 算法通常在数据科学比赛如 Kaggl、AV Hackathon、CrowdAnalytix 中很有效。

更多：详尽了解 Gradient 和 AdaBoost

Python代码

```
#Import Library
from sklearn.ensemble import GradientBoostingClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of te
# Create Gradient Boosting Classifier object
model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=

# Train the model using the training sets and check score
model.fit(X, y)

#Predict Output
predicted= model.predict(x_test)
```

R代码

```
library(caret)
x <- cbind(x_train,y_train)

# Fitting model
fitControl <- trainControl( method = "repeatedcv", number = 4, repeats = 4)
fit <- train(y ~ ., data = x, method = "gbm", trControl = fitControl,verbose = FALSE)
predicted= predict(fit,x_test,type= "prob")[,2]
```

GradientBoostingClassifier 和随机森林是两种不同的 boosting 树分类器。人们常常问起这两个算法之间的区别。

结语

现在我能确定，你对常用的机器学习算法应该有了大致的了解。写这篇文章并提供 Python 和 R 语言代码的唯一目的，就是让你立马开始学习。如果你想要掌握机器学习，那就立刻开始吧。做做练习，理性地认识整个过程，应用这些代码，并感受乐趣吧！