本文资源翻译@酒酒 Angie：伊利诺伊大学香槟分校统计学同学，大四在读，即将开始计算机的研究生学习。希望认识更多喜欢大数据和机器学习的朋友，互相交流学习。

内容校正调整：寒小阳 && 龙心尘
时间：2016 年 4 月
出处：http://blog.csdn.net/han_xiaoyang/article/details/51191386
http://blog.csdn.net/longxinchen_ml/article/details/51192086

_

> 谷歌的无人车和机器人得到了很多关注，但我们真正的未来却在于能够使电脑变得更聪明，更人性化的技术，机器学习。
>
> – 埃里克 施密特（谷歌首席执行官）

当计算从大型计算机转移至个人电脑再转移到云的今天，我们可能正处于人类历史上最关键的时期。之所以关键，并不是因为已经取得的成就，而是未来几年里我们即将要获得的进步和成就。

对我来说，如今最令我激动的就是计算技术和工具的普及，从而带来了计算的春天。作为一名数据科学家，我可以建造一个数据处理系统来进行复杂的算法运算，这样每小时能赚几美金。可是学习这些算法却花了我无数个日日夜夜。

那么谁能从这篇文章里收益最多呢？

这篇文章有可能是我写的所有文章里最有价值的一篇。

写这篇文章的目的，就是希望它可以让有志于从事数据科学和机器学习的诸位在学习算法的路上少走些路。我会在文章中举例一些机器学习的问题，你们也可以在思考解决这些问题的过程中得到启发。我也会写下对于各种机器学习算法的一些个人理解，并且提供 R 和 Python 的执行代码。读完这篇文章，读者们至少可以行动起来亲手试试写一个机器学习的程序。

不过，这篇文章并没有阐述这些算法背后的统计学原理，有时候从实践入手也是很好的学习路径。如果你希望了解的是这些统计学原理，那么这篇文章的内容可能并不适合你。

一般说来，机器学习有三种算法：

1. 监督式学习

监督式学习算法包括一个目标变量(因变量)和用来预测目标变量的预测变量(自变量)。通过这些变量我们可以搭建一个模型，从而对于一个已知的预测变量值，我们可以得到对应的目标变量值。重复训练这个模型，直到它能在训练数据集上达到预定的准确度。

属于监督式学习的算法有：回归模型，决策树，随机森林，K 邻近算法，逻辑回归等。

2. 无监督式学习

与监督式学习不同的是，无监督学习中我们没有需要预测或估计的目标变量。无监督式学习是用来对总体对象进行分类的。它在根据某一指标将客户分类上有广泛应用。

属于无监督式学习的算法有：关联规则，K-means 聚类算法等。

3. 强化学习

这个算法可以训练程序做出某一决定。程序在某一情况下尝试所有的可能行动，记录不同行动的结果并试着找出最好的一次尝试来做决定。

属于这一类算法的有马尔可夫决策过程。

常见的机器学习算法

以下是最常用的机器学习算法，大部分数据问题都可以通过它们解决：

1.线性回归 (Linear Regression)

2.逻辑回归 (Logistic Regression)

3.决策树 (Decision Tree)

4.支持向量机（SVM）

5.朴素贝叶斯 (Naive Bayes)

6.K 邻近算法（KNN）

7.K-均值算法（K-means）

8.随机森林 (Random Forest)

9.降低维度算法（Dimensionality Reduction Algorithms）

10.Gradient Boost 和 Adaboost 算法

1.线性回归（Linear Regression）

线性回归是利用连续性变量来估计实际数值（例如房价，呼叫次数和总销售额等）。我们通过线性回归算法找出自变量和因变量间的最佳线性关系，图形上可以确定一条最佳直线。这条最佳直线就是回归线。这个回归关系可以用 Y=aX+b 表示。

我们可以假想一个场景来理解线性回归。比如你让一个五年级的孩子在不问同学具体体重多少的情况下，把班上的同学按照体重从轻到重排队。这个孩子会怎么做呢？他有可能会通过观察大家的身高和体格来排队。这就是线性回归！这个孩子其实是认为身高和体格与人的体重有某种相关。而这个关系就像是前一段的 Y 和 X 的关系。

在 Y=aX+b 这个公式里：

-

Y- 因变量

-
-

a- 斜率

-
-

X- 自变量

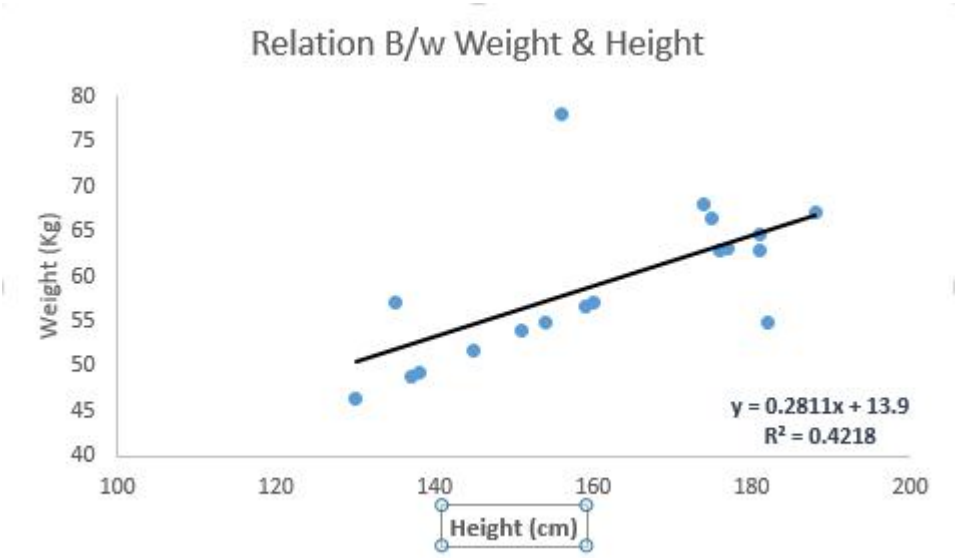
-
-

b- 截距

-

a 和 b 可以通过最小化因变量误差的平方和得到（最小二乘法）。

下图中我们得到的线性回归方程是 y=0.2811X+13.9。通过这个方程，我们可以根据一个人的身高得到他的体重信息。



线性回归主要有两种：一元线性回归和多元线性回归。一元线性回归只有一个自变量，而多元线性回归有多个自变量。拟合多元线性回归的时候，可以利用多项式回归（Polynomial Regression）或曲线回归 (Curvilinear Regression)。

Python 代码

#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)
```

- 1
- 2
- 3
- 4
- 5

- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24

R 代码

```
#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_datasetsy_train <- target_variables_values_training_datasetsx_test <- input_variables_values_test_datasetsx <- cbind(x_train,y_train)
# Train the model using the training sets and check scorelinear <- lm(y_train ~ ., data = x)summary(linear)
#Predict Outputpredicted= predict(linear,x_test)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14

- 15

2. 逻辑回归

别被它的名字迷惑了，逻辑回归其实是一个分类算法而不是回归算法。通常是利用已知的自变量来预测一个离散型因变量的值（像二进制值 0/1，是/否，真/假）。简单来说，它就是通过拟合一个逻辑函数（logit fuction）来预测一个事件发生的概率。所以它预测的是一个概率值，自然，它的输出值应该在 0 到 1 之间。

同样，我们可以用一个例子来理解这个算法。

假设你的一个朋友让你回答一道题。可能的结果只有两种：你答对了或没有答对。为了研究你最擅长的题目领域，你做了各种领域的题目。那么这个研究的结果可能是这样的：如果是一道十年级的三角函数题，你有 70%的可能性能解出它。但如果是一道五年级的历史题，你会的概率可能只有 30%。逻辑回归就是给你这样的概率结果。

回到数学上，事件结果的胜算对数（log odds）可以用预测变量的线性组合来描述：

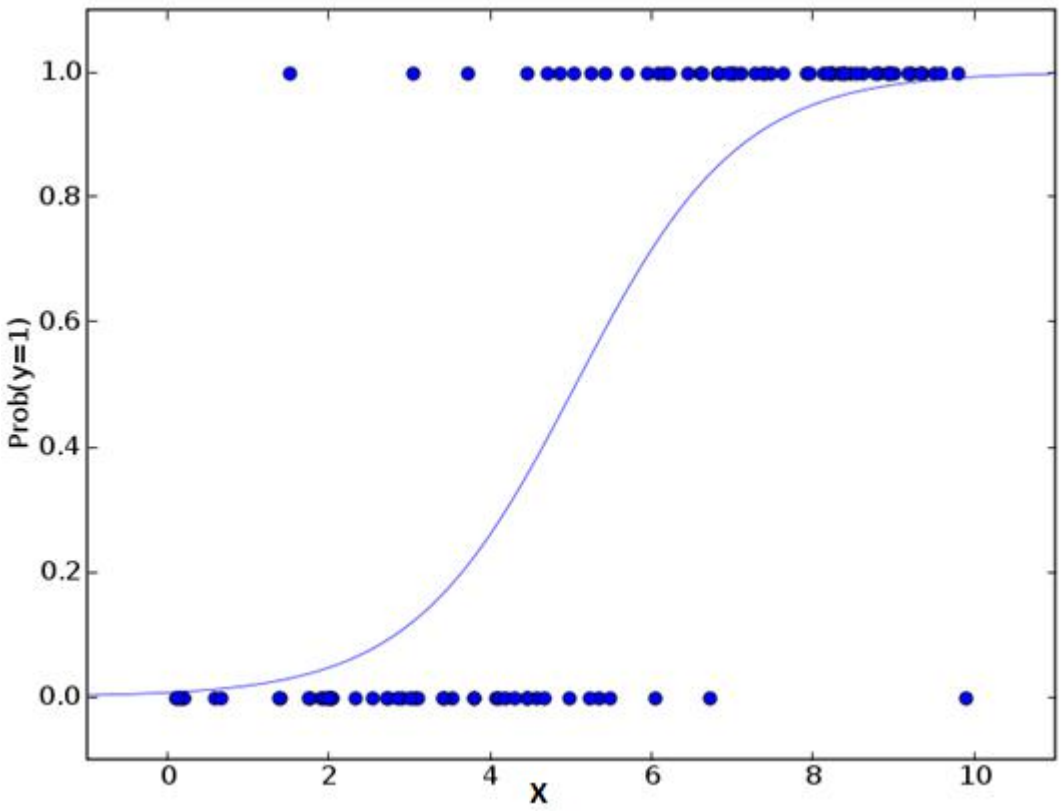
```
odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence
```

```
ln(odds) = ln(p/(1-p))
```

```
logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk
```

在这里，p 是我们感兴趣的事件出现的概率。它通过筛选出特定参数值使得观察到的样本值出现的概率最大化，来估计参数，而不是像普通回归那样最小化误差的平方和。

你可能会问为什么需要做对数呢？简单来说这是重复阶梯函数的最佳方法。因本篇文章旨不在此，这方面就不做详细介绍了。



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 test_dataset
# 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)
```

- 1
- 2
- 3
- 4
- 5
- 6

- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19

R 代码

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

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

延伸：

以下是一些可以尝试的优化模型的方法：

-

加入交互项（interaction）

-
-

减少特征变量

-
-

正则化（regularization）

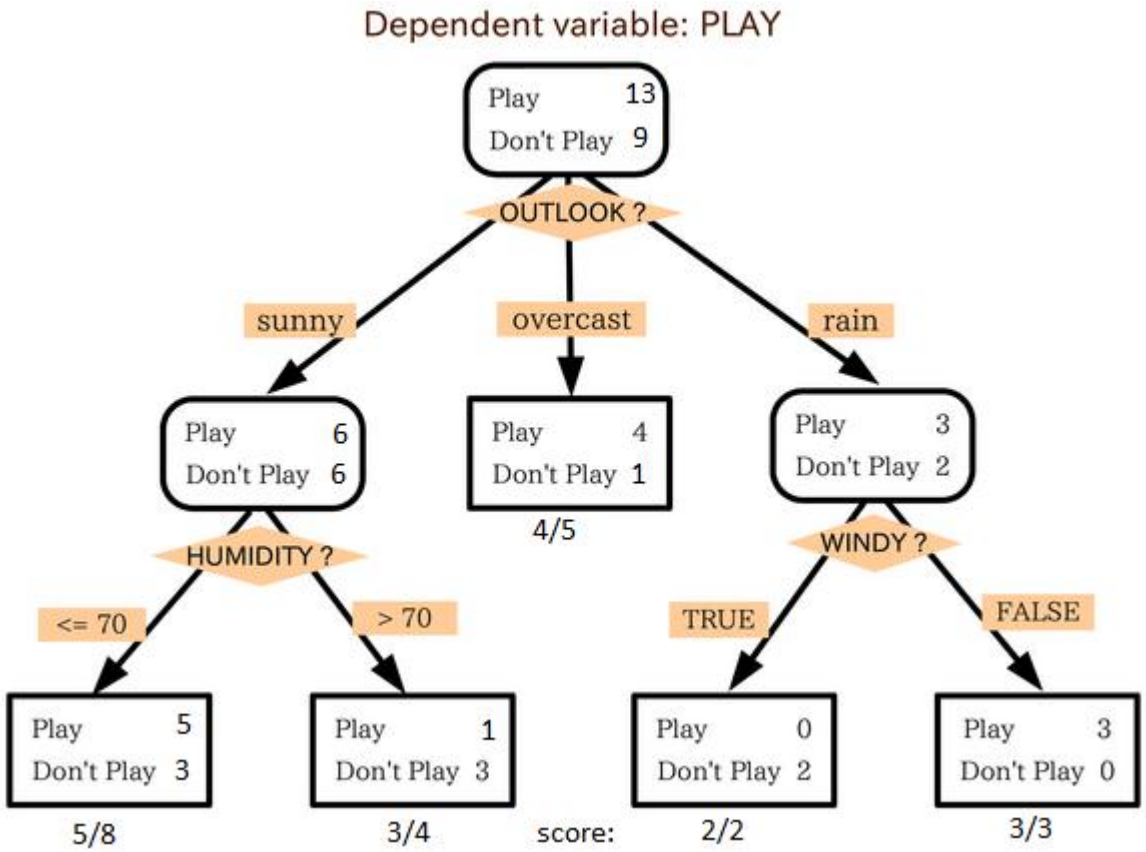
-
-

使用非线性模型

-

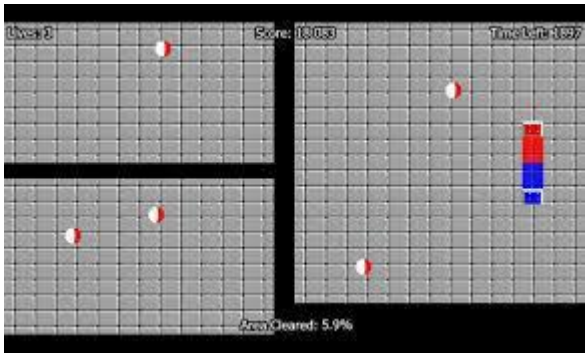
3. 决策树

这是我最喜欢也是能经常使用到的算法。它属于监督式学习，常用来解决分类问题。令人惊讶的是，它既可以运用于类别变量（categorical variables）也可以作用于连续变量。这个算法可以让我们把一个总体分为两个或多个群组。分组根据能够区分总体的最重要的特征变量/自变量进行。更详细的内容可以阅读这篇文章 Decision Tree Simplified。



从上图中我们可以看出，总体人群最终在玩与否的事件上被分成了四个群组。而分组是依据一些特征变量实现的。用来分组的具体指标有很多，比如 Gini，information Gain, Chi-square,entropy。

理解决策树原理的最好的办法就是玩 Jezzball 游戏。这是微软的一款经典游戏(见下图)。这个游戏的最终任务是在一个有移动墙壁的房间里，通过建造墙壁来尽可能地将房间分成尽量大的，没有小球的空间。



每一次你用建墙来分割房间，其实就是在将一个总体分成两部分。决策树也是用类似方法将总体分成尽量多的不同组别。

延伸阅读：[Simplified Version of Decision Tree Algorithms]

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 test_dataset

# Create tree object
model = tree.DecisionTreeClassifier(criterion='gini') # for
classification, here you can change the algorithm as gini or entropy
(information gain) by default it is gini

# 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)
```

- 1
- 2
- 3

- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18

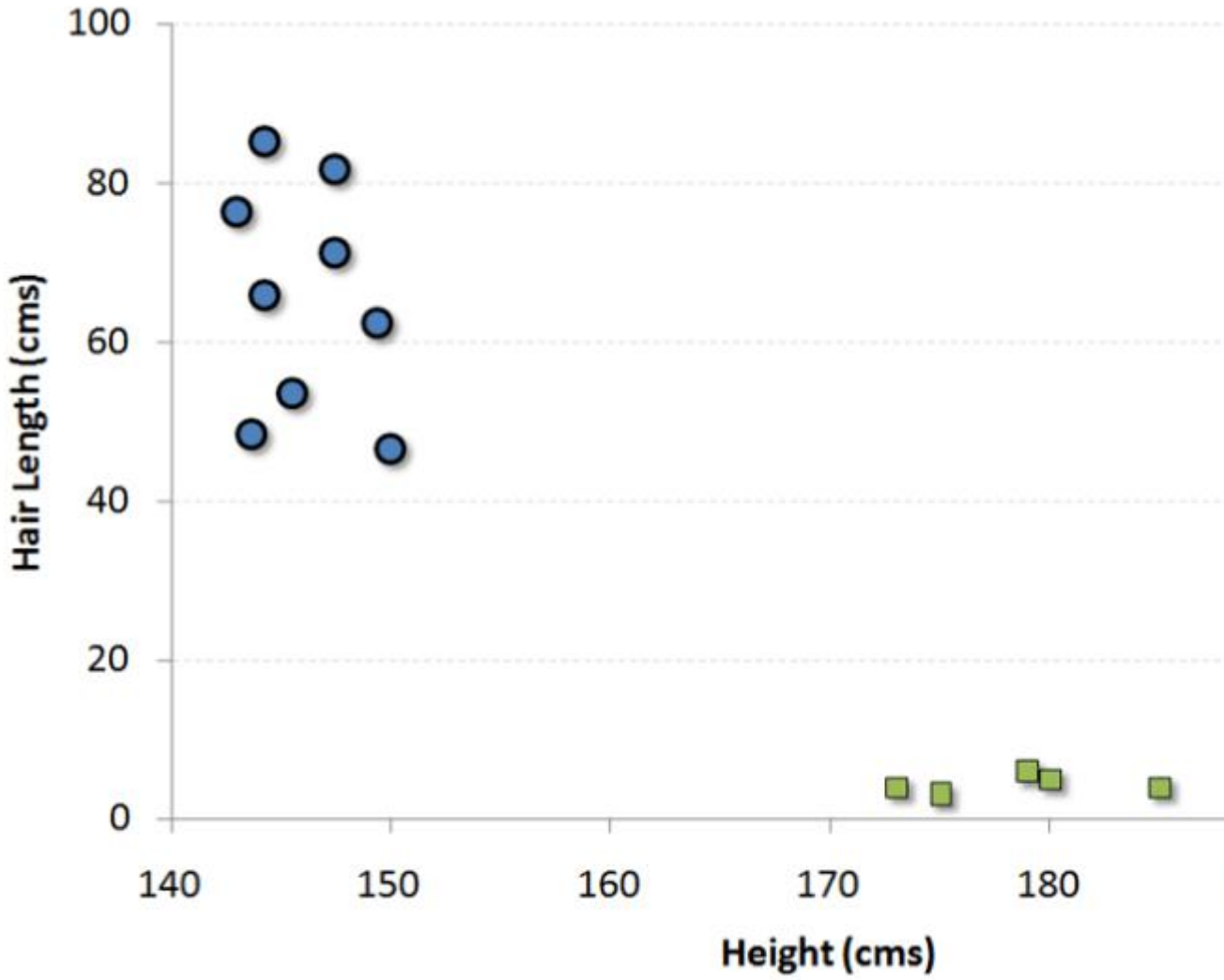
R 代码

```
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)
```

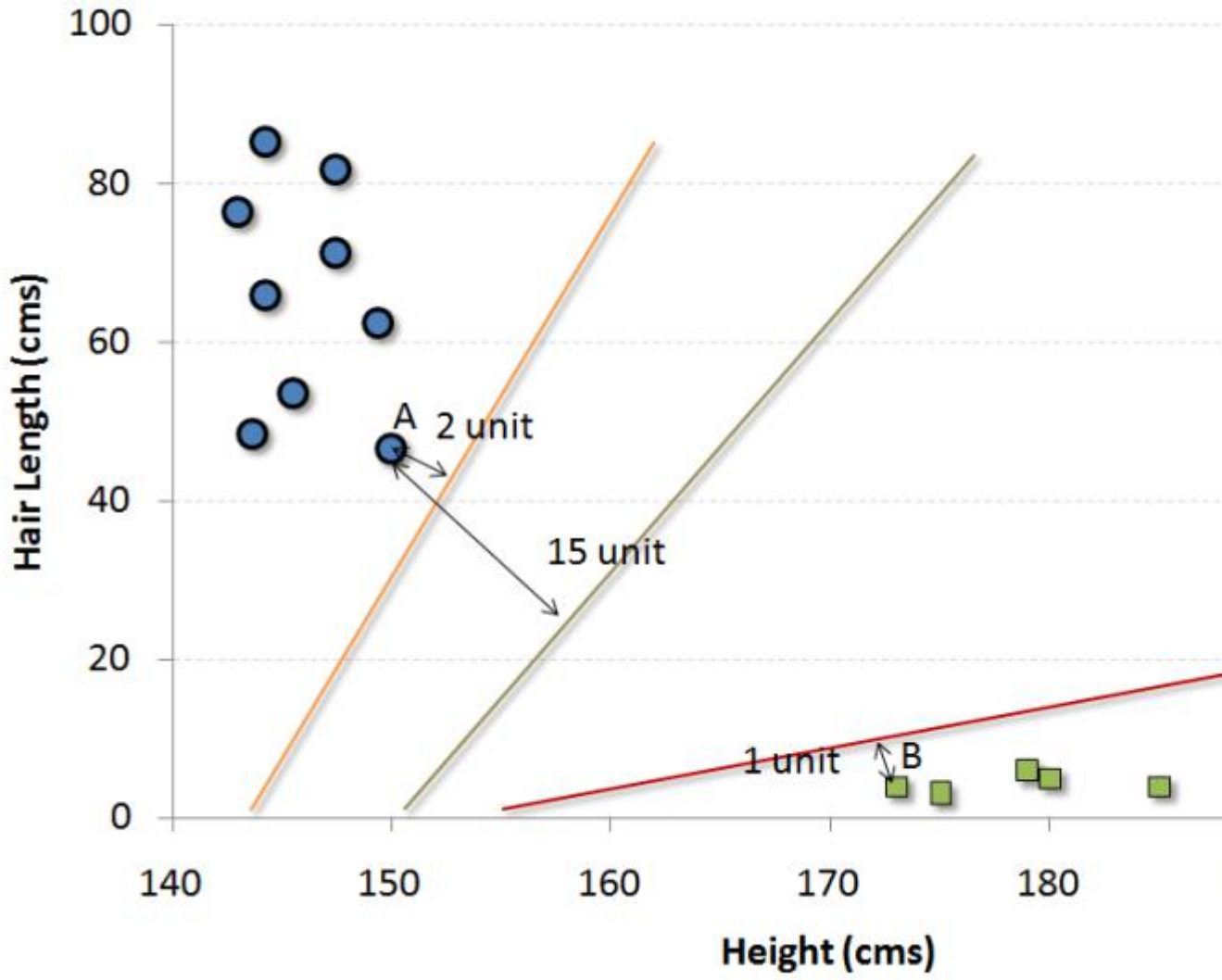
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

4. 支持向量机（SVM）

这是一个分类算法。在这个算法中我们将每一个数据作为一个点在一个 n 维空间上作图（n 是特征数），每一个特征值就代表对应坐标值的大小。比如说我们有两个特征：一个人的身高和发长。我们可以将这两个变量在一个二维空间上作图，图上的每个点都有两个坐标值（这些坐标轴也叫做支持向量）。



现在我们要在图中找到一条直线能最大程度将不同组的点分开。两组数据中距离这条线最近的点到这条线的距离都应该是最远的。



在上图中，黑色的线就是最佳分割线。因为这条线到两组中距它最近的点，点 A 和 B 的距离都是最远的。任何其他线必然会使得到其中一个点的距离比这个距离近。这样根据数据点分布在这条线的哪一边，我们就可以将数据归类。

更多阅读：[Simplified Version of Support Vector Machine]

我们可以把这个算法想成 n 维空间里的 JezzBall 游戏，不过有一些变动：

-

你可以以任何角度画分割线/分割面（经典游戏中只有垂直和水平方向）。

-
-

现在这个游戏的目的是把不同颜色的小球分到不同空间里。

-
-

小球是不动的。

-

Python 代码

```
#Import Libraryfrom sklearn import svm
#Assumed you have, X (predictor) 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 link, for mo# re detail.

# 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 modelfit <-svm(y_train ~ ., data = x)summary(fit)
#Predict Output predicted= predict(fit,x_test)
```

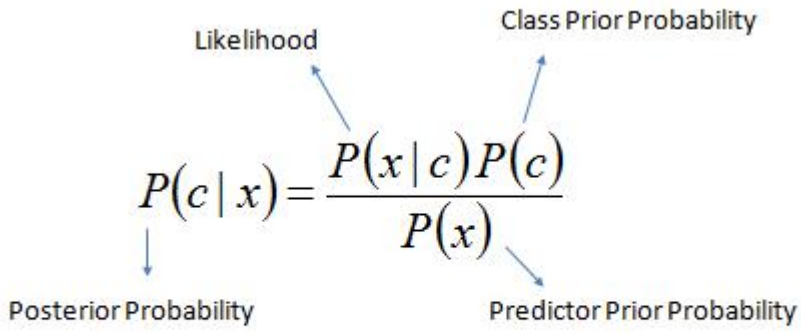
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

5. 朴素贝叶斯

这个算法是建立在贝叶斯理论上的分类方法。它的假设条件是自变量之间相互独立。简言之，朴素贝叶斯假定某一特征的出现与其它特征无关。比如说，如果一个水果它是红色的，圆状的，直径大概 7cm 左右，我们可能猜测它为苹果。即使这些特征之间存在一定关系，在朴素贝叶斯算法中我们都认为红色，圆状和直径在判断一个水果是苹果的可能性上是相互独立的。

朴素贝叶斯的模型易于建造，并且在分析大量数据问题时效率很高。虽然模型简单，但很多情况下工作得比非常复杂的分类方法还要好。

贝叶斯理论告诉我们如何从先验概率 P(c),P(x)和条件概率 P(x|c)中计算后验概率 P(c|x)。算法如下：



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

-

P(c|x)是已知特征 x 而分类为 c 的后验概率。

-

-

P(c)是种类 c 的先验概率。

-
-

P(x|c)是种类 c 具有特征 x 的可能性。

-
-

P(x)是特征 x 的先验概率。

-

例子： 以下这组训练集包括了天气变量和目标变量“是否出去玩”。我们现在需要根据天气情况将人们分为两组：玩或不玩。整个过程按照如下步骤进行：

步骤 1：根据已知数据做频率表

步骤 2：计算各个情况的概率制作概率表。比如阴天（Overcast）的概率为 0.29，此时玩的概率为 0.64.

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	
Overcast		
Rainy	3	
Sunny	2	
All	5	
	=5/14	=
	0.36	

步骤 3：用朴素贝叶斯计算每种天气情况下玩和不玩的后验概率。概率大的结果为预测值。

提问：天气晴朗的情况下(sunny)，人们会玩。这句陈述是否正确？

我们可以用上述方法回答这个问题。P(Yes | Sunny)=P(Sunny | Yes) * P(Yes) / P(Sunny)。

这里，P(Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64。

那么，P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60>0.5,说明这个概率值更大。

当有多种类别和多种特征时，预测的方法相似。朴素贝叶斯通常用于**文本分类和多类别分类**问题。

Python 代码

```
#Import Libraryfrom sklearn.naive_bayes import GaussianNB
#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset

# Create SVM classification object model = GaussianNB() # there is other
distribution for multinomial classes like Bernoulli Naive Bayes, Refer
link


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

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

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

R 代码

```
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)
```

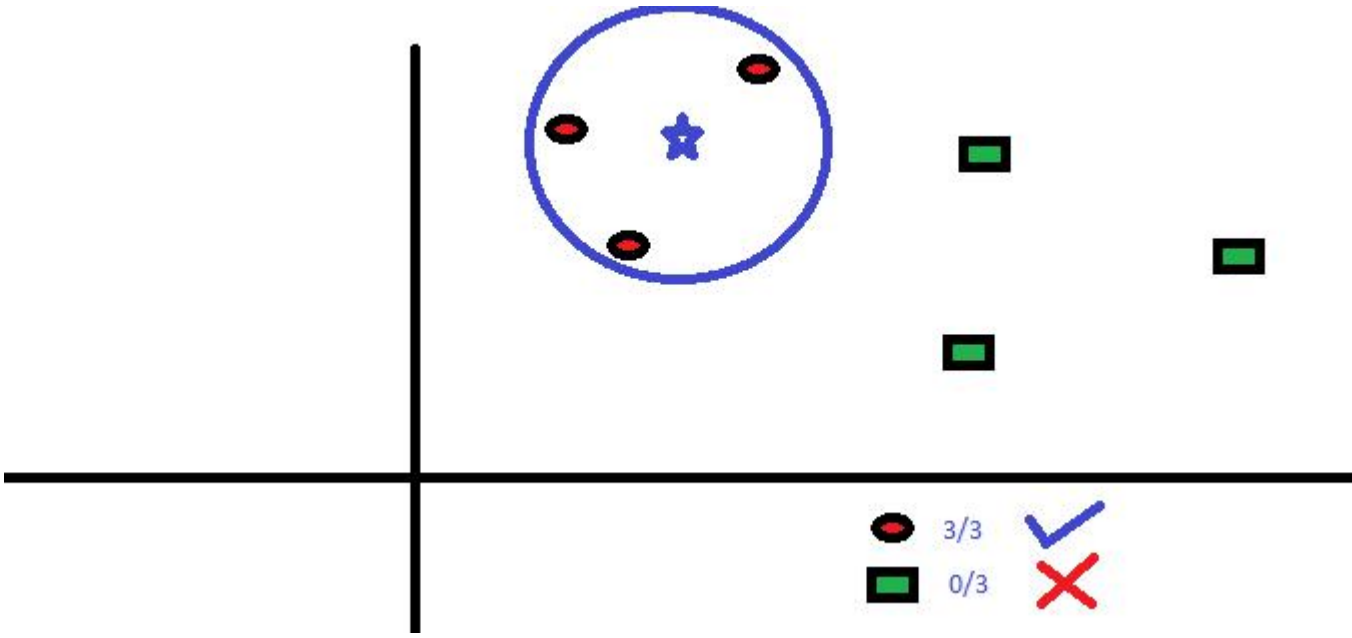
- 1

- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

6. KNN（K-邻近算法）

这个算法既可以解决分类问题，也可以用于回归问题，但工业上用于分类的情况更多。 KNN 先记录所有已知数据，再利用一个距离函数，找出已知数据中距离未知事件最近的 K 组数据，最后按照这 K 组数据里最常见的类别预测该事件。

距离函数可以是欧式距离，曼哈顿距离，闵氏距离 (Minkowski Distance), 和汉明距离（Hamming Distance）。前三种用于连续变量，汉明距离用于分类变量。如果 K=1，那问题就简化为根据最近的数据分类。K 值的选取时常是 KNN 建模里的关键。



KNN 在生活中的运用很多。比如，如果你想了解一个不认识的人，你可能就会从这个人的好朋友和圈子中了解他的信息。

在用 KNN 前你需要考虑到：

-

KNN 的计算成本很高

-
-

所有特征应该标准化数量级，否则数量级大的特征在计算距离上会有偏移。

-
-

在进行 KNN 前预处理数据，例如去除异常值，噪音等。

-

Python 代码

```
#Import Libraryfrom sklearn.neighbors import KNeighborsClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset
# 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)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13

R 代码

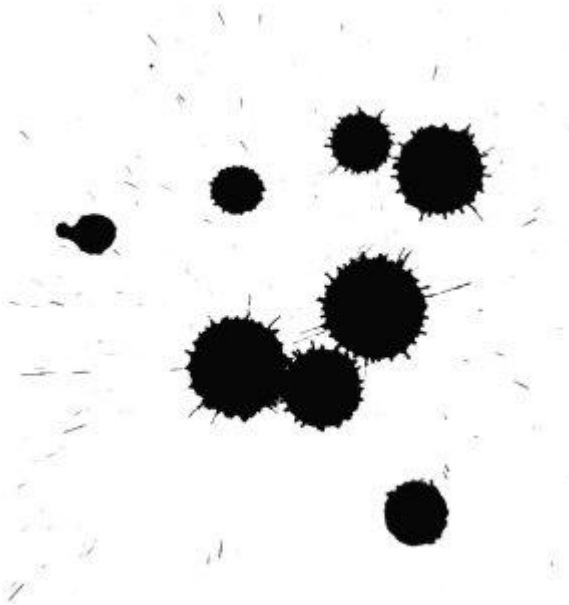
```
library(knn)x <- cbind(x_train,y_train)
# Fitting modelfit <-knn(y_train ~ ., data = x,k=5)summary(fit)
#Predict Output predicted= predict(fit,x_test)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

7. K 均值算法（K-Means）

这是一种解决聚类问题的非监督式学习算法。这个方法简单地利用了一定数量的集群（假设 K 个集群）对给定数据进行分类。同一集群内的数据点是同类的，不同集群的数据点不同类。

还记得你是怎样从墨水渍中辨认形状的么？K 均值算法的过程类似，你也要通过观察集群形状和分布来判断集群数量！



K 均值算法如何划分集群：

1.

从每个集群中选取 K 个数据点作为质心（centroids）。

2.
3.

将每一个数据点与距离自己最近的质心划分在同一集群，即生成 K 个新集群。

4.
5.

找出新集群的质心，这样就有了新的质心。

6.
7.

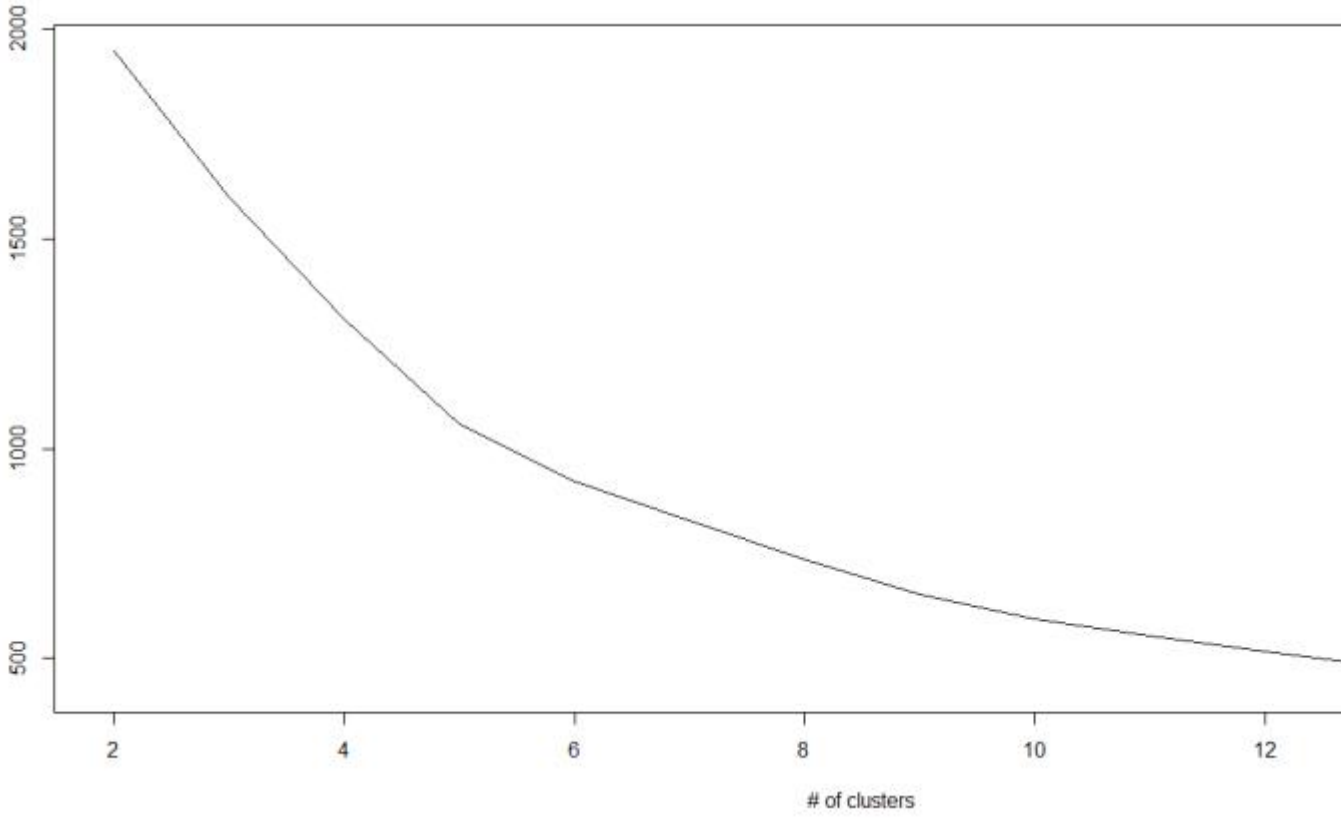
重复 2 和 3，直到结果收敛，即不再有新的质心出现。

8.

怎样确定 K 的值：

如果我们在每个集群中计算集群中所有点到质心的距离平方和，再将不同集群的距离平方和相加，我们就得到了这个集群方案的总平方和。

我们知道，随着集群数量的增加，总平方和会减少。但是如果用总平方和对 K 作图，你会发现在某个 K 值之前总平方和急速减少，但在这个 K 值之后减少的幅度大大降低，这个值就是最佳的集群数。



Python 代码

```
#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)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

- 11
- 12

R 代码

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

- 1
- 2

8. 随机森林

随机森林是对决策树集合的特有名称。随机森林里我们有多个决策树（所以叫“森林”）。为了给一个新的观察值分类，根据它的特征，每一个决策树都会给出一个分类。随机森林算法选出投票最多的分类作为分类结果。

怎样生成决策树：

1.

如果训练集中有 N 种类别，则有重复地随机选取 N 个样本。这些样本将组成培养决策树的训练集。

2.
3.

如果有 M 个特征变量，那么选取数 m << M，从而在每个节点上随机选取 m 个特征变量来分割该节点。m 在整个森林养成中保持不变。

4.
5.

每个决策树都最大程度上进行分割，没有剪枝。

6.

比较决策树和调节模型参数可以获取更多该算法细节。我建议读者阅读这些文章：

1.

Introduction to Random forest – Simplified

2.

3.

Comparing a CART model to Random Forest (Part 1)

4.
5.

Comparing a Random Forest to a CART model (Part 2)

6.
7.

Tuning the parameters of your Random Forest model

8.

Python 代码

```
#Import Library
from sklearn.ensemble import RandomForestClassifier#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# 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)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12

R 代码

```
library(randomForest)x <- cbind(x_train,y_train)# Fitting modelfit <-
randomForest(Species ~ ., x,ntree=500)summary(fit)
#Predict Output predicted= predict(fit,x_test)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

9.降维算法（Dimensionality Reduction Algorithms）

在过去的 4-5 年里，可获取的数据几乎以指数形式增长。公司/政府机构/研究组织不仅有了更多的数据来源，也获得了更多维度的数据信息。

例如：电子商务公司有了顾客更多的细节信息，像个人信息，网络浏览历史，个人喜恶，购买记录，反馈信息等，他们关注你的私人特征，比你天天去的超市里的店员更了解你。

作为一名数据科学家，我们手上的数据有非常多的特征。虽然这听起来有利于建立更强大精准的模型，但它们有时候反倒也是建模中的一大难题。怎样才能从 1000 或 2000 个变量里找到最重要的变量呢？这种情况下降维算法及其他算法，如决策树，随机森林，PCA，因子分析，相关矩阵，和缺省值比例等，就能帮我们解决难题。

进一步的了解可以阅读 Beginners Guide To Learn Dimension Reduction Techniques。

Python 代码

更多信息在这里

```
#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_features)# 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)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12

R 代码

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

- 1
- 2
- 3
- 4

10.Gradient Boosing 和 AdaBoost

GBM 和 AdaBoost 都是在有大量数据时提高预测准确度的 boosting 算法。Boosting 是一种集成学习方法。它通过有序结合多个较弱的分类器/估测器的估计结果来提高预测准确度。这些 boosting 算法在 Kaggle，AV Hackthon, CrowdAnalytix 等数据科学竞赛中有出色发挥。

更多阅读： Know about Gradient and AdaBoost in detail

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 test_dataset# Create Gradient Boosting Classifier object
model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)
# Train the model using the training sets and check score
```

```
model.fit(X, y)#Predict Output
predicted= model.predict(x_test)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

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]
```

- 1
- 2
- 3
- 4
- 5
- 6

GradientBoostingClassifier 和随机森林是两种不同的 boosting 分类树。人们经常提问 这两个算法有什么不同。

结束语

至此我相信读者对于常用的机器学习算法已经有了一定了解。写这篇文章并且提供 R 和 Python 的代码就是为了让你可以立马着手学习。动起手来去练一练吧，加深对这些算法过程的认识，运用他们，你会喜欢上机器学习的！

附：英文原文地址
http://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/