機器學習10種經典算法的Python實現

機器學習 作者：dicksonjyl560101 時間：2019-08-13 09:40:43 505 0

廣義來說，有三種機器學習算法

1、 監督式學習

工作機制：這個算法由一個目標變量或結果變量（或因變量）組成。這些變量由已知的一系列預示變量（自變量）預測而來。利用這一系列變量，我們生成一個將輸入值映射到期望輸出值的函數。這個訓練過程會一直持續，直到模型在訓練數據上獲得期望的精確度。監督式學習的例子有：回歸、決策樹、隨機森林、K – 近鄰算法、邏輯回歸等。

2、非監督式學習

工作機制：在這個算法中，沒有任何目標變量或結果變量要預測或估計。這個算法用在不同的組內聚類分析。這種分析方式被廣泛地用來細分客戶，根據干預的方式分為不同的用戶組。非監督式學習的例子有：關聯算法和K – 均值算法。

3、強化學習

工作機制：這個算法訓練機器進行決策。它是這樣工作的：機器被放在一個能讓它通過反複試錯來訓練自己的環境中。機器從過去的經驗中進行學習，並且嘗試利用了解最透徹的知識作出精確的商業判斷。強化學習的例子有馬爾可夫決策過程。

常見機器學習算法名單

這裡是一個常用的機器學習算法名單。這些算法幾乎可以用在所有的數據問題上：

- 線性回歸
- 邏輯回歸
- 決策樹
- SVM
- 樸素貝葉斯
- K最近鄰算法
- K均值算法
- 隨機森林算法
- 降維算法
- Gradient Boost 和Adaboost 算法

1、線性回歸

線性回歸通常用於根據連續變量估計實際數值（房價、呼叫次數、總銷售額等）。我們通過擬合最佳直線來建立自變量和因變量的關係。這條最佳直線叫做回歸線，並且用Y= a *X + b 這條線性等式來表示。

理解線性回歸的最好辦法是回顧一下童年。假設在不問對方體重的情況下，讓一個五年級的孩子按體重從輕到重的順序對班上的同學排序，你覺得這個孩子會怎麼做？他（她）很可能會目測人們的身高和體型，綜合這些可見的參數來排列他們。這是現實生活中使用線性回歸的例子。實際上，這個孩子發現了身高和體型與體重有一定的關係，這個關係看起來很像上面的等式。

在這個等式中：

- Y：因變量
- a：斜率
- x：自變量
- b ：截距

係數a 和b 可以通過最小二乘法獲得。

參見下例。我們找出最佳擬合直線y=0.2811x+13.9 。已知人的身高，我們可以通過這條等式求出體重。

dicksonjyl56

專家姓名：金玉林

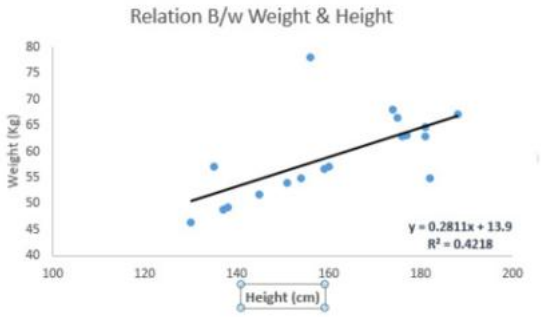
行萬里路，讀萬卷書，閱無
愛吃湘菜，川菜，粵菜與杭
食。攝影愛好者，遊遍名川
賞遍人間春色。愛看影，尤
大片。英文名：DicksonJin，
BOY。精通SAP供應鏈諮詢
人工智能。

註冊時間： 2014-08-27

博文量
2236

最新文章

不經意之中成就他人，
告別2019：屬於深度學
一文看盡2019年NLP前
深度學習的應用與實踐
SAP S4HANA如何取到
SAP 對HU做貨物移動
交通安全培訓，讓我無
SAP MM 一個含有多個
國外的月亮並不比國內
清華2019最新AI發展報



線性回歸的兩種主要類型是一元線性回歸和多元線性回歸。一元線性回歸的特點是只有一個自變量。多元線性回歸的特點正如其名，存在多個自變量。找最佳擬合直線的時候，你可以擬合到多項或者曲線回歸。這些就被叫做多項或曲線回歸。

Python 代碼

```
#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)
```

2、邏輯回歸

別被它的名字迷惑了！這是一個分類算法而不是一個回歸算法。該算法可根據已知的一系列因變量估計離散數值（比方說二進制數值0 或1 ，是或否，真或假）。簡單來說，它通過將數據擬合進一個邏輯函數來預估一個事件出現的概率。因此，它也被叫做邏輯回歸。因為它預估的是概率，所以它的輸出值大小在0 和1 之間（正如所預計的一樣）。

讓我們再次通過一個簡單的例子來理解這個算法。

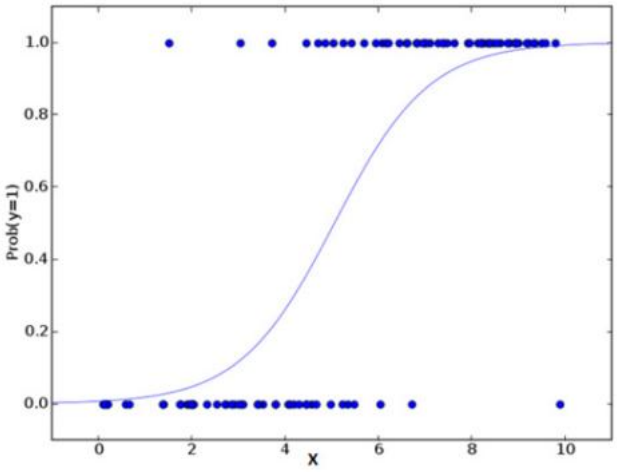
假設你的朋友讓你解開一個謎題。這只會有兩個結果：你解開了或是你沒有解開。想像你要解答很多道題來找出你所擅長的主題。這個研究的結果就會像是這樣：假設題目是一道十年級的三角函數題，你有70%的可能會解開這道題。然而，若題目是個五年級的歷史題，你只有30%的可能性回答正確。這就是邏輯回歸能提供給你的信息。

從數學上看，在結果中，機率的對數使用的是預測變量的線性組合模型。

```
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
```

在上面的式子裡，p 是我們感興趣的特徵出現的概率。它選用使觀察樣本值的可能性最大化的值作為參數，而不是通過計算誤差平方和的最小值（就如一般的回歸分析用到的一樣）。

現在你也許要問了，為什麼我們要求出對數呢？簡而言之，這種方法是複制一個階梯函數的最佳方法之一。我本可以更詳細地講述，但那就違背本篇指南的主旨了。



Python代碼

```
#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)
```

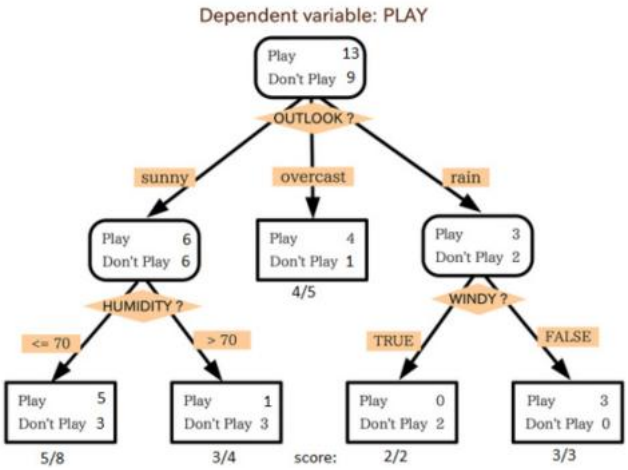
更进一步：

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

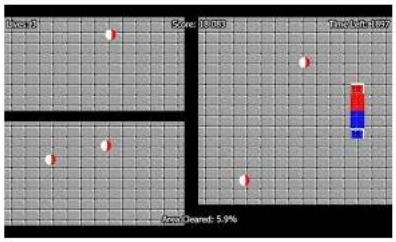
这是我最喜爱也是最频繁使用的算法之一。这个监督式学习算法通常被用于分类问题。令人惊奇的是，它同时适用于分类变量和连续因变量。在这个算法中，我们将总体分成两个或更多的同类群。这是根据最重要的属性或者自变量来分成尽可能不同的组别。想要知道更多，可以阅读：简化决策树 。



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

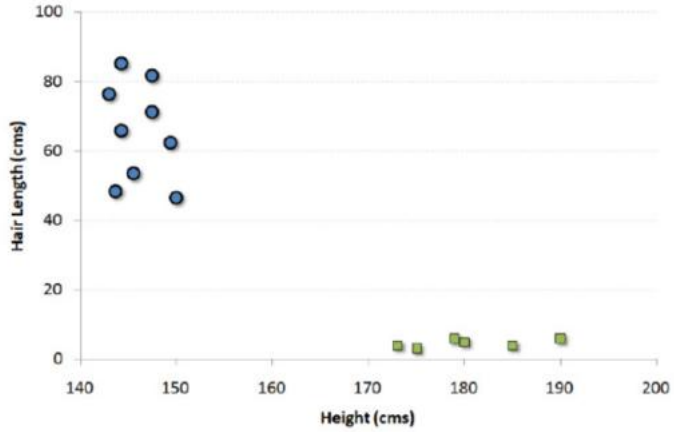
Python代码

```
#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 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)
```

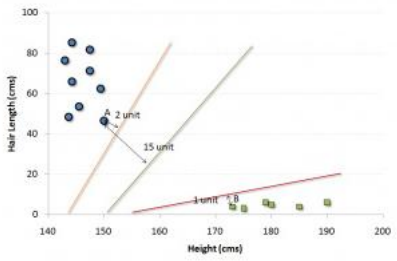
4、支持向量机

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 link, for mo# re detail.

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)
```

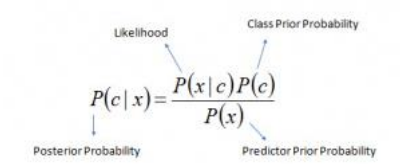
```
#Predict Output

predicted= model.predict(x_test)
```

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：



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

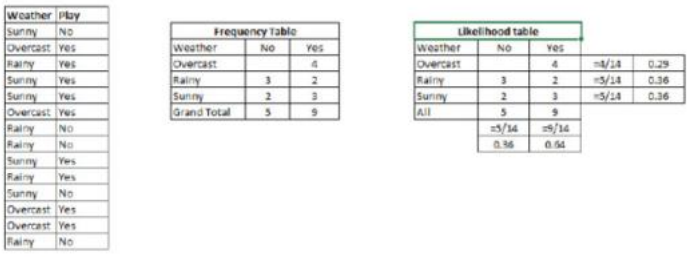
在这里，

- 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

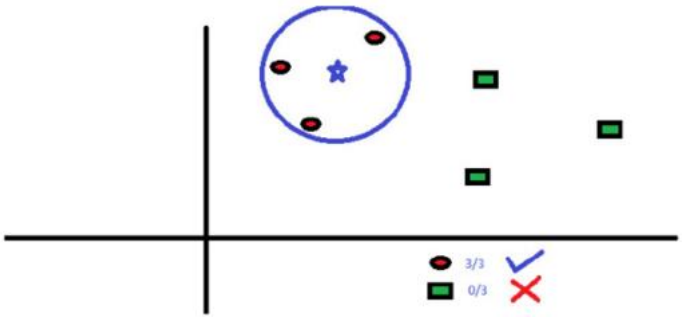
```
#Import Library

from sklearn.naive_bayes import GaussianNB

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

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

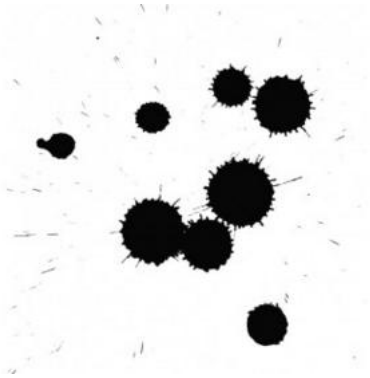
```
#Import Library

from sklearn.neighbors import KNeighborsClassifier

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

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。



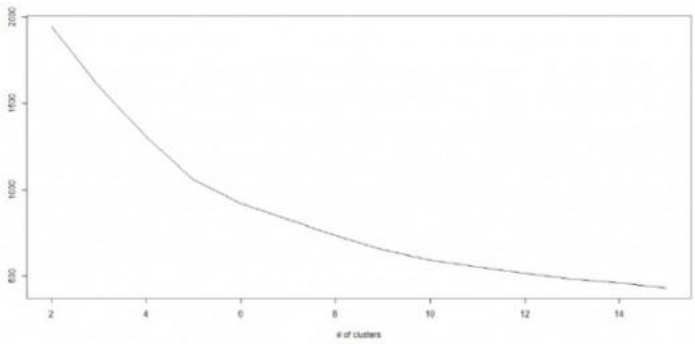
K – 均值算法怎样形成集群：

- K – 均值算法给每个集群选择k个点。这些点称作为质心。
- 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。
- 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。
- 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

```
#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)
```

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：

- 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。
- 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。
- 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：

- 随机森林入门—简化版
- 将 CART 模型与随机森林比较（上）
- 将随机森林与 CART 模型比较（下）
- 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 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)
```

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_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)

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

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 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)
```

GradientBoostingClassifier 和隨機森林是兩種不同的boosting 樹分類器。人們常常問起這兩個算法之間的區別。

https://www.toutiao.com/a6724079298635891211/

上一篇： 史上最簡單的人臉識別項目登上GitHub趨勢榜 下一篇： 這位22歲華裔青年，用3年創造出矽谷最新…

請登錄後發表評論 登錄

全部評論