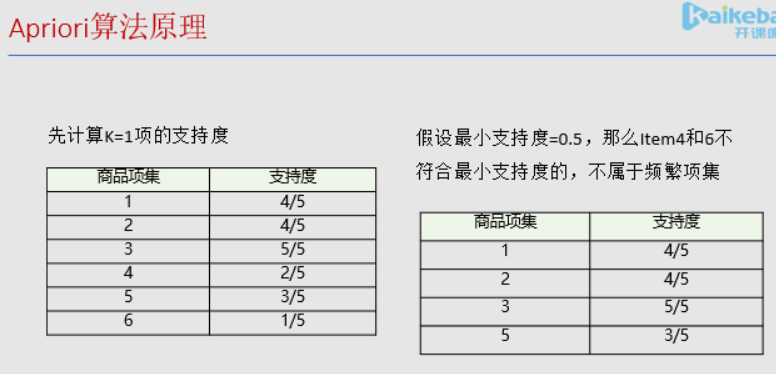
L2 挖掘数据中的关联规则

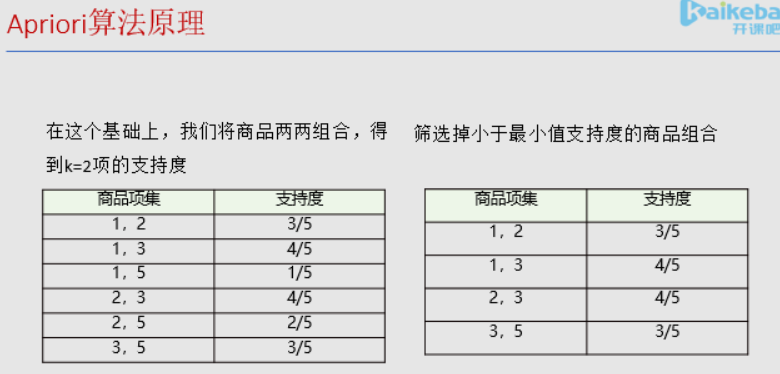
1. Theory
   1. 什么是关联规则
      1. 关联规则 (Association Rules，或者是 Basket Analysis) 是形如X→Y的蕴涵式，其中， X和Y分别称为关联规则的先导(antecedent或left-hand-side, LHS)和后继(consequent或right-hand-side, RHS) 。 关联规则XY，存在支持度和信任度。解释了：如果一个消费者购买了产品A，那么他有多大几率会购买产品B? 关联分析可以用于回答“哪些商品经常被同时购买？”之类的问题
   2. Apriori算法的流程是怎样的
      1. Apriori算法的流程： Step1，K=1，计算K项集的支持度； Step2，筛选掉小于最小支持度的项集； Step3，如果项集为空，则对应K-1项集的结果为最终结果。 否则K=K+1，重复1-3步。

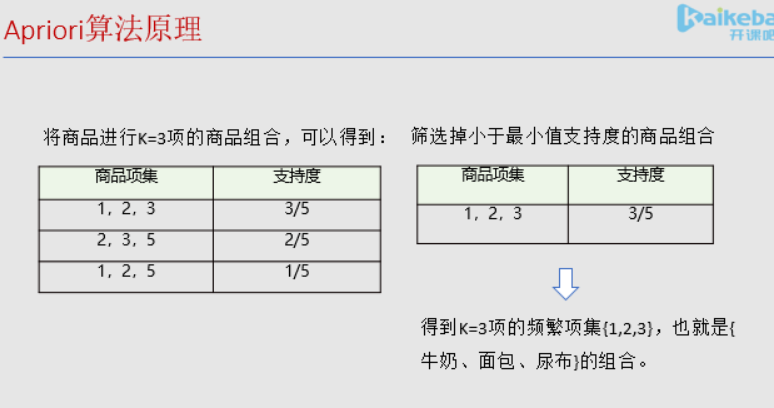
Apriori算法就是查找频繁项集(frequent itemset)的过程

频繁项集：支持度大于等于最小支持度(Min Support)阈值的项集。

非频繁项集：支持度小于最小支持度的项集







* 1. Apriori算法存在哪些不足
     1. apriori算法多次扫描交易数据库，每次利用候选频繁集产生频繁集. 因此对比FP-Growth, 算法效率相对低。

Apriori算法是发现频繁项集的一种方法，Apriori算法的两个输入参数分别是最小支持度和数据集，该算法首先会生成所有单个元素的项集列表。接着扫描数据集来查看哪些项集满足最小支持度要求，那些不满足最小支持度的集合会被去掉，然后，对剩下来的集合进行组合以生成包含两个元素的项集，接下来，再重新扫描交易记录，去掉不满足最小支持度的项集。该过程重复进行直到所有项集都被去掉。

经典的关联规则挖掘算法包括Apriori算法和FP-growth算法。apriori算法多次扫描交易数据库，每次利用候选频繁集产生频繁集；而FP-growth则利用树形结构，无需产生候选频繁集而是直接得到频繁集，大大减少扫描交易数据库的次数，从而提高了算法的效率。但是apriori的算法扩展性较好，可以用于并行计算等领域。

* 1. 相关性分析与回归分析
     1. 定义

相关性分析（correlation analysis），考虑的是两个或两个以上处于同等地位的随机变量间的相关关系的统计分析方法，查看两个变量之间是否有相关性，比如人的身高和体重。 回归分析（regression analysis)，考虑的是两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，是用一个变量对另一个变量进行预测，自变量和因变量的确定是很重要的，两个位置互换了其意义就完全不同了。

* + 1. 联系

相关分析是回归分析的前提，两个变量间只有在存在相关性的前提下才能进行回归分析。比如人的头发长短和工资之间毫无相关性， 那么就无法用头发长度对工资进行解释和预测，同样也无法用工资来预测头发的长度。 但是对于受教育程度和工资这两个变量，因为它们之间存在相关性， 所以就可以用受教育程度对工资进行预测，但这也仅仅止步于回归分析，更进一步的因果分析（即因为受教育程度高，所以工资高）还需要勇其他方法来确定。

* + 1. 区别

变量的位置顺序。依旧其目的本质不同，相关分析（测度客观存在的变量间联系的紧密程度）和回归分析（采用一个/多个变量对另一个变量进行预测， 依靠变量间的依赖关系）的变量的位置顺序是很重要的，尤其是对于回归分析，一定要搞清楚是对哪个变量进行预测，进而寻找与其具有相关性的可能对其进行解释的变量。

变量的结果数值。在1的基础上可以看出，对于相关分析，变量确定，相关系数也是确定的，当然采用不同的相关性度量指标结果也不一样。 而对于回归分析，在关心的因变量确定的情况下，为了对其进行解释，我们可以寻找一个或者多个的自变量，构建不同的回归模型，所得的结果系数也大不相同。

变量的显著性检验。对于相关性，其数值位于[-1,1]之间，显著性检验的目的是为了检验数值与0（没有相关性）之间是否存在统计上的不同，如果相关性数值在统计上与0存在差别， 那么我们称两个变量存在相关性，并依据数值的正负判断是存在正相关还是负相关。然而对于回归分析，该情况稍微复杂了一点点，在所构建的回归方程中， 一方面需要对自变量对因变量的贡献度（即回归系数）的显著性进行检验，该道理与相关性相同；另一方面需要对整体构建的方程对因变量的解释度进行检验， 这个一般采用R2进行衡量；同时，还需要对整个模型的效能进行评估，这个时候需要采用诸如AIC或者BIC进行评判；同时，还需要注意变量之间的多重共线性等等。总之，回归分析时需要考虑的东西很多

1. Tools
   1. 挖掘频繁项集和关联规则
      1. BreadBasket

apriori\_breadbasket.py

See attached file(s): [apriori\_breadbasket.py](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\apriori_breadbasket.py)

BreadBasket\_DMS.csv

See attached file(s): [BreadBasket\_DMS.csv](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\BreadBasket_DMS.csv)

* 1. 挖掘频繁项集和关联规则
     1. MovieLens

movies\_apriori.py

See attached file(s): [movies\_apriori.py](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\movies_apriori.py)

movies.csv

See attached file(s): [movies.csv](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\movies.csv)

* 1. 数据抓取，挖掘频繁项集和关联规则
     1. MovieActors

movie\_actors.csv

See attached file(s): [movie\_actors.csv](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\movie_actors.csv)

movie\_actors\_apriori.py

See attached file(s): [movie\_actors\_apriori.py](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\movie_actors_apriori.py)

movie\_actors\_download.py

See attached file(s): [movie\_actors\_download.py](file:///D:\Profile\ttrfakd\Desktop\data%20mining\BI%20KW2\training\L2%20挖掘数据中的关联规则%20(2).docx%20-%20Attached%20Files\movie_actors_download.py)

* 1. LinearRegression
  2. PolynomialFeatures

1. Thinking
   1. 关联规则中的支持度、置信度和提升度代表的什么，如何计算
      1. 概念

商品组合：一个商品组合包含一个以上的产品。 关联规则：如果一个商品组合 A 和另一个商品组合 B 同时出现在一个或者多个顾客的购物清单上，那么我们就说这两个商品组合中存在关联规则。 如果我们要研究买了 A 的顾客再买 B 的情况，这样的规则就表示为 A→B，→ 之前的称为 前导，→ 之后的称为 后继

antecedents：规则的先导 consequents：规则的后继 antecedent support：先导的支持度 consequent support：后继的支持度 support：规则的支持度 confidence：规则的置信度 lift：规则的提升度 leverage：杠杆率 conviction：确信度

* + 1. 支持度(Support)：

选择支持度的目的是找出同时被购买的两个商品，可以提高推荐转换率。 某个商品组合出现的次数Ai与总次数n之间的比例。 Support = Ai /n 10 个顾客的购物清单中出现了 4 次可乐，所以“可乐”的支持度是 4/10 = 0.4。出现了 3 次“可乐 + 啤酒”，所以“可乐 + 啤酒”的支持度是 3/10 = 0.3。

总共有10000个消费者购买了商品，

其中购买尿布的有1000人，

购买啤酒的有2000人，

购买面包的有500人，

同时购买尿布和啤酒的有800人，

同时购买尿布的面包的有100人。

关联规则

关联规则：用于表示数据内隐含的关联性，例如：购买尿布的人往往会购买啤酒。

支持度（support）

支持度：{X, Y}同时出现的概率，例如：{尿布，啤酒}同时出现的概率

[Math Processing Error]support=总人数同时购买{X,Y}的人数​

{尿布，啤酒}的支持度 = 800 / 10000 = 0.08

{尿布，面包}的支持度 = 100 / 10000 = 0.01

注意：{尿布，啤酒}的支持度等于{啤酒，尿布}的支持度，支持度没有先后顺序之分

* + 1. 置信度(Confidence)：

购买了商品 Ai 的顾客中，有多大比例购买商品 B i->j。 Confidenc (A->B) = B i->j / A i （ i: 在n中，A 独立出现次数 / j: 在n中， B 独立出现的次数 / i->j: 在n 中，A 出现情况下， B出现的次数 10次交易，4 位顾客购买了可乐，其中 3 位买了啤酒， 因此我们可以说存在一条从可乐到啤酒 的关联规则， 这条规则的置信度是 3/4 = 0.75。写作：置信度(可乐 → 啤酒) = 0.75。

置信度（confidence）

置信度：购买X的人，同时购买Y的概率，例如：购买尿布的人，同时购买啤酒的概率，而这个概率就是购买尿布时购买啤酒的置信度

[Math Processing Error]confidence（X−>Y）=购买X的人数同时购买{X,Y}的人数​

[Math Processing Error]confidence（Y−>X）=购买Y的人数同时购买{X,Y}的人数​

( 尿布 -> 啤酒 ) 的置信度 = 800 / 1000 = 0.8

( 啤酒 -> 尿布 ) 的置信度 = 800 / 2000 = 0.4

* + 1. 提升度(Lift)：

P(B|A)/P(B)，称之为A条件对B事务的提升度，即有A作为前提，对B出现的概率有什么样的影响， 如果提升度=1说明A和B没有任何关联， 如果<1，说明A事务和B事务是排斥的，>1， 我们认为A和B是有关联的，但是在具体的应用之中，我们认为物品和人的提升度差异很大。人的提升度通常比物品高。 例如 豆瓣案例黄渤 宁浩案例。

* + 1. 应用

如果销售数据中存在一定的关联规则，那么我们就可以用这几个指标来对关联规则进行评价： 首先，我们希望符合规律的商品/商品组合本身有足够销量，也就是有足够的 支持度，譬如就算发现了商品或者商品组合 A 和 B 之间存在很强的销售规律，但是买 A 和 B 的顾客数量或比例很少，那这样的规则意义也不大。 其次，我们希望商品组合 A 对商品组合 B 有足够强的 提升度 判读规则是否冗余的标准：如果一个更具体的关联规则的置信度比其更一般的规则更低， 那具体的关联规则为冗余规则，没有必要保留。 尿布 → 啤酒 的支持度是 0.5，置信度 0.83，提升度是 1.19，也说明可以向购买尿布的顾客推荐啤酒。 比较有意思的是 (尿布,面包)→ 啤酒 这条关联规则，支持度 0.4，置信度 0.8，提升度和 尿布 → 啤酒 对比一下，我们可以看出两点： 两个规则的后继都是 啤酒，而先导条件 (尿布,面包) 比 尿布 更加具体，也就是要求更高； 同时，我们会发现，更高的要求却没有带来更高的置信度。也就是说发现一个既买了尿布， 又买了面包的顾客再向他推荐啤酒，还不如一看到有人买尿布不用管他有没有买面包就直接推荐啤酒， 所以这条规则没有带来额外的价值，是冗余的。

Apriori算法就是查找频繁项集(frequent itemset)的过程 频繁项集：支持度大于等于最小支持度(Min Support)阈值的项集。 非频繁项集：支持度小于最小支持度的项集

* 1. 关联规则与协同过滤的区别
     1. 两种推荐算法的思考维度不同，很多时候，我们需要把多种推荐方法的结果综合起来做一个混合的推荐
     2. 关联规则是基于transaction，

商品组合使用的是购物篮分析，也就是Apriori算法. 关联规则没有利用“用户偏好”，而是基于购物订单进行的频繁项集挖掘. 当下的需求：推荐的基础是且只是当前一次的购买/点击 不需要考虑用户一定时期内的偏好，而是基于Transaction 只要能将数据转换成Transaction，就可以做购物篮分析： Step1、把数据整理成id=>item形式，转换成transaction Step2、设定关联规则的参数（support、confident）挖掘关联规则 Step3、按某个指标（lift、support等）对以关联规则排序

* + 1. 而协同过滤基于用户偏好（评分）

协同过滤计算的是相似度. 长期偏好： 基于用户历史的行为进行分析，建立一定时间内的偏好排序

* 1. 为什么我们需要多种推荐算法
     1. 基于关联规则的推荐算法： Apriori算法 FPGrowth算法 PrefixSpan算法

Apriori在计算的过程中存在的不足： 可能产生大量的候选集。因为采用排列组合的方式， 把可能的项集都组合出来了 每次计算都需要重新扫描数据集，计算每个项集的支持度。 浪费了计算空间和时间

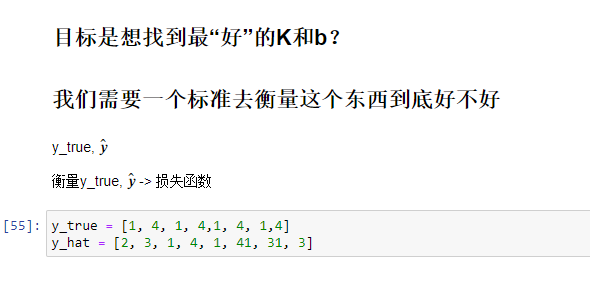
Apriori算法的流程： #Step1，K=1，计算K项集的支持度； #Step2，筛选掉小于最小支持度的项集； #Step3，如果项集为空，则对应K-1项集的结果为最终结果。 #否则K=K+1，重复1-3步。

在Apriori算法基础上提出了FP-Growth算法： 创建了一棵FP树来存储频繁项集。在创建前对不满足最小支持度的 项进行删除，减少了存储空间。 整个生成过程只遍历数据集2次，大大减少了计算量

* 1. 关联规则中的最小支持度、最小置信度该如何确定
     1. 最小支持度，最小置信度是实验出来的
     2. 最小支持度： 不同的数据集，最小值支持度差别较大。可能是0.01到0.5之间 可以从高到低输出前20个项集的支持度作为参考。数据集越大，选择的支持率越小。
     3. 最小置信度：可能是0.5到1之间
     4. 提升度：表示使用关联规则可以提升的倍数，是置信度与期望置信度的比值 提升度至少要大于1。
     5. Rule1 采用efficient\_apriori工具包 # 挖掘频繁项集和频繁规则， min\_confidence 筛选关联规则 # 先求出itemsets, 然后找itemsets之间关联，A->B, 筛选方式用 min\_confidence >= x
  2. 都有哪些常见的回归分析方法，评价指标是什么
     1. 回归分析是一种预测性的建模技术，它研究的是因变量（目标）和自变量（预测器）之间的关系。 这种技术通常用于预测分析，时间序列模型以及发现变量之间的因果关系。 例如，司机的鲁莽驾驶与道路交通事故数量之间的关系，最好的研究方法就是回归。 有各种各样的回归技术用于预测。这些技术主要有三个度量（自变量的个数，因变量的类型以及回归线的形状）
     2. Class

Linear

1. 第一种方法：最简单的方法，随机找！记录最优的值！目标是想找到最“好”的K和b？ 我们需要一个标准去衡量这个东西到底好不好



#加载数据 Load Data

import pandas as pd

import numpy as np

import sklearn as sk

#import matplotlib as plt

import scipy as sc

from sklearn.datasets import load\_boston

data = load\_boston()

data['feature\_names']

room\_index=5

X,y = data['data'],data['target']

print(X[1],y[1])

print(X.shape, len(y))

#matplotlib inline

import matplotlib.pyplot as plt

plt.scatter(X[:,5],y)

plt.show()

#目标：就是要找一个“最佳”的直线，来拟合卧室和房价的关系

#随机画了一根直线，结果发现，离得很远

import random

k,b = random.randint(-100,100), random.randint(-100,100)

def func(x):

return k\*x + b

X\_rm = X[:,5]

y\_hat = [func(x) for x in X\_rm]

plt.scatter(X[:,5],y)

plt.show()

plt.plot(X\_rm, y\_hat)

plt.show()

def draw\_room\_and\_price():

plt.scatter(X[:, 5], y)

def price(x, k, b):

return k\*x + b

k, b = random.randint(-100, 100), random.randint(-100, 100)

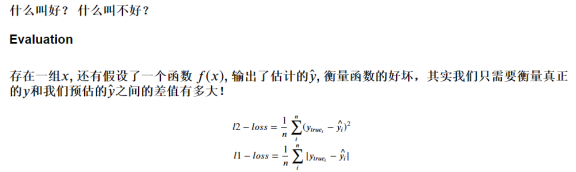
price\_by\_random\_k\_and\_b = [price(r, k, b) for r in X\_rm]

print('the random k : {}, b: {}'.format(k, b))

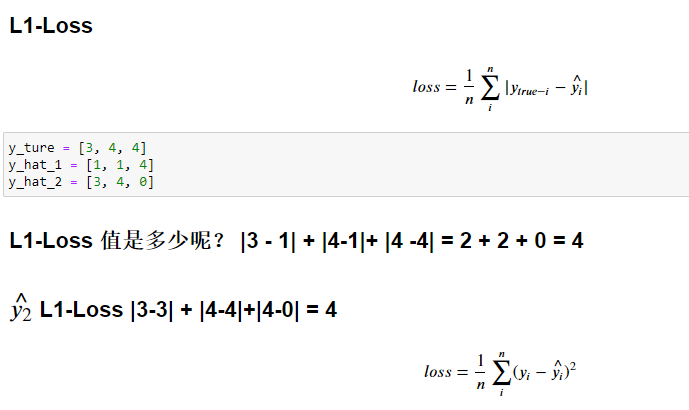
draw\_room\_and\_price()

plt.scatter(X\_rm, price\_by\_random\_k\_and\_b)

plt.show()



2. 方法 进行方向的调整 k的变化有两种： 增大和减小 b的变化也有两种：增大和减小



directions = [

(+1, -1),

(+1, +1),

(-1, -1),

(-1, +1)

]

def price(x, k, b):

return k\*x + b

trying\_times = 10000

best\_k = random.random() \* 100 - 200

best\_b = random.random() \* 100 - 200

next\_direction = random.choice(directions)

min\_cost = float('inf')

losses = []

scala = 0.3

for i in range(trying\_times):

current\_direction = next\_direction

k\_direction, b\_direction = current\_direction

current\_k = best\_k + k\_direction \* scala

current\_b = best\_b + b\_direction \* scala

price\_by\_random\_k\_and\_b = [price(r, current\_k, current\_b) for r in X\_rm]

cost = loss(list(y), price\_by\_random\_k\_and\_b)

if cost < min\_cost:

min\_cost = cost

best\_k, best\_b = current\_k,current\_b

print('在第{}， k和b更新了'.format(i))

losses.append((i, min\_cost))

next\_direction = current\_direction

else:

next\_direction = random.choice(list(set(directions) - {current\_direction}))

zip()

>>>a = [1,2,3]

>>> b = [4,5,6]

>>> c = [4,5,6,7,8]

>>> zipped = zip(a,b) # 打包为元组的列表

[(1, 4), (2, 5), (3, 6)]

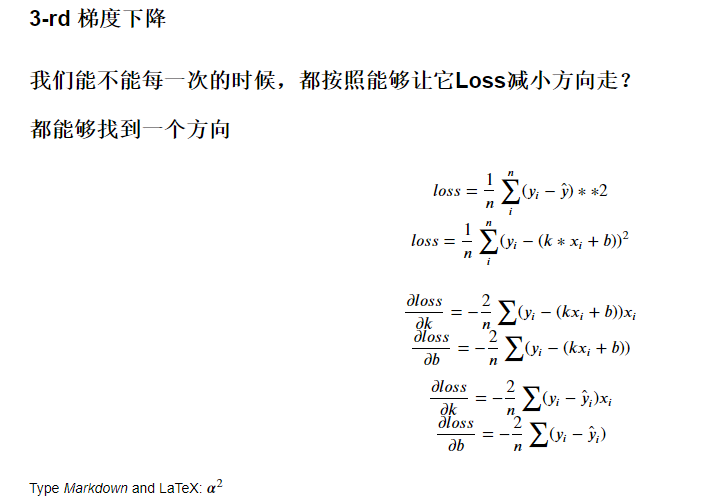
>>> zip(a,c) # 元素个数与最短的列表一致

[(1, 4), (2, 5), (3, 6)]

>>> zip(\*zipped) # 与 zip 相反，\*zipped 可理解为解压，返回二维矩阵式

[(1, 2, 3), (4, 5, 6)]

3. 梯度下降 我们能不能每一次的时候，都按照能够让它Loss减小方向走？ 都能够找到一个方向



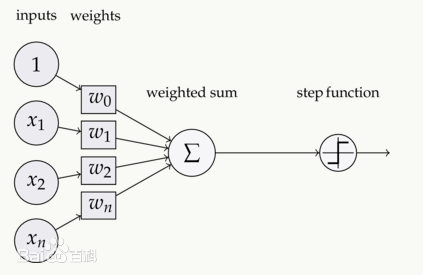
linear + sigmoid

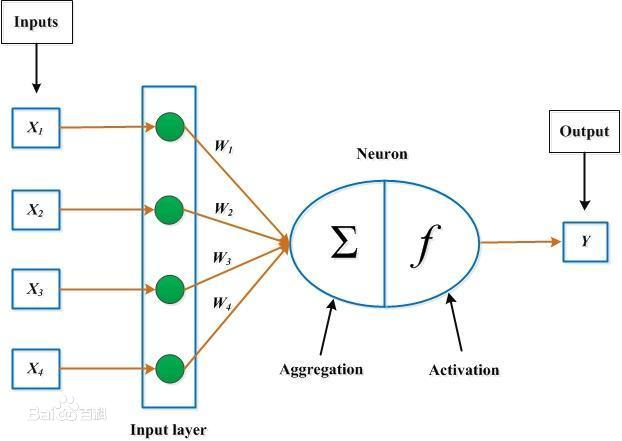
4. 世界中的真实的关系大多都不是简简单单的线性关系 我们能不能构建一些基本的模块，然后用模块来组合成复杂的函数?理论上 所有的函数 都可以用多层的线性函数 + 非线性变化 来进行拟合

激活函数

激活函数定义

将非线性特性引入到我们的网络中。如图1，在神经元中，输入的 inputs 通过加权，求和后，还被作用了一个函数，这个函数就是激活函数。引入激活函数是为了增 加神经网络模型的非线性。没有激活函数的每层都相当于矩阵相乘。就算你叠加了若干层之后，无非还是个矩阵相乘罢了。



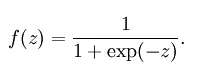


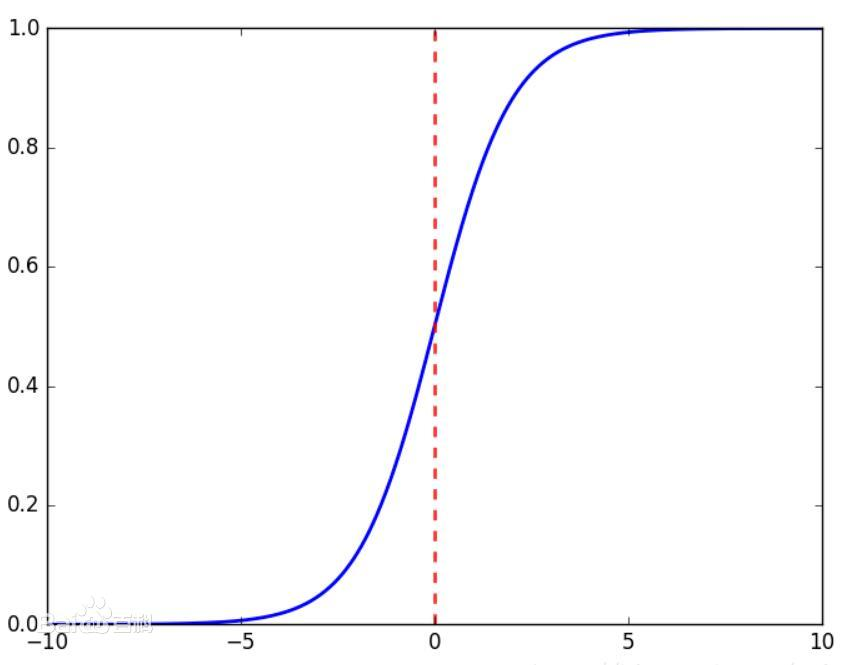
为什么要用激活函数

为什么要用激活函数编辑 如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合，这种情况就是最原始的感知机（Perceptron）。 如果使用的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。

常用的激活函数

Sigmoid函数

graphic 



也叫 Logistic 函数，用于隐层神经元输出

取值范围为(0,1)

它可以将一个实数映射到(0,1)的区间，可以用来做二分类。

在特征相差比较复杂或是相差不是特别大时效果比较好。

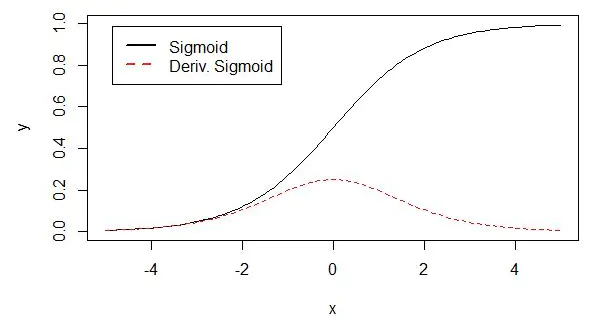
sigmoid缺点：

激活函数计算量大，反向传播求误差梯度时，求导涉及除法

反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练

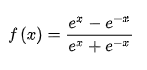
下面解释为何会出现梯度消失：

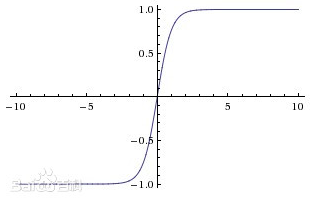
反向传播算法中，要对激活函数求导，sigmoid 的导数表达式为：



Sigmoid函数是一个在生物学中常见的S型函数，也称为S型生长曲线。在信息科学中，由于其单增以及反函数单增等性质，Sigmoid函数常被用作神经网络的阈值函数，将变量映射到0,1之间 [2] 。公式如下

Tanh函数

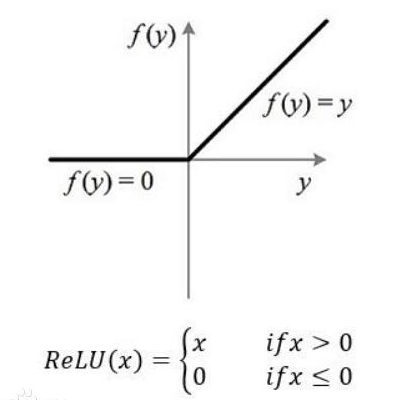




Tanh是双曲函数中的一个，Tanh()为双曲正切。在数学中，双曲正切“Tanh”是由基本双曲函数双曲正弦和双曲余弦推导而来

ReLU函数

graphic



输入信号 <0 时，输出都是0，>0 的情况下，输出等于输入

ReLU 的优点：

Krizhevsky et al. 发现使用 ReLU 得到的 SGD 的收敛速度会比 sigmoid/tanh 快很多

ReLU 的缺点：

训练的时候很”脆弱”，很容易就”die”了

例如，一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了，那么这个神经元的梯度就永远都会是 0.

如果 learning rate 很大，那么很有可能网络中的 40% 的神经元都”dead”了。

作者：不会停的蜗牛

链接：https://www.jianshu.com/p/22d9720dbf1a

来源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

优点：1.相比起Sigmoid和tanh，ReLU在SGD中能够快速收敛。

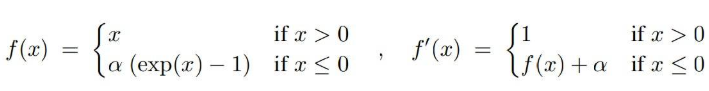
2.Sigmoid和tanh涉及了很多很expensive的操作（比如指数），ReLU可以更加简单的实现。3.有效缓解了梯度消失的问题。4.在没有无监督预训练的时候也能有较好的表现。

5.提供了神经网络的稀疏表达能力。

缺点：随着训练的进行，可能会出现神经元死亡，权重无法更新的情况。如果发生这种情况，那么流经神经元的梯度从这一点开始将永远是0。也就是说，ReLU神经元在训练中不可逆地死亡了。

LReLU、PReLU与RReLU

通常在LReLU和PReLU中，我们定义一个激活函数为：



LReLU当ai比较小而且固定的时候，我们称之为LReLU。LReLU最初的目的是为了避免梯度消失。但在一些实验中，我们发现LReLU对准确率并没有太大的影响。很多时候，当我们想要应用LReLU时，我们必须要非常小心谨慎地重复训练，选取出合适的a，LReLU的表现出的结果才比ReLU好。因此有人提出了一种自适应地从数据中学习参数的PReLU。

PReLUPReLU是LReLU的改进，可以自适应地从数据中学习参数。PReLU具有收敛速度快、错误率低的特点。PReLU可以用于反向传播的训练，可以与其他层同时优化。

Relu激活函数（The Rectified Linear Unit），用于隐层神经元输出

如何选择

选择的时候，就是根据各个函数的优缺点来配置，例如： 如果使用 ReLU，要小心设置 learning rate，注意不要让网络出现很多 “dead” 神经元，如果不好解决，可以试试 Leaky ReLU、PReLU 或者 Maxout.

* + 1. Linear Regression线性回归

因变量是连续的，自变量可以是连续的也可以是离散的，回归线的性质是线性的。 线性回归使用最佳的拟合直线（也就是回归线）在因变量（Y）和一个或多个自变量（X）之间建立一种关系。 用一个方程式来表示它，即Y=a+b\*X + e ，其中a表示截距，b表示直线的斜率，e是误差项。这个方程可以根据给定的预测变量（s）来预测目标变量的值。

R-square指标来评估模型性能

* + 1. Logistic Regression逻辑回归

逻辑回归是用来计算“事件=Success”和“事件=Failure”的概率。当因变量的类型属于二元（1 / 0，真/假，是/否）变量时，我们就应该使用逻辑回归。这里，Y的值从0到1，它可以用下方程表示。

逻辑回归不要求自变量和因变量是线性关系。它可以处理各种类型的关系，因为它对预测的相对风险指数OR使用了一个非线性的log转换。 为了避免过拟合和欠拟合，我们应该包括所有重要的变量。有一个很好的方法来确保这种情况，就是使用逐步筛选方法来估计逻辑回归。 它需要大的样本量，因为在样本数量较少的情况下，极大似然估计的效果比普通的最小二乘法差。 自变量不应该相互关联的，即不具有多重共线性。然而，在分析和建模中，我们可以选择包含分类变量相互作用的影响。 如果因变量的值是定序变量，则称它为序逻辑回归。 如果因变量是多类的话，则称它为多元逻辑回归。

* + 1. others

Polynomial Regression多项式回归

Stepwise Regression逐步回归

Ridge Regression岭回归

Lasso Regression套索回归

ElasticNet回归

1. Action
   1. 针对MarketBasket数据集进行购物篮分析（频繁项集及关联规则挖掘）
      1. Rule1: efficient\_apriori

efficient\_apriori\_BI CW2.py

See document(s): [efficient\_apriori\_BI CW2.py](efficient_apriori_BI%20CW2.py)

* + 1. Rule2 mixtend.frequent\_patterns

mixtend.frequent\_patterns\_BI CW2.py

See document(s): [mixtend.frequent\_patterns\_BI CW2.py](mixtend.frequent_patterns_BI%20CW2.py)