**目 录**

**[一、 有监督学习算法](#_Toc9609)** [2](#_Toc9609)

[1. KNN 算法 2](#_Toc2421)

[2. 推荐系统：协同过滤算法 6](#_Toc31560)

[3. 线性回归 9](#_Toc23547)

[4. 逻辑回归算法 9](#_Toc10896)

[5. 朴素贝叶斯算法 11](#_Toc29492)

[朴素贝叶斯应用 --- 垃圾邮件过滤 12](#_Toc10445)

[6. 决策树 --- 随机森林 -- Adaboost算法 14](#_Toc4961)

[解决3：集成算法 --- 随机森林。 15](#_Toc17336)

[解决4：集成算法---AdaBoost算法 15](#_Toc27626)

[7. SVM（Support Vector Machine, SVM） 17](#_Toc27494)

**[二、 无监督学习](#_Toc22960)** [28](#_Toc22960)

[1. K-Means算法（数据自动划分为固定（手动传值）的类） 28](#_Toc2502)

[2. DBSCAN 30](#_Toc1303)

**[三、 数据挖掘与数据分析，的相关算法：](#_Toc15132)** [32](#_Toc15132)

**[四、 关联性分析算法（Apriori，FP-growth）](#_Toc279)** [32](#_Toc279)

[1. Apriori 算法（先验算法） --- 关联性分析算法 32](#_Toc5942)

[2. FP-growth（Frequent Pattern）算法 --- 关联性分析算法 36](#_Toc31765)

**[五、 降维算法](#_Toc31091)** [46](#_Toc31091)

[1. PCA（主成分分析）算法 --- 降维算法 对矩阵做分解 46](#_Toc22617)

[2. SVD （奇异值分解）算法 --- 降维算法 对矩阵做分解 48](#_Toc26067)

**[六、正则化](#_Toc29016)** [50](#_Toc29016)

[1. L1 绝对值的差距 50](#_Toc23456)

[2. L2 平方差和 50](#_Toc15406)

**[七、其他-问题：](#_Toc10884)** [51](#_Toc10884)

[1. 样本包括特征和目标变量 51](#_Toc9461)

[2. 为什么要做机器学习 51](#_Toc17655)

[3. 梯度下降 --- 批梯度下降 --- 随机梯度下降 --- 小批的梯度下降 51](#_Toc22502)

[4. 导数消亡 51](#_Toc9065)

[5. 过拟合 欠拟合 51](#_Toc23649)

[6. 反向传播 52](#_Toc9635)

[7. SVM的前身，感知机 53](#_Toc22863)

[7.1 对二分类问题的具体描述 53](#_Toc3110)

[7.2 代价函数 53](#_Toc21883)

[7.3 算法的流程 53](#_Toc12697)

[8. 激活函数的作用 53](#_Toc7129)

[9. 激活函数的种类 54](#_Toc11391)

**机器学习**

**机器学习，**分为**有监督学习**，**非监督学习**，**半监督学习（强化学习）**

**（1）监督学习：**

有明确的目标变量

目标变量：类别

**（2）非监督学习：**

没有明确的目标变量

**（3）半监督学习 强化学习**：

根据奖惩机制来进行学习

**用机器学习解决实际问题的开发思路：**

1. 对数据进行处理 确定特征矩阵
2. 构建特征矩阵，构建目标向量
3. 选择模型，进行训练
4. 如果得分不高，优化模型

优化手段：增加数据，换模型，调超参数，检查代码逻辑

1. **有监督学习算法**

**包括KNN算法（K近邻法k-nearest neighbors，KNN），线性回归，逻辑回归，朴素贝叶斯，决策树，随机森林，集成算法---adaboost，SVM。**

1. **KNN 算法**

近朱者赤，近墨者黑，谁离我近就是谁

**KNN 工作原理**

1. 假设有一个带有标签的样本数据集（训练样本集），其中包含每条数据与所属分类的对应关系。

2. 输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较。

1. 计算新数据与样本数据集中每条数据的距离。

2. 对求得的所有距离进行排序（从小到大，越小表示越相似）。

3. 取前 k （k 一般小于等于 20 ）个样本数据对应的分类标签。

4. 求 k 个数据中出现次数最多的分类标签作为新数据的分类

**Knn算法实现流程**

1.加载数据，并构建对应矩阵

2.**归一化数据，消除特征影响差距**

3.计算样本距离，预测样本分类

4.用测试数据测试算法准确性

**KNN算法特点**

优点：精度高、对异常值不敏感、无数据输入假定

缺点：计算复杂度高、空间复杂度高

适用数据范围：数值型和标称型

**KNN算法代码实现1：**  
from collections import Counter  
# 实现knn的思路：  
# 1. 加载数据（数据集，新样本）  
def load(filepath):  
 X = []  
 y = []  
 with open(filepath,'r') as r:  
 for line in r.readlines():  
 datas = line.strip().split('\t')  
 X.append(datas[0:3])  
 y.append(datas[-1])  
 return X,y  
# 2. 对数据进行归一化处理  
def guiyi(X):  
 X = X[1:]  
 # print(X)  
 newX = []  
 l1 = []  
 l2 = []  
 l3 = []  
 # 找到每一列的最大值，每一个元素除以对应列的最大值  
 for i in X:  
 # 拆分每一列  
 l1.append(int(i[0]))  
 l2.append(float(i[1]))  
 l3.append(float(i[2]))  
 # 找到每一列的最大值  
 l1\_max = max(l1)  
 l2\_max = max(l2)  
 l3\_max = max(l3)  
 # 重新组合归一化后的列表  
 for i in range(len(l1)):  
 newX.append([l1[i]/l1\_max,l2[i]/l2\_max,l3[i]/l3\_max])  
 return newX  
# # 3.计算新样本到每个样本的距离  
# 求测试数据到训练数据的最小距离，并返回最终预测返回值  
# # x -- 测试样本 [0.14924473726498474, 0.23176661627632156, 0.4201726607605745]  
# # X -- 训练样本  
def knn\_y(x,newX,y,k):  
 dis = []  
 yy = []  
 for xx in newX:  
 dis\_each = ((x[0]-xx[0])\*\*2 + (x[1]-xx[1])\*\*2 + (x[2]-xx[2])\*\*2)\*\*0.5  
 dis.append(dis\_each)  
 # 4.取出前k个距离最小的样本  
 new\_dis = sorted(dis)[:k]  
 # 需要知道前k个最小样本，在原训练样本的下标  
 for i in [dis.index(d) for d in new\_dis]:  
 yy.append(y[i])  
 # 5.统计新样本的类别，将出现次数最多的类别作为新样本的类别  
 # # Counter类的目的：跟踪值出现的次数，它是一个无序的容器类型，以字典的键值对形式存储，其中元素为key，计数为value  
 # # .most\_common(1)[0][0] 返回前1个元素，[0]列表第一个元素 + [0]元组第一个元素取出具体值 例：[('a', 5), ('r', 2), ('b', 2)]  
 return Counter(yy).most\_common(1)[0][0]  
if \_\_name\_\_ == '\_\_main\_\_':  
 filepath = r'F:\z-百知教育正式学习\第五阶段\机器学习\资料\dating.txt'  
 x = [0.47940793005598586, 0.37680909668842943, 0.785864134656273]  
 X,y = load(filepath)  
 newX = guiyi(X)  
 k = 2  
 # knn\_y(x,newX,y,k)  
 print(knn\_y(x,newX,y,k))

**KNN算法代码实现2：**

机器学习的框架 Scikit-learn 练习  
# knn K最近邻(k-Nearest Neighbor，KNN)  
# 分类算法的核心思想：如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。  
# KNN算法可用于多分类，KNN算法不仅可以用于分类，还可以用于回归。  
# 通过找出一个样本的k个最近邻居，将这些邻居的属性的平均值赋给该样本，作为预测值  
import pandas as pd  
from sklearn import preprocessing  
# train\_test\_split分类器：训练---测试---拆分  
from sklearn.model\_selection import train\_test\_split  
# 加载数据  
# 训练数据  
from sklearn.neighbors import KNeighborsClassifier  
# usecols 决定使用文件中的哪些列 sep 指定列的分隔符  
X = pd.read\_table(r"F:\机器学习\资料\dating.txt",usecols=(0,1,2),sep='\t')  
Y = pd.read\_table(r"F:\机器学习\资料\dating.txt",usecols=(3,),sep='\t')  
# 测试数据  
test\_X = pd.read\_table(r"F:\机器学习\资料\dating\_test.txt",usecols=(0,1,2),sep='\t')  
test\_y = pd.read\_table(r"F:\机器学习\资料\dating\_test.txt",usecols=(3,),sep='\t')  
  
# 对数据做归一化处理,可以理解为数据归一化制定的一个规则  
# 预处理相关的类，最大值最小值归一化，默认归一化到0-1之间  
scaler = preprocessing.MinMaxScaler(feature\_range=(0,1))  
# 按照规则 训练并转化数据  
X = scaler.fit\_transform(X)  
test\_X = scaler.fit\_transform(test\_X)  
# 做分类  
# KNeighborsClassifier使用很简单，三步：  
# 1）创建KNeighborsClassifier对象，  
# 2）调用fit函数，  
# 3）调用predict函数进行预测  
knn = KNeighborsClassifier(n\_neighbors=3)  
# 将训练样本集中的数据拆分成两个样本集：一个训练 一个测试 train\_test\_split(X,Y,test\_size=0.2)  
# 参数1：准备拆分的数组 参数2：拆分的比例 （训练集和测试集分别占多少，默认比例为25% 一般20% 或者random\_state 随机比例 或者shuffle=True 打乱数据集）  
# 可以同时拆分多个数组，返回的是列表  
# for i in range(10):  
# train\_x,test\_x,train\_y,test\_y=train\_test\_split(X,Y,test\_size=0.2)  
# 训练样本数据，knn中就会有一个模型，再输入测试数据，根据训练好的模型，做预测  
knn.fit(X,Y) # 训练模型  
print(knn.predict(test\_X)) # 对新数据做预测  
  
# 计算测试集预测的准确率  
print(knn.score(test\_X,test\_y))

**KNN算法代码实现3：knn--numpy 版**  
# NumPy系统是python开发的用于科学计算的类库。  
# 这种工具可用来存储和处理大型矩阵，打破了全局锁 效率很高。  
# 比Python自身的嵌套列表（nested list structure)结构要高效的多（该结构也可以用来表示矩阵（matrix））  
  
# NumPy中最重要的封装好的对象，Ndarray对象 多维数组对象 二维数组  
from collections import Counter  
import pandas as pd  
import numpy as np  
# 1.加载数据（数据集，新样本）  
def loadData(filepath):  
 X = pd.read\_table(filepath,usecols=(0,1,2),sep='\t')  
 Y = pd.read\_table(filepath,usecols = (3,),sep='\t')  
 X\_nd = np.array(X)  
 Y\_nd = np.array(Y)  
 return X\_nd,Y\_nd  
# np.array(X).tolist()  
# 构建Ndarray对象 最常用的是numpy.array()  
# NumPy数组实际上被称为ndarray

# 2.对数据进行归一化处理  
def guiyi(X):  
 new\_X = X/X.max(axis = 0)  
 return new\_X  
def knn\_min(x,X,y,k):  
 # 3.计算新样本到每个样本的距离  
 dis = []  
 for xx in X:  
 dis.append((x-xx).dot((x-xx).T)\*\*0.5) # .dot() 求点积  
 # 4.取出前k个距离最小的样本  
 # new\_dis = sorted(dis,reverse=True)[0:k]  
 new\_dis = sorted(dis)[0:k]  
 yy = []  
 for i in [dis.index(distance) for distance in new\_dis]:  
 yy.append(tuple(y[i].tolist()))  
 # 5.统计新样本的类别，将出现次数最多的类别作为新样本的类别  
 return Counter(yy).most\_common(1)[0][0][0]  
if \_\_name\_\_ == '\_\_main\_\_':  
 filepath = r"F:\机器学习\资料\dating.txt"  
 X,Y = loadData(filepath)  
 new\_X = guiyi(X)  
 x = [0.47940793005598586, 0.37680909668842943, 0.785864134656273]  
 k = 2  
 print(knn\_min(x,new\_X,Y,k))

**2. 推荐系统：协同过滤算法**

**两种模型：**

1. **基于用户的推荐系统 朋友之间的推荐**
2. 给用户找相似用户 协同
3. 计算所有相似用户对所有未消费商品的平均得分
4. 过滤掉用户已经消费的商品 过滤
5. 取出得分最高的前2个商品（根据业务决定）
6. 将商品推荐给用户

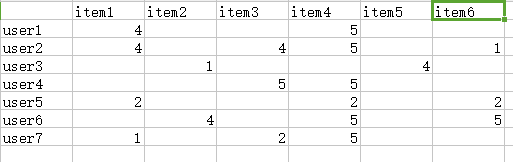
**实现思路：**

1. 加载数据 填充缺省值
2. 计算用户之间的距离 欧氏距离 余弦距离 杰卡德距离
3. 找k个距离最小的用户
4. 过滤掉用户消费过的商品
5. 计算所有相似用户对所有未消费的商品的平均值
6. 找到得分最高的前两个商品
7. **基于商品的推荐系统 相关物品之间的推荐**
8. 给商品找相似商品
9. 过滤已经消费该商品的用户
10. 计算所有相似商品的未消费该商品的用户的平均得分
11. 取出得分最高的前2个用户
12. 将用户推荐给商品

**推荐系统和机器学习的集成：**

**使用机器学习算法替换掉相似用户的找法**

**推荐系统代码实现：**

****

# 推荐系统  
# 给用户推商品  
# 加载excel表格 用pandas  
import pandas as pd  
import numpy as np  
  
# 加载数据，填充缺省值  
datas = pd.read\_excel(r"F:\z-百知教育正式学习\第五阶段\机器学习\资料\user-item.xlsx")  
datas = datas.fillna(0) # 所有NAN 都换成0  
items=datas.columns  
# user = datas.ix[0] # 取第一行  
# 也可以用切片来取第一行 （这是个二维列表）  
# user = datas[0:1]  
# 也可以将pandas对象直接扔到numpy中  
datas = np.array(datas)  
  
user = datas[0]  
# 通过计算欧式距离找相似用户 ---求其他用户与user的欧式距离，找到前k个最小的即最相似的用户  
###################################################################################################  
# dis\_user = []  
# for u in datas:  
# # if (u!=user).any():  
# # dis\_user.append(np.dot(user-u,(user-u).T)\*\*0.5)  
# dis\_user.append(np.dot(user-u,(user-u).T))  
# print(dis\_user)  
# users = [datas[dis\_user.index(d)] for d in sorted(dis\_user) if d!=0.0][0:3]  
# print(users)  
# # dis：[0.0, 17.0, 58.0, 41.0, 17.0, 57.0, 13.0]  
# # 结果：[array([1., 0., 2., 5., 0., 0.]), array([4., 0., 4., 5., 0., 1.]), array([4., 0., 4., 5., 0., 1.])]  
# # 这种写法的缺点：因为有两个相同的距离，所以会产生相同的用户  
# # 既然值一样，说明这两个用户很相似，但是 要推荐的商品却不一样，所以这种用index的方法 不合适  
###################################################################################################  
# 解决办法：将用户和距离用字典绑定起来  
dis\_user = {}  
for u in datas:  
 # if (u!=user).any():  
 # dis\_user.append(np.dot(user-u,(user-u).T)\*\*0.5) # .dot 求点积  
 dis\_user.update({tuple(u):np.dot(user-u,(user-u).T)})  
dis = sorted(set(dis\_user.values()))[0:3]  
# 相似用户 --- 根据最小距离找到所有相似用户  
users = []  
for key,value in dis\_user.items():  
 if value in dis:  
 users.append(key)  
print(users)  
# 拿到相似用户的矩阵  
users = np.array(users)  
# 根据评分找物品  
# 1. 求相似用户购买的所有物品评分的平均值 --- 计算所有商品的平均分  
scores=users.mean(axis=0).tolist()  
print(scores)  
new\_scores=users.mean(axis=0).tolist()  
  
# 过滤掉user用户消费过的商品  
for u in user:  
 if u != 0:  
 new\_scores.remove(scores[user.tolist().index(u)])  
tuijian\_items = [items[scores.index(i)] for i in sorted(new\_scores,reverse = True)[0:2]]  
print(tuijian\_items)

**3. 线性回归**

**解决的问题：**假设特征与目标变量之间的关系是线性的，**求出最合适的参数。**

**线性回归需要记住整个开发流程：**

第一步：提出假设函数（损失函数）。

第二步，找到一个合适的**损失函数**，将求假设--参数的过程，转变为**求损失函数最小值**的过程，损失函数是基于一个样本的。

第三步，找**成本函数**（代价函数），求整个样本集的损失值，求出来的值最小，说明调出来的值是最优的。

第四步，拿到代价函数，如何求解？如果简单的话，用最小二乘法，不简单的话，用梯**度下降法**，梯度下降包括三种（原生梯度下降，随机梯度下降，小批量梯度下降），这三种的区别是每一次计算的时候使用多少的数据集，原生的用整个样本集，随机梯度用一个样本，小批量用一部分。

第五步，将求出来的值带入到原来的**假设函数**中，这个假设就成立了。

第六步，带入之后，发现测试集的分数很差，需要优化模型，方法一，调梯度下降的学习率，调小一点，可能计算更准确。方法二，增加样本数量。方法三，换函数模型，可能这种线性函数模型并不适用于这个问题，换逻辑回归，或局部加权线性回归试一下。

**为什么线性回归的损失函数用平方的形式？**

答：在线性回归中，对于训练数据样本(xi,yi)(xi,yi)，我们有如下的拟合直线：yiˆ=θ\* xi+b

构建的损失函数是：  

表示每一个训练点到拟合直线的竖直距离的平方和，通过最小化上面的损失函数可以求得拟合直线的最佳参数θ。

这里的损失函数之所以使用平方形式，是使用了“最小二乘法”的思想，这里的“二乘”指的是用平方来度量观测点与估计点的距离（远近），“最小”指的是参数值要保证各个观测点与估计点的距离的平方和达到最小。

第二种解释是 极大似然估计误差的思想，暂时没搞明白，先保留。参考：http://blog.csdn.net/saltriver/article/details/57544704

**4. 逻辑回归算法**

在线性回归的外面套了一层Sigmoid函数,函数公式 y=1/(1+e^-x)。当x特别大的时候，y约等于1，当x特别小的时候，y约等于0。

使用线性回归，效果不是很好，因为y的预测有可能大于1或者小于0，这个时候，我们一般习惯在线性回归方程的外面再套一层sigmoid()函数来进行转换。sigmoid函数与y轴的交点是0.5，是一个平滑的曲线，x越大越接近1，x越小，越接近0 。

开发流程与线性回归一样，不过是换了一下损失函数和代价函数。

假设：y=1/(1+e^-(w·x+b)) 平滑的曲线 0-1之间

损失函数 y=-(ylogy^+(1-y)log(1-y^))

**损失函数的要求：** 损失函数必须满足它的最小值意味着预测值接近真实值（任意情况）

损失函数：用来评估算法性能的函数，作用于单个样本，可以用二分之一平方差，也可  
以使用别的函数。 式子的值越小越好，f(x)的值越小越好。

**这个式子为什么可以作为损失函数？**  
 当y=1的时候，y的预测值就应该尽可能的大，但是由于y的预测值，是由sigmoid函数计算得出，所以最大也不会超过1，因此应该是y的预测值越接近1越好。同样的，当y=0的时候，y的预测值，越接近0越好。

**代价函数：**用来评估算法性能的函数，作用于全部样本



用梯度下降求最合适的参数。

**注意：**使用sk-learn开发逻辑回归 有数学计算的都需要归一化

**官方Logistic回归算法工作原理：**

在每个特征上都乘以一个回归系数，然后把所有结果值相加，将这个总和代入 Sigmoid 函数中，进而得到一个范围在 0~1 之间的数值。任何大于 0.5 的数据被分入 1 类，小于 0.5 即被归入 0 类。所以， Logistic 回归也可以被看成是一种**概率估计**

**Logistic回归算法实现流程**

1.加载数据，并将数据类型转化为数值型

2.将每个回归系数初始化为 1 （**回归系数的作用？？？**）

3.计算整个数据集的梯度

4.使用步长 x 梯度更新回归系数的向量

5.3-4步重复R次

6.函数收敛之后返回回归系数

7.对测试数据基于训练完成的回归系数进行简单的回归计算，判定它们所属类别

**Logistic回归算法特点**

优点：计算代价不高，易于理解和实现。

缺点：容易欠拟合，分类精度可能不高

适用数据范围：数值型和标称型

**Logistic 逻辑回归算法的代码实现：**

from sklearn import preprocessing  
from sklearn.linear\_model import LogisticRegression # 逻辑回归  
# from sklearn.model\_selection import train\_test\_split  
# from sklearn.linear\_model import LinearRegression # 线性回归  
from sklearn.linear\_model import LogisticRegression  
# 加载数据 训练样本和测试样本  
from sklearn.model\_selection import train\_test\_split  
X = pd.read\_table(r'F:\机器学习\资料\dating.txt',sep='\t',usecols=range(0,2))  
Y = pd.read\_table(r'F:\机器学习\资料\dating.txt',sep='\t',usecols=(3,))  
test\_X = pd.read\_table(r'F:\机器学习\资料\dating\_test.txt',sep='\t',usecols=range(0,2))  
test\_Y = pd.read\_table(r'F:\机器学习\资料\dating\_test.txt',sep='\t',usecols=(3,))  
  
# 对数据做归一化处理（预处理preprocessing）  
scaler = preprocessing.MinMaxScaler(feature\_range=(0,1))  
# print(scaler)-------->MinMaxScaler(copy=True, feature\_range=(0, 1))  
# 按照规则训练并转化数据  
X = scaler.fit\_transform(X)  
test\_X = scaler.fit\_transform(test\_X)  
# 训练模型  
lr = LogisticRegression()

# lr = LinearRegression() # 如果是线性回归，使用LinearRegression( )  
for i in range(10):  
 train\_x, test\_x, train\_y, test\_y = train\_test\_split(X, Y, test\_size=0.2)  
 lr.fit(train\_x,train\_y)  
 # print(lr.predict(test\_X))  
 print(lr.score(test\_x,test\_y))

**5. 朴素贝叶斯算法**

条件独立公式，如果X和Y相互独立：

条件概率公式： 或者 

全概率公式：

贝叶斯公式：

hbw总结：**假设特征之间相互独立**，根据贝叶斯公式，计算在给定类别的情况下，出现某个特征的概率，谁大就是谁。

yy总结：假设特征之间相互独立，根据朴素贝叶斯公式，计算在某个特征下，分到一类别的概率，和分到二类别的概率是多少。在计算的时候，算在给定一类别的条件下，各个特征出现的概率是多少。

**本质：求概率，哪个概率大，就属于哪个类别。**

朴素是因为要特征相互独立。事实上，特征独立的情况特别少，我们只能人为的认为是相对独立的。在计算的时候，认为相互独立能够算出一个好的结果。

官方：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。

**朴素贝叶斯算法实现流程**

1）确定特征属性，获取训练样本

2）对每个类别分别计算概率P（y）

3）对每个特征属性计算所有划分的条件概率

4）对于每个类别计算概率p（x|y）p(y)

5）以上个步骤中计算出的最大值作为x的类别

**朴素贝叶斯的主要优点：**

1）朴素贝叶斯模型发源于古典数学理论，有稳定的分类效率。

2）对小规模的数据表现很好，在数据较少的情况下仍然有效，能够处理多分类任务，适合增量式训练，尤其是数据量超出内存时，我们可以一批批的去增量训练。

3）对缺失数据不太敏感，算法也比较简单，常用于文本分类。

**朴素贝叶斯的主要缺点：**

1） 理论上，朴素贝叶斯模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为朴素贝叶斯模型给定输出类别的情况下,假设属性之间相互独立，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。而在属性相关性较小时，朴素贝叶斯性能最为良好。对于这一点，有半朴素贝叶斯之类的算法通过考虑部分关联性适度改进。

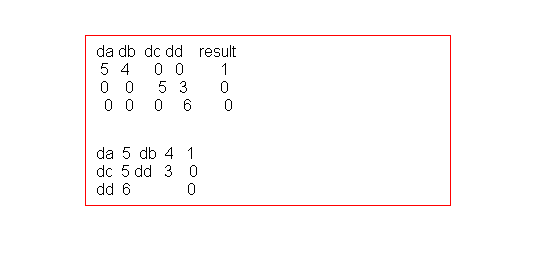
2）需要知道先验概率，且先验概率很多时候取决于假设，假设的模型可以有很多种，因此在某些时候会由于假设的先验模型的原因导致预测效果不佳。

3）由于我们是通过先验和数据来决定后验的概率从而决定分类，所以分类决策存在一定的错误率。

4）对输入数据的表达形式很敏感。

**朴素贝叶斯算法的适用数据范围：标称型。**

**朴素贝叶斯应用 --- 垃圾邮件过滤**

1. 正常邮件和垃圾邮件的内容 一个个的词 垃圾邮件和正常邮件的常用词是不一样的。
2. 
3. 选择部分在所有邮件中最常出现的词（前3000个）
4. 构建特征矩阵和目标向量
5. 选择模型进行训练

**过滤垃圾邮件的代码实现：**

# 过滤垃圾邮件 垃圾邮件的类别为 -1 正常邮件的类别为 1  
 # 1. 加载数据，分析数据  
 # 2. 遍历所有邮件，找出出现次数最多的常用词，比如找出3000  
 # 3. 将找出的3000个词构建，特征矩阵  
 # 4. 构建目标向量  
 # 5. 选择模型，进行训练  
 # 6. 测试，随便一封邮件，测试看是否准确  
import os  
import numpy as np  
from collections import Counter  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.naive\_bayes import BernoulliNB  
  
tingyong=['is','are'] #停用词  
def get\_word\_from\_files(filepath):  
 # 1. 获取所有训练样本集文件  
 l = [] # 所有文件所有词  
 words = [] # 一个大列表里面，包含n个小列表，每一个小列表中包含一个文件（邮件）里面的所有词  
 for filename in os.listdir(filepath):  
 # 2. 读取每一个文件，获取常用词  
 w = []  
 with open(os.path.join(filepath,filename)) as r:  
 for word in r.read().split(' '):  
 if word.isalnum() and word not in tingyong and len(word)>=2:  
 l.append(word)  
 w.append(word)  
 words.append(w)  
 # 获取常用词  
 # 如果列表里面是一个个元组，那么可以直接在外面套一层字典，转成字典  
 # print(dict(Counter(l).most\_common(3000)))  
 # list(dict(Counter(l).most\_common(3000))) 即 一个列表，包含字典中所有的键，即所有常用词  
 often\_words = list(dict(Counter(l).most\_common(3000)))  
 return often\_words,words  
def mk\_feature\_matrix(often\_words,words):  
 # 先构建一个全为0的特征矩阵,行为文件个数，列为常用词个数 3000  
 X = np.zeros((len(words),len(often\_words)))  
 # 统计每个常用词在每个文件中出现的次数  
 for i, each\_file in enumerate(words):  
 for j,each\_often\_word in enumerate(often\_words):  
 if each\_often\_word in each\_file:  
 X[i][j]= each\_file.count(each\_often\_word)  
 return X

if \_\_name\_\_ == '\_\_main\_\_':  
 train\_filepath = r'F:\z-百知教育正式学习\第五阶段\机器学习\资料\email\train-mails'  
 test\_filepath = r'F:\z-百知教育正式学习\第五阶段\机器学习\资料\email\test-mails'  
 # 训练样本集  
 often\_words,words = get\_word\_from\_files(train\_filepath)  
 X = mk\_feature\_matrix(often\_words,words)  
 # Y = np.zeros(len(words))  
 Y = np.zeros(702)  
 Y[0:351] = 1  
 # 测试样本集  
 test\_often\_words, test\_words = get\_word\_from\_files(test\_filepath)  
 test\_X = mk\_feature\_matrix(often\_words,test\_words)  
 test\_Y = np.zeros(260) # 构建一维向量  
 test\_Y[0:130] = 1  
  
 # 训练样本  
 # 创建模型对象  
 gaussianNB = GaussianNB()  
 gaussianNB.fit(X,Y)  
 print(gaussianNB.score(test\_X,test\_Y))

**6. 决策树 --- 随机森林 -- Adaboost算法**

**hbw总结**：计算系统的信息熵，找到最大信息增益对应的一组最佳特征解 根据新的特征组合构建一棵树，根据树做预测。

**yy总结**：

**决策树的构建是数据逐步分裂的过程，构建的步骤如下:**

步骤1：将所有的数据看成是一个节点，进入步骤2；

步骤2：从所有的数据特征中挑选一个数据特征对节点进行分割，进入步骤3；

步骤3：生成若干孩子节点，对每一个孩子节点进行判断，如果满足停止分裂的条件，进入；否则，进入步骤2；

步骤4：设置该节点是子节点，其输出的结果为该节点数量占比最大的类别。

**从上述步骤可以看出，决策生成过程中有三个重要的问题：**

https://www.cnblogs.com/yonghao/p/5061873.html

1. **数据如何分割：**

分裂属性的数据类型分为离散型和连续性两种情况，对于离散型的数据，按照属性值进行分裂，每个属性值对应一个分裂节点；对于连续性属性，一般性的做法是对数据按照该属性进行排序，再将数据分成若干区间，如[0,10]、[10,20]、[20,30]…，一个区间对应一个节点，若数据的属性值落入某一区间则该数据就属于其对应的节点。

1. **如何选择分裂的属性？**

决策树采用贪婪思想进行分裂，即选择可以得到最优分裂结果的属性进行分裂。最理想的情况是能找到一个属性刚好能够将不同类别分开，但是大多数情况下分裂很难一步到位，我们希望每一次分裂之后孩子节点的数据尽量”纯”。

选择分裂属性是要找出能够使所有孩子节点数据最纯的属性，决策树使用**信息增益或者信息增益率**作为选择属性的依据。

1. **什么时候停止分裂？**

决策树不可能不限制地生长，总有停止分裂的时候，最极端的情况是当节点分裂到只剩下一个数据点时自动结束分裂，但这种情况下树过于复杂，而且预测的精度不高。一般情况下为了降低决策树复杂度和提高预测的经度，会适当提前终止节点的分裂。

　　以下是决策树节点停止分裂的一般性条件：

　　①最小节点数

　　当节点的数据量小于一个指定的数量时，不继续分裂。两个原因：一是数据量较少时，再做分裂容易强化噪声数据的作用；二是降低树生长的复杂性。提前结束分裂一定程度上有利于降低过拟合的影响。

　　②熵或者基尼值小于阀值。

     由上述可知，熵和基尼值的大小表示数据的复杂程度，当熵或者基尼值过小时，表示数据的纯度比较大，如果熵或者基尼值小于一定程度数，节点停止分裂。

　　③决策树的深度达到指定的条件

　  节点的深度可以理解为节点与决策树跟节点的距离，如根节点的子节点的深度为1，因为这些节点与跟节点的距离为1，子节点的深度要比父节点的深度大1。决策树的深度是所有叶子节点的最大深度，当深度到达指定的上限大小时，停止分裂。

　　④所有特征已经使用完毕，不能继续进行分裂。

     被动式停止分裂的条件，当已经没有可分的属性时，直接将当前节点设置为叶子节点。

**决策树的构建方法：**

根据决策树的输出结果，决策树可以分为**分类树**和**回归树**，分类树输出的结果为具体的类别，而回归树输出的结果为一个确定的数值。

决策树的构建算法主要有**ID3、C4.5、基于Gini的CART**三种，其中ID3和C4.5是分类树，CART是分类回归树。其中ID3是决策树最基本的构建算法，而C4.5和CART是在ID3的基础上进行优化的算法。

**在给定特征的情况下，每一个特征谁做第一个节点，谁做第二个节点，是有影响的。**

**怎么来衡量这个影响？**

引入了信息熵的概念，用信息熵公式pi\*log(pi)，求信息熵，**对信息熵进行一个评估**。通过信息熵的评估，得出信息增益，**信息熵最小，信息增益最大**，这个最大的信息增益对应的特征即是最优特征，这种方式即**ID3**。

**问题：离散性很强的没有意义的那些特征，会影响到数据集。**

**解决1：**除了计算系统的信息熵意外，还要计算它自身的信息熵，根据公式，得到**信息增益率**。如果它的信息增益率比较大的话，说明它是一个合适的节点。拿到节点再去构建决策树，即**C4.5**算法

**解决2**：**CART算法**，换了信息熵的计算公式。

**解决3：**集成算法 --- **随机森林**。

**随机森林：多棵树并行做预测 将所有树出现的结果做投票选举（少数服从多数）。**

一棵树构建完成后，感觉不太好，可以使用集成算法。多棵树进行并行计算。得到多个结果，最终结果由投票选举获得。

**随机森林算法实现流程：**

1.采取有放回的**抽样方式**构造子数据集，保证不同子集之间的数量级一样（不同子集／同一子集 之间的元素可以重复）

2. 利用子数据集来构建子决策树，将这个数据放到每个子决策树中，每个子决策树输出一个结果

3. 然后统计子决策树的投票结果，得到最终的分类就是随机森林的输出结果

**随机森林算法特点**

1.优点：它能够处理很高维度（feature很多）的数据，并且不用做特征选择，在训练完后，它能够给出哪些feature比较重要，对于不平衡的数据集来说，它可以平衡误差。在创建随机森林的时候，模型泛化能力强，训练速度快，容易做成并行化方法

2.缺点：随机森林已经被证明在某些噪音较大的分类或回归问题上会过拟合，另外对于有不同取值的属性的数据，取值划分较多的属性会对随机森林产生更大的影响，所以随机森林在这种数据上产出的属性权值是不可信的

3.适用数据范围：数值型和标称型

**解决4：集成算法---AdaBoost算法**

多个训练集串行计算（即使用一堆弱训练器串行），开始的训练器会标注训练错误的数据，第一个训练错误的数据，在往后传的时候，通过加权的方式，强制后续训练集去关注这个数据，越往后传，正确率越高，从而提高整体的准确率。

**官方：Adaboost算法工作原理：**

用反复修改的数据（主要是修正数据的权重）来训练一系列的弱学习器(一个弱学习器模型仅仅比随机猜测好一点, 比如一个简单的决策树)，由这些弱学习器的预测结果通过加权投票(或加权求和)的方式组合, 得到我们最终的预测结果。

**Adaboost算法实现流程**

1. 准备数据

2. 用准备的数据训练一个弱训练器

3. 将训练完的结果中错误的样本权重加大，正确的样本减小

4. 重复上述过程，直到一系列弱训练器全部使用完毕

5. 将每个训练器加权（注：此时权重根据弱训练器的错误率进行计算）求和得出最终结果。

**Adaboost算法特点：**

1.优点：泛化（由具体的、个别的扩大为一般的）错误率低，易编码，可以应用在大部分分类器上，无参数调节。

2.缺点：对离群点敏感。

3.适用数据范围：数值型和标称型

**官方：决策树算法工作原理**

决策树是通过一系列规则对数据进行分类的过程。它提供一种在什么条件下会得到什么值的类似规则的方法。决策树分为**分类树**和**回归树**两种，分类树对离散变量做决策树，回归树对连续变量做决策树。

**决策树算法实现流程**

1. 加载数据集并构建对应矩阵

2. 熵的计算

3. 根据最佳分割feature进行数据分割

4. 根据最大信息增益选择最佳分割feature

5. 递归构建决策树

6. 样本分类

**决策树算法特点**

1. 优点：计算复杂度不高，输出结果易于理解，对中间值缺失不敏感，可以处理不相关的特征数据。

2. 缺点：可能会产生过度匹配的问题

3. 适用数据范围：数值型和标称型

**决策树应用案例：<https://www.cnblogs.com/baiboy/p/ml3.html#_label2>**

1. **SVM（Support Vector Machine, SVM）**

**SVM** 应用**核技巧** 解决**非线性数据分类**的方法**，属于大间隔分类器。**

支持向量：决定了分割线的位置的那些样本点

机：算法

向量的范数：向量的长度 向量中所有元素的平方和再开根号 表示||X||

向量的内积：两个向量相乘=一个向量的长度乘以另外一个向量投影到它的长度  
 超平面：分隔线或者分割面

常见的核函数：

三种常见核函数：

（1）多项式核函数：,a,b,r是常数

（2）RBF径向基核函数：也叫高斯核函数，因为长得像

（3）sigmoid核函数：

一般来讲RBF核已经很好用了，它可以投射到无穷维（回顾一下exex的tylor展开式）

你可能还会有疑问，都什么函数能当核函数。答案是，所有**半正定矩阵**。

**Hbw总结：**利用核函数将低维度不可分的数据转化为高维度可分，找到一个最佳的分割位置（最大间隔）（超平面）。

**yy总结**：

随着样本量m的增加，SVM模型的计算复杂度会呈m2或m3增加。

流程，**求大间隔的好处**，使容错性更高，鲁棒性更强。

**如何求间隔，怎么知道哪条线最好**？

通过求向量内积，以及损失函数做对比。

对线性不可分的情况，利用核函数，将低维不可分的转化为高维可分的。这时候，就可以做到，在任意情况下，不管数据集合样本集是什么样子的，都可以通过核函数来计算。

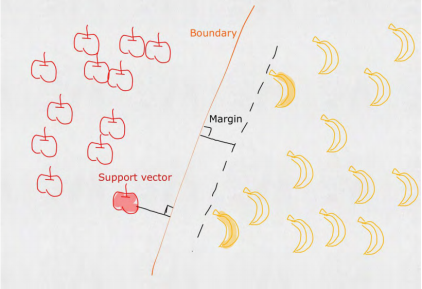
参考网址：https://www.cnblogs.com/Belter/p/8975606.html

**---------------------**

**官方：SVM支持向量机算法工作原理：**

1.寻找最大分类间距

2.通过拉格朗日函数求优化的问题



**SVM支持向量机算法实现流程：**

收集数据：可以使用任意方法。

准备数据：需要数值型数据。

分析数据：有助于可视化分隔超平面。

训练算法：SVM的大部分时间都源自训练，该过程主要实现两个参数的调优。

测试算法：十分简单的计算过程就可以实现。

使用算法：几乎所有分类问题都可以使用SVM，值得一提的是，SVM本身是一个二类分类器，对多类问题应用SVM需要对代码做一些修改

**对于线性可分的情况，SVM效果明显**

**而对于非线性的情况，**SVM此时需要用到一种叫核函数(kernel)的工具，将数据映射到高维空间，从而达到线性可分。

**SVM支持向量机算法特点：**

优点：泛化（由具体的、个别的扩大为一般的，就是说：模型训练完后的新样本）错误率低，计算开销不大，结果易理解。

缺点：对参数调节和核函数的选择敏感，原始分类器不加修改仅适合于处理二分类问题。

使用数据类型：数值型和标称型数据

**SVM的代码实现：<https://blog.csdn.net/csqazwsxedc/article/details/71513197>**

**详看SVM的推导过程文档**

from numpy import \*

def loadDataSet(filename): #读取数据

dataMat=[]

labelMat=[]

fr=open(filename)

for line in fr.readlines():

lineArr=line.strip().split('\t')

dataMat.append([float(lineArr[0]),float(lineArr[1])])

labelMat.append(float(lineArr[2]))

return dataMat,labelMat #返回数据特征和数据类别

def selectJrand(i,m): #在0-m中随机选择一个不是i的整数

j=i

while (j==i):

j=int(random.uniform(0,m))

return j

def clipAlpha(aj,H,L): #保证a在L和H范围内（L <= a <= H）

if aj>H:

aj=H

if L>aj:

aj=L

return aj

def kernelTrans(X, A, kTup): #核函数，输入参数,X:支持向量的特征树；A：某一行特征数据；kTup：('lin',k1)核函数的类型和参数

m,n = shape(X)

K = mat(zeros((m,1)))

if kTup[0]=='lin': #线性函数

K = X \* A.T

elif kTup[0]=='rbf': # 径向基函数(radial bias function)

for j in range(m):

deltaRow = X[j,:] - A

K[j] = deltaRow\*deltaRow.T

K = exp(K/(-1\*kTup[1]\*\*2)) #返回生成的结果

else:

raise NameError('Houston We Have a Problem -- That Kernel is not recognized')

return K

#定义类，方便存储数据

class optStruct:

def \_\_init\_\_(self,dataMatIn, classLabels, C, toler, kTup): # 存储各类参数

self.X = dataMatIn #数据特征

self.labelMat = classLabels #数据类别

self.C = C #软间隔参数C，参数越大，非线性拟合能力越强

self.tol = toler #停止阀值

self.m = shape(dataMatIn)[0] #数据行数

self.alphas = mat(zeros((self.m,1)))

self.b = 0 #初始设为0

self.eCache = mat(zeros((self.m,2))) #缓存

self.K = mat(zeros((self.m,self.m))) #核函数的计算结果

for i in range(self.m):

self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

def calcEk(oS, k): #计算Ek（参考《统计学习方法》p127公式7.105）

fXk = float(multiply(oS.alphas,oS.labelMat).T\*oS.K[:,k] + oS.b)

Ek = fXk - float(oS.labelMat[k])

return Ek

#随机选取aj，并返回其E值

def selectJ(i, oS, Ei):

maxK = -1

maxDeltaE = 0

Ej = 0

oS.eCache[i] = [1,Ei]

validEcacheList = nonzero(oS.eCache[:,0].A)[0] #返回矩阵中的非零位置的行数

if (len(validEcacheList)) > 1:

for k in validEcacheList:

if k == i:

continue

Ek = calcEk(oS, k)

deltaE = abs(Ei - Ek)

if (deltaE > maxDeltaE): #返回步长最大的aj

maxK = k

maxDeltaE = deltaE

Ej = Ek

return maxK, Ej

else:

j = selectJrand(i, oS.m)

Ej = calcEk(oS, j)

return j, Ej

def updateEk(oS, k): #更新os数据

Ek = calcEk(oS, k)

oS.eCache[k] = [1,Ek]

#首先检验ai是否满足KKT条件，如果不满足，随机选择aj进行优化，更新ai,aj,b值

def innerL(i, oS): #输入参数i和所有参数数据

Ei = calcEk(oS, i) #计算E值

if ((oS.labelMat[i]\*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i]\*Ei > oS.tol) and (oS.alphas[i] > 0)): #检验这行数据是否符合KKT条件 参考《统计学习方法》p128公式7.111-113

j,Ej = selectJ(i, oS, Ei) #随机选取aj，并返回其E值

alphaIold = oS.alphas[i].copy()

alphaJold = oS.alphas[j].copy()

if (oS.labelMat[i] != oS.labelMat[j]): #以下代码的公式参考《统计学习方法》p126

L = max(0, oS.alphas[j] - oS.alphas[i])

H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])

else:

L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)

H = min(oS.C, oS.alphas[j] + oS.alphas[i])

if L==H:

print("L==H")

return 0

eta = 2.0 \* oS.K[i,j] - oS.K[i,i] - oS.K[j,j] #参考《统计学习方法》p127公式7.107

if eta >= 0:

print("eta>=0")

return 0

oS.alphas[j] -= oS.labelMat[j]\*(Ei - Ej)/eta #参考《统计学习方法》p127公式7.106

oS.alphas[j] = clipAlpha(oS.alphas[j],H,L) #参考《统计学习方法》p127公式7.108

updateEk(oS, j)

if (abs(oS.alphas[j] - alphaJold) < oS.tol): #alpha变化大小阀值（自己设定）

print("j not moving enough")

return 0

oS.alphas[i] += oS.labelMat[j]\*oS.labelMat[i]\*(alphaJold - oS.alphas[j])#参考《统计学习方法》p127公式7.109

updateEk(oS, i) #更新数据

#以下求解b的过程，参考《统计学习方法》p129公式7.114-7.116

b1 = oS.b - Ei- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.K[i,i] -

oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.K[i,j]

b2 = oS.b - Ej- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.K[i,j]-

oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.K[j,j]

if (0 < oS.alphas[i]<oS.C):

oS.b = b1

elif (0 < oS.alphas[j]<oS.C):

oS.b = b2

else:

oS.b = (b1 + b2)/2.0

return 1

else:

return 0

#SMO函数，用于快速求解出alpha

def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)): #输入参数：数据特征，数据类别，参数C，阀值toler，最大迭代次数，核函数（默认线性核）

oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler, kTup)

iter = 0

entireSet = True

alphaPairsChanged = 0

while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):

alphaPairsChanged = 0

if entireSet:

for i in range(oS.m): #遍历所有数据

alphaPairsChanged += innerL(i,oS)

print("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged)) #显示第多少次迭代，那行特征数据使alpha发生了改变，这次改变了多少次alpha

iter += 1

else:

nonBoundIs = nonzero((oS.alphas.A > 0) \* (oS.alphas.A < C))[0]

for i in nonBoundIs: #遍历非边界的数据

alphaPairsChanged += innerL(i,oS)

print("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))

iter += 1

if entireSet:

entireSet = False

elif (alphaPairsChanged == 0):

entireSet = True

print("iteration number: %d" % iter)

return oS.b,oS.alphas

def testRbf(data\_train,data\_test):

dataArr,labelArr = loadDataSet(data\_train) #读取训练数据

b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', 1.3)) #通过SMO算法得到b和alpha

datMat=mat(dataArr)

labelMat = mat(labelArr).transpose()

svInd=nonzero(alphas)[0] #选取不为0数据的行数（也就是支持向量）

sVs=datMat[svInd] #支持向量的特征数据

labelSV = labelMat[svInd] #支持向量的类别（1或-1）

print("there are %d Support Vectors" % shape(sVs)[0]) #打印出共有多少的支持向量

m,n = shape(datMat) #训练数据的行列数

errorCount = 0

for i in range(m):

kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', 1.3)) #将支持向量转化为核函数

predict=kernelEval.T \* multiply(labelSV,alphas[svInd]) + b #这一行的预测结果（代码来源于《统计学习方法》p133里面最后用于预测的公式）注意最后确定的分离平面只有那些支持向量决定。

if sign(predict)!=sign(labelArr[i]): #sign函数 -1 if x < 0, 0 if x==0, 1 if x > 0

errorCount += 1

print("the training error rate is: %f" % (float(errorCount)/m)) #打印出错误率

dataArr\_test,labelArr\_test = loadDataSet(data\_test) #读取测试数据

errorCount\_test = 0

datMat\_test=mat(dataArr\_test)

labelMat = mat(labelArr\_test).transpose()

m,n = shape(datMat\_test)

for i in range(m): #在测试数据上检验错误率

kernelEval = kernelTrans(sVs,datMat\_test[i,:],('rbf', 1.3))

predict=kernelEval.T \* multiply(labelSV,alphas[svInd]) + b

if sign(predict)!=sign(labelArr\_test[i]):

errorCount\_test += 1

print("the test error rate is: %f" % (float(errorCount\_test)/m))

#主程序

def main():

filename\_traindata='C:\\Users\\Administrator\\Desktop\\data\\traindata.txt'

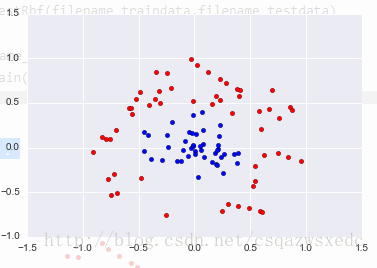
filename\_testdata='C:\\Users\\Administrator\\Desktop\\data\\testdata.txt'

testRbf(filename\_traindata,filename\_testdata)

if \_\_name\_\_=='\_\_main\_\_':

main()

样例数据如下：



训练数据：train\_data

-0.214824 0.662756 -1.000000

-0.061569 -0.091875 1.000000

0.406933 0.648055 -1.000000

0.223650 0.130142 1.000000

0.231317 0.766906 -1.000000

-0.748800 -0.531637 -1.000000

-0.557789 0.375797 -1.000000

0.207123 -0.019463 1.000000

0.286462 0.719470 -1.000000

0.195300 -0.179039 1.000000

-0.152696 -0.153030 1.000000

0.384471 0.653336 -1.000000

-0.117280 -0.153217 1.000000

-0.238076 0.000583 1.000000

-0.413576 0.145681 1.000000

0.490767 -0.680029 -1.000000

0.199894 -0.199381 1.000000

-0.356048 0.537960 -1.000000

-0.392868 -0.125261 1.000000

0.353588 -0.070617 1.000000

0.020984 0.925720 -1.000000

-0.475167 -0.346247 -1.000000

0.074952 0.042783 1.000000

0.394164 -0.058217 1.000000

0.663418 0.436525 -1.000000

0.402158 0.577744 -1.000000

-0.449349 -0.038074 1.000000

0.619080 -0.088188 -1.000000

0.268066 -0.071621 1.000000

-0.015165 0.359326 1.000000

0.539368 -0.374972 -1.000000

-0.319153 0.629673 -1.000000

0.694424 0.641180 -1.000000

0.079522 0.193198 1.000000

0.253289 -0.285861 1.000000

-0.035558 -0.010086 1.000000

-0.403483 0.474466 -1.000000

-0.034312 0.995685 -1.000000

-0.590657 0.438051 -1.000000

-0.098871 -0.023953 1.000000

-0.250001 0.141621 1.000000

-0.012998 0.525985 -1.000000

0.153738 0.491531 -1.000000

0.388215 -0.656567 -1.000000

0.049008 0.013499 1.000000

0.068286 0.392741 1.000000

0.747800 -0.066630 -1.000000

0.004621 -0.042932 1.000000

-0.701600 0.190983 -1.000000

0.055413 -0.024380 1.000000

0.035398 -0.333682 1.000000

0.211795 0.024689 1.000000

-0.045677 0.172907 1.000000

0.595222 0.209570 -1.000000

0.229465 0.250409 1.000000

-0.089293 0.068198 1.000000

0.384300 -0.176570 1.000000

0.834912 -0.110321 -1.000000

-0.307768 0.503038 -1.000000

-0.777063 -0.348066 -1.000000

0.017390 0.152441 1.000000

-0.293382 -0.139778 1.000000

-0.203272 0.286855 1.000000

0.957812 -0.152444 -1.000000

0.004609 -0.070617 1.000000

-0.755431 0.096711 -1.000000

-0.526487 0.547282 -1.000000

-0.246873 0.833713 -1.000000

0.185639 -0.066162 1.000000

0.851934 0.456603 -1.000000

-0.827912 0.117122 -1.000000

0.233512 -0.106274 1.000000

0.583671 -0.709033 -1.000000

-0.487023 0.625140 -1.000000

-0.448939 0.176725 1.000000

0.155907 -0.166371 1.000000

0.334204 0.381237 -1.000000

0.081536 -0.106212 1.000000

0.227222 0.527437 -1.000000

0.759290 0.330720 -1.000000

0.204177 -0.023516 1.000000

0.577939 0.403784 -1.000000

-0.568534 0.442948 -1.000000

-0.011520 0.021165 1.000000

0.875720 0.422476 -1.000000

0.297885 -0.632874 -1.000000

-0.015821 0.031226 1.000000

0.541359 -0.205969 -1.000000

-0.689946 -0.508674 -1.000000

-0.343049 0.841653 -1.000000

0.523902 -0.436156 -1.000000

0.249281 -0.711840 -1.000000

0.193449 0.574598 -1.000000

-0.257542 -0.753885 -1.000000

-0.021605 0.158080 1.000000

0.601559 -0.727041 -1.000000

-0.791603 0.095651 -1.000000

-0.908298 -0.053376 -1.000000

0.122020 0.850966 -1.000000

-0.725568 -0.292022 -1.000000

---------------------

测试数据：test\_data

0.676771 -0.486687 -1.000000

0.008473 0.186070 1.000000

-0.727789 0.594062 -1.000000

0.112367 0.287852 1.000000

0.383633 -0.038068 1.000000

-0.927138 -0.032633 -1.000000

-0.842803 -0.423115 -1.000000

-0.003677 -0.367338 1.000000

0.443211 -0.698469 -1.000000

-0.473835 0.005233 1.000000

0.616741 0.590841 -1.000000

0.557463 -0.373461 -1.000000

-0.498535 -0.223231 -1.000000

-0.246744 0.276413 1.000000

-0.761980 -0.244188 -1.000000

0.641594 -0.479861 -1.000000

-0.659140 0.529830 -1.000000

-0.054873 -0.238900 1.000000

-0.089644 -0.244683 1.000000

-0.431576 -0.481538 -1.000000

-0.099535 0.728679 -1.000000

-0.188428 0.156443 1.000000

0.267051 0.318101 1.000000

0.222114 -0.528887 -1.000000

0.030369 0.113317 1.000000

0.392321 0.026089 1.000000

0.298871 -0.915427 -1.000000

-0.034581 -0.133887 1.000000

0.405956 0.206980 1.000000

0.144902 -0.605762 -1.000000

0.274362 -0.401338 1.000000

0.397998 -0.780144 -1.000000

0.037863 0.155137 1.000000

-0.010363 -0.004170 1.000000

0.506519 0.486619 -1.000000

0.000082 -0.020625 1.000000

0.057761 -0.155140 1.000000

0.027748 -0.553763 -1.000000

-0.413363 -0.746830 -1.000000

0.081500 -0.014264 1.000000

0.047137 -0.491271 1.000000

-0.267459 0.024770 1.000000

-0.148288 -0.532471 -1.000000

-0.225559 -0.201622 1.000000

0.772360 -0.518986 -1.000000

-0.440670 0.688739 -1.000000

0.329064 -0.095349 1.000000

0.970170 -0.010671 -1.000000

-0.689447 -0.318722 -1.000000

-0.465493 -0.227468 -1.000000

-0.049370 0.405711 1.000000

-0.166117 0.274807 1.000000

0.054483 0.012643 1.000000

0.021389 0.076125 1.000000

-0.104404 -0.914042 -1.000000

0.294487 0.440886 -1.000000

0.107915 -0.493703 -1.000000

0.076311 0.438860 1.000000

0.370593 -0.728737 -1.000000

0.409890 0.306851 -1.000000

0.285445 0.474399 -1.000000

-0.870134 -0.161685 -1.000000

-0.654144 -0.675129 -1.000000

0.285278 -0.767310 -1.000000

0.049548 -0.000907 1.000000

0.030014 -0.093265 1.000000

-0.128859 0.278865 1.000000

0.307463 0.085667 1.000000

0.023440 0.298638 1.000000

0.053920 0.235344 1.000000

0.059675 0.533339 -1.000000

0.817125 0.016536 -1.000000

-0.108771 0.477254 1.000000

-0.118106 0.017284 1.000000

0.288339 0.195457 1.000000

0.567309 -0.200203 -1.000000

-0.202446 0.409387 1.000000

-0.330769 -0.240797 1.000000

-0.422377 0.480683 -1.000000

-0.295269 0.326017 1.000000

0.261132 0.046478 1.000000

-0.492244 -0.319998 -1.000000

-0.384419 0.099170 1.000000

0.101882 -0.781145 -1.000000

0.234592 -0.383446 1.000000

-0.020478 -0.901833 -1.000000

0.328449 0.186633 1.000000

-0.150059 -0.409158 1.000000

-0.155876 -0.843413 -1.000000

-0.098134 -0.136786 1.000000

0.110575 -0.197205 1.000000

0.219021 0.054347 1.000000

0.030152 0.251682 1.000000

0.033447 -0.122824 1.000000

-0.686225 -0.020779 -1.000000

-0.911211 -0.262011 -1.000000

0.572557 0.377526 -1.000000

-0.073647 -0.519163 -1.000000

-0.281830 -0.797236 -1.000000

-0.555263 0.126232 -1.000000

---------------------

1. **无监督学习**

包括：K-means，DBSCAN(基于密度的聚类算法)。

**1. K-Means算法（**数据自动划分为固定（手动传值）的类**）**

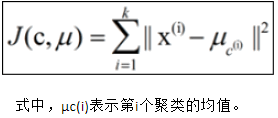
Hbw总结：随意k个质心 计算每个点到该点的距离，划分形成k个簇 将每个簇的平均值作为新的质心，迭代（停止条件，达到迭代次数，或者满足精度要求）

yy总结：

参考网址：http://www.cnblogs.com/ahu-lichang/p/7161613.html

**k-means算法的基础：**最小误差平方和准则。

**k-means算法的代价函数：**

****

各类簇内的样本越相似，其与该类均值间的误差平方越小，对所有类所得到的误差平方求和，即可验证分为k类时，各聚类是否是最优的。

上式的代价函数无法用解析的方法最小化，只能用迭代的方法。

**K-Means算法工作原理**

首先, 随机确定 K 个初始点作为质心（不是数据中的点），然后将数据集中的每个点分配到一个簇中, 具体来讲, 就是为每个点找到距其最近的质心, 并将其分配该质心所对应的簇. 这一步完成之后, 每个簇的质心更新为该簇所有点的平均值。

**K-Means算法实现流程：**

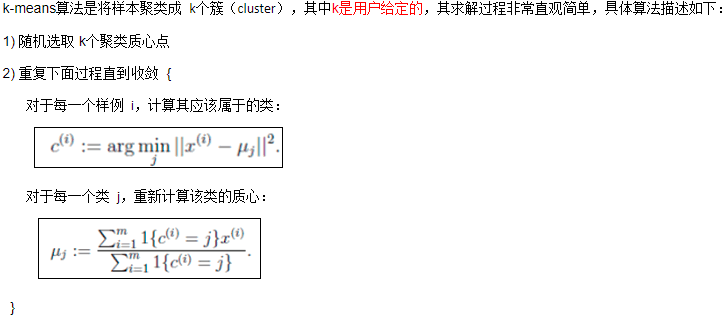
（1） 创建 k 个点作为起始质心（通常是随机选择）。

（2） 当任意一个点的簇分配结果发生改变时，

对于数据集中的每个数据点，计算每个质心与数据点之间的距离，

然后将数据点分配到距其最近的簇。

1. 对每一个簇，计算每个簇中所有点的均值，并将均值作为质心。



**K-Means算法特点：**

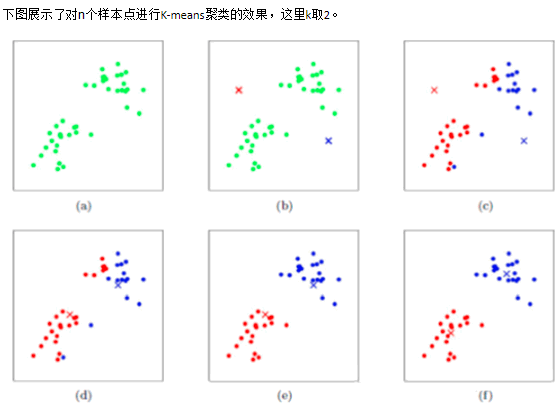
（1） 优点：容易实现

（2） 缺点：**可能收敛到局部最小值**, 在大规模数据集上收敛较慢

（3） 适用数据范围：数值型

**k个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。**

**K-Means算法步骤图解：**



**K-means算法的**缺点**及**改进方法：****

（1）**k值的选择是用户指定的，不同的k得到的结果会有挺大的不同**，如下图所示，左边是k=3的结果，这个就太稀疏了，蓝色的那个簇其实是可以再划分成两个簇的。而右图是k=5的结果，可以看到红色菱形和蓝色菱形这两个簇应该是可以合并成一个簇的：

**改进：**

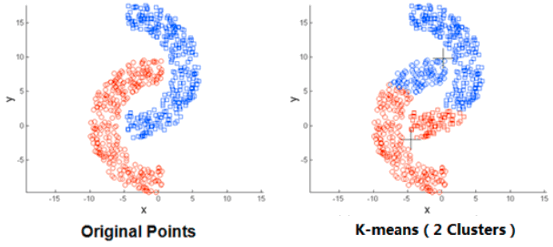
**对k的选择可以先用一些算法分析数据的分布，如重心和密度等，然后选择合适的k。**

（2）**对k个初始质心的选择比较敏感，容易陷入局部最小值**。例如，我们上面的算法运行的时候，有可能会得到不同的结果，如下面这两种情况。K-means也是收敛了，只是收敛到了局部最小值：

**改进：**

**有人提出了另一个成为二分k均值（bisecting k-means）算法，它对初始的k个质心的选择就不太敏感。**

1. **存在局限性，如下面这种非球状的数据分布就搞不定了**：



1. **数据集比较大的时候，收敛会比较慢。**
2. **DBSCAN（Density-Based Spatial Clustering of Applications with Noise,**具有噪声的基于密度的聚类方法**）**

**Hbw总结：**根据给定的eps邻域和最小点数来找核心点，据核心点找到对应的所有密度可达点 形成一个簇。

**yy总结：**

**DBSCAN（基于密度的聚类算法）工作原理：**

假定类别可以通过样本分布的紧密程度决定。同一类别的样本，他们之间是紧密相连的，也就是说，在该类别任意样本周围不远处一定有同类别的样本存在。通过将紧密相连的样本划为一类，这样就得到了一个聚类类别。通过将所有各组紧密相连的样本划为各个不同的类别，则我们就得到了最终的所有聚类类别结果。

**DBSCAN（基于密度）聚类算法实现流程：**

1. 导入数据样本D，初始化所有点为未访问，半径ε，和最少点数MinPts。

2. 建立neighbor队列。

3. 如果D中数据全部处理完，则算法结束，否则从D中选择一个未处理的点，标记为已访问，获得其所有**直接密度可达点**，如果为非核心点则标记为noise，重复步骤3，否则生成新的cluster，进入步骤4。

4. 将当前核心点放入该cluster，将该核心点的直接密度可达点放入neighbor队列，并遍历该队列，如果neighbor队列全部遍历完则回溯至步骤3。

5. 如果该点已经访问过，则进入步骤6，否则标记为已访问，然后获得该点的所有密度可达点，如果这个点也为核心点，则将该点的所有直接密度可达点放入neighbor队列。

6. 如果该点不属于任何cluster，则放入当前cluster。

7. 算法结束。

**DBSCAN（基于密度）聚类算法特点:**

1. **优点：**

1） 可以对任意形状的稠密数据集进行聚类，相对的，K-Means之类的聚类算法一般只适用于凸数据集。

2） 可以在聚类的同时发现异常点，对数据集中的异常点不敏感。

3） 聚类结果没有偏倚，相对的，K-Means之类的聚类算法初始值对聚类结果有很大影响。

2. **缺点：**

1）如果样本集的密度不均匀、聚类间距差相差很大时，聚类质量较差，这时用DBSCAN聚类一般不适合。

2）如果样本集较大时，聚类收敛时间较长，此时可以对搜索最近邻时建立的KD树或者球树进行规模限制来改进。

3）调参相对于传统的K-Means之类的聚类算法稍复杂，主要需要对距离阈值ϵ，邻域样本数阈值MinPts联合调参，不同的参数组合对最后的聚类效果有较大影响。

3. 适用数据范围：任意稠密性数据

**相关概念：**

MinPts：最少点数。

Eps邻域：给定对象半径Eps内的邻域称为该对象的Eps邻域。

核心对象：如果对象的Eps邻域至少包含最小数目MinPts的对象，则称该对象为核心对象。

边界点：边界点不是核心点，但落在某个核心点的邻域内。

噪音点：既不是核心点，也不是边界点的任何点。

直接密度可达：给定一个对象集合D，如果p在q的Eps邻域内，而q是一个核心对象，则称对象p 从对象q出发时是直接密度可达的(directly density-reachable)。

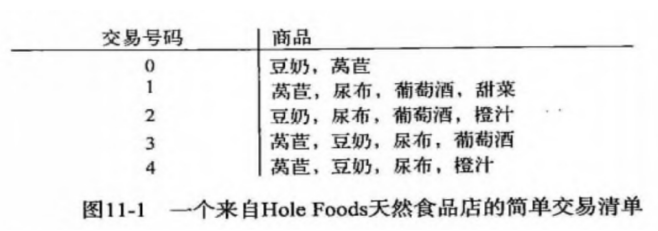
密度可达：对于点Q，如果存在p1,p2,...,pn且p1=p2,...,pn−1=pn,pn=Q，即p1到pn都是直接（密度）可达的，那么Q对于p1（密度）可达。

密度相连：如果存在对象O∈D，使对象p和q都是从O关于Eps和MinPts密度可达的，那么对象p到q是关于Eps和MinPts密度相连的(density-connected)。

1. **数据挖掘与数据分析，的相关算法：**

包括：降维算法（PCA，SVD），关联性分析算法（Apriori，fp-growth）

1. **关联性分析算法（Apriori，FP-growth）**



**关联分析：**是一种在大规模数据集中寻找有趣关系的任务。这些任务有两种形式：频繁项集和关联规则。

**频繁项集**（frequent item sets）: 经常出现在一起的物品的集合，上图中的 {葡萄酒, 尿布, 豆奶} 就是一个频繁项集的例子。

**关联规则**（associational rules）: 暗示两种物品之间可能存在很强的关系，上图中尿布 -> 葡萄酒就是一个关联规则。这意味着如果顾客买了尿布，那么他很可能会买葡萄酒。

**支持度**: 数据集中包含该项集的记录所占的比例。例如上图中，{豆奶} 的支持度为 4/5。{豆奶, 尿布} 的支持度为 3/5。

**可信度**: 针对一条诸如{尿布} -> {葡萄酒} 这样具体的关联规则来定义的。这条规则的可信度被定义为支持度({尿布, 葡萄酒})/支持度({尿布})，从图中可以看出支持度({尿布, 葡萄酒}) = 3/5，支持度({尿布}) = 4/5，所以{尿布} -> {葡萄酒} 的可信度 = 3/5 / 4/5 = 3/4 = 0.75。

可信度越高，关联规则越强。

**1. Apriori 算法（先验算法） --- 关联性分析算法**

**Apriori算法工作原理**

**Hbw总结：**

Apriori原理：一个频繁项集，它所有的子集也都是频繁项集

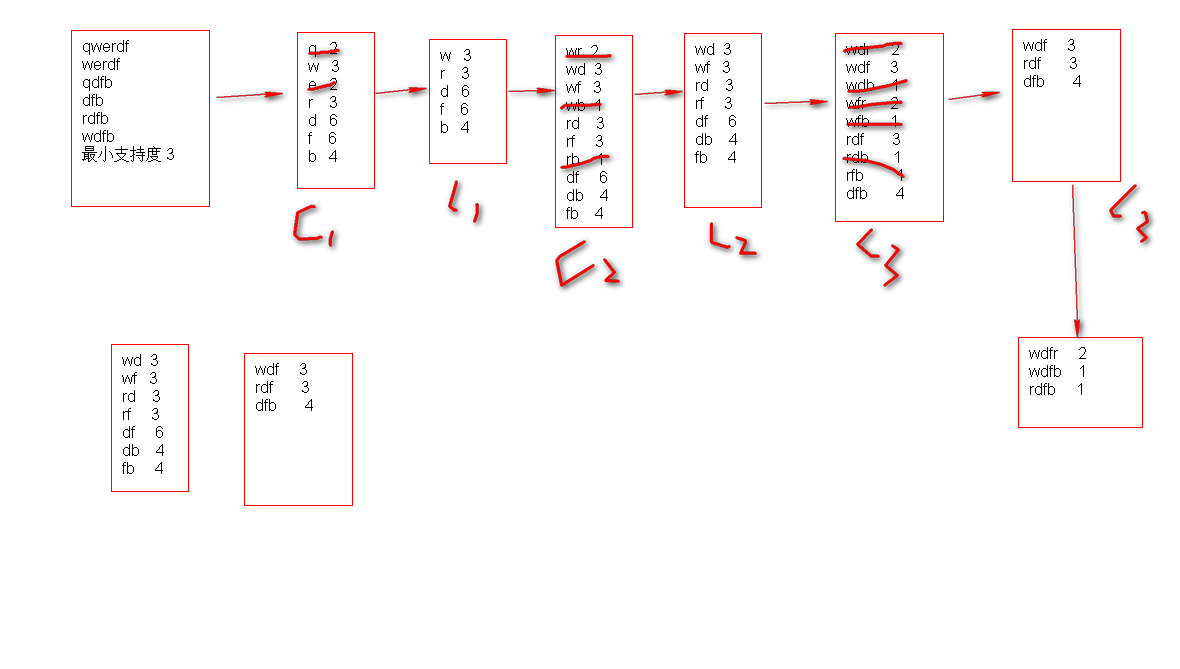
一个非频繁项集，其所有父类集合都是非频繁项集 剪枝

**yy总结：**

为了减少关联规则学习时所需的计算量，研究人员发现了一种所谓的 Apriori 原理，即某个项集是频繁的，那么它的所有子集也是频繁的。 例如，如果 {0, 1} 是频繁的，那么 {0}, {1} 也是频繁的。 该原理直观上没有什么帮助，但是如果反过来看就有用了，也就是说如果一个项集是 非频繁项集，那么它的所有超集（父集）也是非频繁项集。

Aprior算法中的两个术语：

（1）剪枝前：Cn （2）剪枝后：Ln



**Apriori 算法实现流程**

1. 导入数据

2. 生成所有单个数据的项集列表

3. 扫描数据集来查看哪些项集满足最小支持度要求，去掉那些不满足最小支持度要求的集合

4. 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

5. 重复3-4过程，直到所有项集被去掉

6. 利用相同思路去掉所有不满足最小可信度的集合

7. 利用剩余集合进行组合生成新的关联规则

最终得到符合你设置的最小支持度的有关联的集合。

**Apriori 算法特点**

1.优点：易编码实现

2.缺点：在大数据集上可能较慢

3.适用数据范围：数值型和标称型

**Apriori 算法的代码实现：**

def createC1(dataSet):  
 C1 = []  
 for transaction in dataSet:  
 for item in transaction:  
 if not [item] in C1:  
 C1.append([item]) #store all the item unrepeatly  
  
 C1.sort()  
 #return map(frozenset, C1)#frozen set, user can't change it.  
 return list(map(frozenset, C1))  
  
def scanD(D,Ck,minSupport):  
#参数：数据集、候选项集列表 Ck以及感兴趣项集的最小支持度 minSupport  
 ssCnt={}  
 for tid in D:#遍历数据集  
 for can in Ck:#遍历候选项  
 if can.issubset(tid):#判断候选项中是否含数据集的各项  
 #if not ssCnt.has\_key(can): # python3 can not support  
 if not can in ssCnt:  
 ssCnt[can]=1 #不含设为1  
 else: ssCnt[can]+=1#有则计数加1  
 numItems=float(len(D))#数据集大小  
 retList = []#L1初始化  
 supportData = {}#记录候选项中各个数据的支持度  
 for key in ssCnt:  
 support = ssCnt[key]/numItems#计算支持度  
 if support >= minSupport:  
 retList.insert(0,key)#满足条件加入L1中  
 supportData[key] = support  
 return retList, supportData  
  
def aprioriGen(Lk, k): #组合，向上合并  
 #creates Ck 参数：频繁项集列表 Lk 与项集元素个数 k  
 retList = []  
 lenLk = len(Lk)  
 for i in range(lenLk):  
 for j in range(i+1, lenLk): #两两组合遍历  
 L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]  
 L1.sort(); L2.sort()  
 if L1==L2: #若两个集合的前k-2个项相同时,则将两个集合合并  
 retList.append(Lk[i] | Lk[j]) #set union  
 return retList  
  
def apriori(dataSet, minSupport = 0.5):  
 C1 = createC1(dataSet)  
 D = list(map(set, dataSet)) #python3  
 L1, supportData = scanD(D, C1, minSupport)#单项最小支持度判断 0.5，生成L1  
 L = [L1]  
 k = 2  
 while (len(L[k-2]) > 0):#创建包含更大项集的更大列表,直到下一个大的项集为空  
 Ck = aprioriGen(L[k-2], k)#Ck  
 Lk, supK = scanD(D, Ck, minSupport)#get Lk  
 supportData.update(supK)  
 L.append(Lk)  
 k += 1  
 return L, supportData  
  
def generateRules(L, supportData, minConf=0.7):  
 #频繁项集列表、包含那些频繁项集支持数据的字典、最小可信度阈值  
 bigRuleList = [] #存储所有的关联规则  
 for i in range(1, len(L)): #只获取有两个或者更多集合的项目，从1,即第二个元素开始，L[0]是单个元素的  
 # 两个及以上的才可能有关联一说，单个元素的项集不存在关联问题  
 for freqSet in L[i]:  
 H1 = [frozenset([item]) for item in freqSet]  
 #该函数遍历L中的每一个频繁项集并对每个频繁项集创建只包含单个元素集合的列表H1  
 if (i > 1):  
 #如果频繁项集元素数目超过2,那么会考虑对它做进一步的合并  
 rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)  
 else:#第一层时，后件数为1  
 calcConf(freqSet, H1, supportData, bigRuleList, minConf)# 调用函数2  
 return bigRuleList  
  
def calcConf(freqSet, H, supportData, brl, minConf=0.7):  
 #针对项集中只有两个元素时，计算可信度  
 prunedH = []#返回一个满足最小可信度要求的规则列表  
 for conseq in H:#后件，遍历 H中的所有项集并计算它们的可信度值  
 conf = supportData[freqSet]/supportData[freqSet-conseq] #可信度计算，结合支持度数据  
 if conf >= minConf:  
 #print (freqSet-conseq,'-->',conseq,'conf:',conf)  
 #如果某条规则满足最小可信度值,那么将这些规则输出到屏幕显示  
 brl.append((freqSet-conseq, conseq, conf))#添加到规则里，brl 是前面通过检查的 bigRuleList  
 prunedH.append(conseq)#同样需要放入列表到后面检查  
 return prunedH  
  
def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):  
 #参数:一个是频繁项集,另一个是可以出现在规则右部的元素列表 H  
 m = len(H[0])  
 if (len(freqSet) > (m + 1)): #频繁项集元素数目大于单个集合的元素数  
 Hmp1 = aprioriGen(H, m+1)#存在不同顺序、元素相同的集合，合并具有相同部分的集合  
 Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)#计算可信度  
 if (len(Hmp1) > 1):  
 #满足最小可信度要求的规则列表多于1,则递归来判断是否可以进一步组合这些规则  
 rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)  
if \_\_name\_\_ == '\_\_main\_\_':  
 dataSet = [['豆奶', '莴苣'],  
 ['莴苣', '尿布', '葡萄酒', '甜菜'],  
 ['豆奶', '尿布', '葡萄酒', '橙汁'],  
 ['莴苣', '豆奶', '尿布', '葡萄酒'],  
 ['莴苣', '豆奶', '尿布', '葡萄酒','橙汁']]  
 #dataSet=[[1, 2, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]  
 L, suppData = apriori(dataSet, 0.5)  
 rules = generateRules(L, suppData, 0.7)  
 print(rules)

**2. FP-growth（Frequent Pattern）算法 --- 关联性分析算法**

**FP-growth算法简介：**

对于搜索引擎公司而言，他们需要通过查看互联网上的用词来找出经常在一块出现的词对，因此这些公司就需要能够高效的发现频繁项集的方法。

和Apriori算法相比，FP-growth算法只需要对数据库进行两次遍历（第一次是计算项的支持度，第二次是剪枝的时候需要遍历整个样本集），从而高效发现频繁项集。

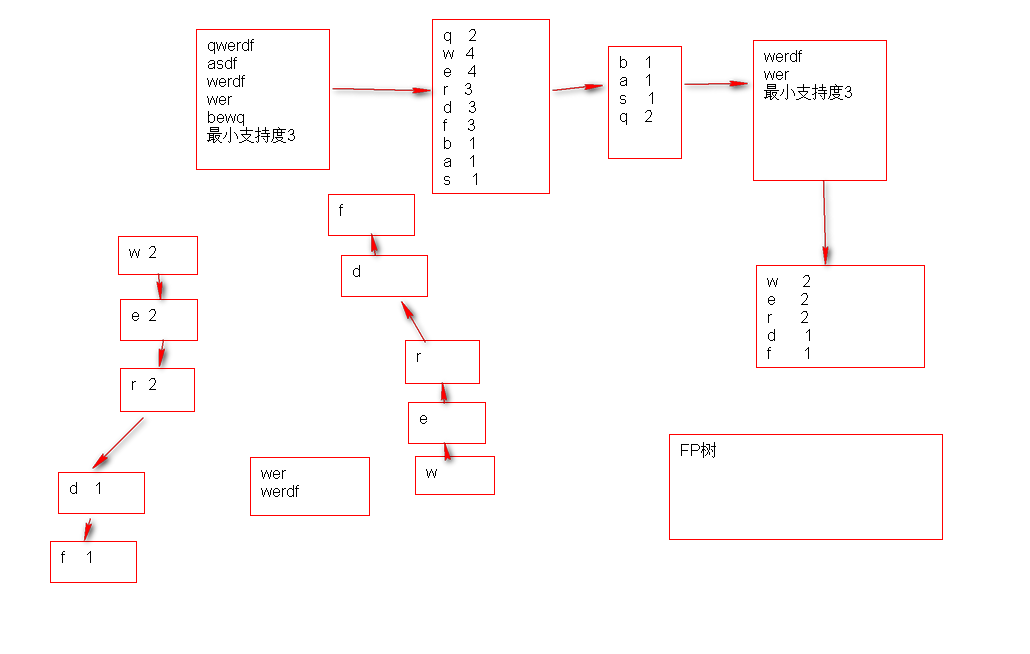
**FP-growth算法工作原理：**

基于Apriori算法构建，但是数据结构不同，通过将数据集存储在FP（Frequent Pattern)树上来发现频繁项集，但不能发现数据之间的关联规则。

FP-growth算法只需要对数据库进行两次扫描，而Apriori算法在求每个潜在的频繁项集时都需要扫描一次数据集，所以说Apriori算法是高效的。其中算法发现频繁项集的过程是：

(1)构建FP树；

(2)从FP树中挖掘频繁项集。



在剪枝的地方，和Aprior不同，计算所有项的支持度后，要在原样本集中，将所有包含非频繁项的样本丢弃（因为一个非频繁项集，它的父集也一定是非频繁项集）。再对剩下的样本集，做各个项的统计。利用满足最小支持度的项集（w，e，r，d，f）来构建一颗FP树（将支持度最小的放在上面，所有数据集合按照得到的顺序重新整理，通过遍历一棵树，来组合频繁项集）。

**FP-growth算法实现流程**

1. 准备数据

2. 遍历所有的数据集合，计算所有项的支持度

3. 丢弃非频繁的项

4. 基于支持度降序排序所有的项

5. 所有数据集合按照得到的顺序重新整理

6. 重新整理完成后，丢弃每个集合末尾非频繁的项

7. 读取每个集合插入FP树中，同时用一个头部链表数据结构维护不同集合的相同项

**FP-growth算法特点：**

1.优点：

（1）因为 FP-growth 算法只需要对数据集遍历两次，所以速度更快。

（2）FP树将集合按照支持度降序排序，不同路径如果有相同前缀路径共用存储空间，使得数据得到了压缩。（相同前缀的单词，存储少了）

（3）不需要生成候选集。

（4）比Apriori更快。

2. 缺点：

（1）FP-Tree第二次遍历会存储很多中间过程的值，会占用很多内存。

（2）构建FP-Tree是比较昂贵的（内存等）。

1. 适用数据范围：标称型数据(离散型数据)

**FP-growth算法代码：**

"""

A Python implementation of the FP-growth algorithm.

Basic usage of the module is very simple:

> from fp\_growth import find\_frequent\_itemsets

> find\_frequent\_itemsets(transactions, minimum\_support)

"""

"""

Note:

1、the original version only support py2.7. github url: "https://github.com/enaeseth/python-fp-growth"

2、This file is a updated version, which is support py3. github url: "https://github.com/Nana0606/python3-fp-growth"

"""

from collections import defaultdict, namedtuple

# original author information, this verison is updated by lina.

\_\_author\_\_ = 'Eric Naeseth <eric@naeseth.com>'

\_\_copyright\_\_ = 'Copyright © 2009 Eric Naeseth'

\_\_license\_\_ = 'MIT License'

def find\_frequent\_itemsets(transactions, minimum\_support, include\_support=False):

"""

Find frequent itemsets in the given transactions using FP-growth. This

function returns a generator instead of an eagerly-populated list of items.

The `transactions` parameter can be any iterable of iterables of items.

`minimum\_support` should be an integer specifying the minimum number of

occurrences of an itemset for it to be accepted.

Each item must be hashable (i.e., it must be valid as a member of a

dictionary or a set).

If `include\_support` is true, yield (itemset, support) pairs instead of

just the itemsets.

"""

items = defaultdict(lambda: 0) # mapping from items to their supports

# Load the passed-in transactions and count the support that individual

# items have.

for transaction in transactions:

for item in transaction:

items[item] += 1

# Remove infrequent items from the item support dictionary.

items = dict((item, support) for item, support in items.items()

if support >= minimum\_support)

# Build our FP-tree. Before any transactions can be added to the tree, they

# must be stripped of infrequent items and their surviving items must be

# sorted in decreasing order of frequency.

def clean\_transaction(transaction):

transaction = filter(lambda v: v in items, transaction)

transaction\_list = list(transaction) # 为了防止变量在其他部分调用，这里引入临时变量transaction\_list

transaction\_list.sort(key=lambda v: items[v], reverse=True)

return transaction\_list

master = FPTree()

for transaction in map(clean\_transaction, transactions):

master.add(transaction)

def find\_with\_suffix(tree, suffix):

for item, nodes in tree.items():

support = sum(n.count for n in nodes)

if support >= minimum\_support and item not in suffix:

# New winner!

found\_set = [item] + suffix

yield (found\_set, support) if include\_support else found\_set

# Build a conditional tree and recursively search for frequent

# itemsets within it.

cond\_tree = conditional\_tree\_from\_paths(tree.prefix\_paths(item))

for s in find\_with\_suffix(cond\_tree, found\_set):

yield s # pass along the good news to our caller

# Search for frequent itemsets, and yield the results we find.

for itemset in find\_with\_suffix(master, []):

yield itemset

class FPTree(object):

"""

An FP tree.

This object may only store transaction items that are hashable

(i.e., all items must be valid as dictionary keys or set members).

"""

Route = namedtuple('Route', 'head tail')

def \_\_init\_\_(self):

# The root node of the tree.

self.\_root = FPNode(self, None, None)

# A dictionary mapping items to the head and tail of a path of

# "neighbors" that will hit every node containing that item.

self.\_routes = {}

@property

def root(self):

"""The root node of the tree."""

return self.\_root

def add(self, transaction):

"""Add a transaction to the tree."""

point = self.\_root

for item in transaction:

next\_point = point.search(item)

if next\_point:

# There is already a node in this tree for the current

# transaction item; reuse it.

next\_point.increment()

else:

# Create a new point and add it as a child of the point we're

# currently looking at.

next\_point = FPNode(self, item)

point.add(next\_point)

# Update the route of nodes that contain this item to include

# our new node.

self.\_update\_route(next\_point)

point = next\_point

def \_update\_route(self, point):

"""Add the given node to the route through all nodes for its item."""

assert self is point.tree

try:

route = self.\_routes[point.item]

route[1].neighbor = point # route[1] is the tail

self.\_routes[point.item] = self.Route(route[0], point)

except KeyError:

# First node for this item; start a new route.

self.\_routes[point.item] = self.Route(point, point)

def items(self):

"""

Generate one 2-tuples for each item represented in the tree. The first

element of the tuple is the item itself, and the second element is a

generator that will yield the nodes in the tree that belong to the item.

"""

for item in self.\_routes.keys():

yield (item, self.nodes(item))

def nodes(self, item):

"""

Generate the sequence of nodes that contain the given item.

"""

try:

node = self.\_routes[item][0]

except KeyError:

return

while node:

yield node

node = node.neighbor

def prefix\_paths(self, item):

"""Generate the prefix paths that end with the given item."""

def collect\_path(node):

path = []

while node and not node.root:

path.append(node)

node = node.parent

path.reverse()

return path

return (collect\_path(node) for node in self.nodes(item))

def inspect(self):

print('Tree:')

self.root.inspect(1)

print

print('Routes:')

for item, nodes in self.items():

print(' %r' % item)

for node in nodes:

print(' %r' % node)

def conditional\_tree\_from\_paths(paths):

"""Build a conditional FP-tree from the given prefix paths."""

tree = FPTree()

condition\_item = None

items = set()

# Import the nodes in the paths into the new tree. Only the counts of the

# leaf notes matter; the remaining counts will be reconstructed from the

# leaf counts.

for path in paths:

if condition\_item is None:

condition\_item = path[-1].item

point = tree.root

for node in path:

next\_point = point.search(node.item)

if not next\_point:

# Add a new node to the tree.

items.add(node.item)

count = node.count if node.item == condition\_item else 0

next\_point = FPNode(tree, node.item, count)

point.add(next\_point)

tree.\_update\_route(next\_point)

point = next\_point

assert condition\_item is not None

# Calculate the counts of the non-leaf nodes.

for path in tree.prefix\_paths(condition\_item):

count = path[-1].count

for node in reversed(path[:-1]):

node.\_count += count

return tree

class FPNode(object):

"""A node in an FP tree."""

def \_\_init\_\_(self, tree, item, count=1):

self.\_tree = tree

self.\_item = item

self.\_count = count

self.\_parent = None

self.\_children = {}

self.\_neighbor = None

def add(self, child):

"""Add the given FPNode `child` as a child of this node."""

if not isinstance(child, FPNode):

raise TypeError("Can only add other FPNodes as children")

if not child.item in self.\_children:

self.\_children[child.item] = child

child.parent = self

def search(self, item):

"""

Check whether this node contains a child node for the given item.

If so, that node is returned; otherwise, `None` is returned.

"""

try:

return self.\_children[item]

except KeyError:

return None

def \_\_contains\_\_(self, item):

return item in self.\_children

@property

def tree(self):

"""The tree in which this node appears."""

return self.\_tree

@property

def item(self):

"""The item contained in this node."""

return self.\_item

@property

def count(self):

"""The count associated with this node's item."""

return self.\_count

def increment(self):

"""Increment the count associated with this node's item."""

if self.\_count is None:

raise ValueError("Root nodes have no associated count.")

self.\_count += 1

@property

def root(self):

"""True if this node is the root of a tree; false if otherwise."""

return self.\_item is None and self.\_count is None

@property

def leaf(self):

"""True if this node is a leaf in the tree; false if otherwise."""

return len(self.\_children) == 0

@property

def parent(self):

"""The node's parent"""

return self.\_parent

@parent.setter

def parent(self, value):

if value is not None and not isinstance(value, FPNode):

raise TypeError("A node must have an FPNode as a parent.")

if value and value.tree is not self.tree:

raise ValueError("Cannot have a parent from another tree.")

self.\_parent = value

@property

def neighbor(self):

"""

The node's neighbor; the one with the same value that is "to the right"

of it in the tree.

"""

return self.\_neighbor

@neighbor.setter

def neighbor(self, value):

if value is not None and not isinstance(value, FPNode):

raise TypeError("A node must have an FPNode as a neighbor.")

if value and value.tree is not self.tree:

raise ValueError("Cannot have a neighbor from another tree.")

self.\_neighbor = value

@property

def children(self):

"""The nodes that are children of this node."""

return tuple(self.\_children.itervalues())

def inspect(self, depth=0):

print((' ' \* depth) + repr(self))

for child in self.children:

child.inspect(depth + 1)

def \_\_repr\_\_(self):

if self.root:

return "<%s (root)>" % type(self).\_\_name\_\_

return "<%s %r (%r)>" % (type(self).\_\_name\_\_, self.item, self.count)

if \_\_name\_\_ == '\_\_main\_\_':

from optparse import OptionParser

import csv

p = OptionParser(usage='%prog data\_file')

p.add\_option('-s', '--minimum-support', dest='minsup', type='int',

help='Minimum itemset support (default: 2)')

p.add\_option('-n', '--numeric', dest='numeric', action='store\_true',

help='Convert the values in datasets to numerals (default: false)')

p.set\_defaults(minsup=2)

p.set\_defaults(numeric=False)

options, args = p.parse\_args()

if len(args) < 1:

p.error('must provide the path to a CSV file to read')

transactions = []

with open(args[0]) as database:

for row in csv.reader(database):

if options.numeric:

transaction = []

for item in row:

transaction.append(long(item))

transactions.append(transaction)

else:

transactions.append(row)

result = []

for itemset, support in find\_frequent\_itemsets(transactions, options.minsup, True):

result.append((itemset, support))

result = sorted(result, key=lambda i: i[0])

for itemset, support in result:

print(str(itemset) + ' ' + str(support))

1. **降维算法**

数据降维： 可能会有精度损失 主要是为了计算方便 有时候可以做特征选择。

**包括： PCA（主成分分析算法），SVD（奇异值分解 矩阵分解）**

1. **PCA（主成分分析）算法 --- 降维算法 对矩阵做分解**

**找到一个主成分，然后根据主成分做训练。**

**PCA（主成分分析）算法工作原理：**

PCA的思想是将n维特征映射到k维上（k<n），这k维是全新的正交特征。这k维特征称为**主成分**，是重新构造出来的k维特征，然后再重新分析数据。

假设A是一个方阵，如果存在一个实数r和一个向量m，满足A\*m=r\*m,把r叫做特征值，把m叫做特征向量。

方差： 注：X^是平均值，此时评估的是一组数据的方差，

方差即数据的偏离程度。

协方差：两组数据之间的偏离程度 X Y。 

**构建协方差矩阵**：（两两样本求出的协方差构建出的矩阵，即协方差矩阵）

求协方差矩阵的所有的特征值和特征向量

例：特征值 100 2 特征向量 [1,2,3] [4,5,6]

将特征值降序排列，找总量大于所有特征值的和的90%的那些特征值

根据那些特征值对应的特征向量构建一个新的特征矩阵（这个新的列数，一定小于原来）

如此达到了降维的手段。

（**PCA的思路：到底留哪些列，不留哪些列，通过计算协方差矩阵的特征值和特征向量，来选择保留最大的特征值和对应的特征向量**）

**PCA（主成分分析）算法实现流程：**

1. 找出第一个主成分的方向，也就是数据方差最大的方向。

2. 找出第二个主成分的方向，也就是数据方差次大的方向，并且该方向与第一个主成分方向正交(orthogonal 如果是二维空间就叫垂直)。

3. 通过这种方式**计算出所有的主成分方向**。

4. 通过数据集的协方差矩阵及其特征值分析，我们就可以得到这些主成分的值。

5. 一旦得到了协方差矩阵的特征值和特征向量，我们就可以保留最大的 N 个特征。这些特征向量也给出了 N 个最重要特征的真实结构，我们就可以通过将数据乘上这 N 个特征向量 从而将它转换到新的空间上。

**PCA（主成分分析）算法特点：**

1.优点：通过 PCA 进行降维处理，我们就可以同时获得 SVM 和决策树的优点：(得到了和决策树一样简单的分类器，同时分类间隔和SVM一样好) ，还可以降低数据的复杂性，识别最重要的多个特征

2.缺点：不一定需要，且可能损失有用信息。

3.适用数据范围：数值型

**PCA算法代码实现：**

from sklearn.decomposition import PCA

import pandas as pd

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

def datemap(datas):

# 1.要知道数据集有多少非数字列

for c in [c for c in datas.columns if type(datas[c][0])==str]:

# 2.要知道每一列中有多少不重复的值

index=0

for d in datas[c].drop\_duplicates():

# 3.把每一种值赋值为一个数字（每次变化）

datas[c][datas[c].str.strip()==d.strip()]=index

index+=1

return datas

if \_\_name\_\_ == '\_\_main\_\_':

filename=r'E:\AI137班\机器学习\资料\adult.txt'

test\_filename = r'E:\AI137班\机器学习\资料\adult\_test.txt'

X=pd.read\_table(filename,sep=',',usecols=range(14))

Y = pd.read\_table(filename, sep=',', usecols=(14,))

# test\_X = pd.read\_table(test\_filename, sep=',', usecols=range(14))

# test\_Y = pd.read\_table(test\_filename, sep=',', usecols=(14,))

X=datemap(X)

Y=datemap(Y)

pca=PCA(n\_components=7) # 参数n\_components指降到几维

X=pca.fit\_transform(X)

# print(X)

sc=MinMaxScaler()

X=sc.fit\_transform(X)

gbn = GaussianNB()

for i in range(10):

tarin\_X,test\_X,tarin\_Y,test\_Y=train\_test\_split(X,Y,test\_size=0.1)

# test\_X = datemap(test\_X)

# test\_Y = datemap(test\_Y)

# gbn=LinearRegression()

gbn.fit(tarin\_X.astype('float'),tarin\_Y.astype('int')) # 将y的值的类型换为int

print(gbn.score(test\_X.astype('float'),test\_Y.astype('int')))

**2. SVD （奇异值分解）算法 --- 降维算法 对矩阵做分解**

**SVD （奇异值分解）算法工作原理：**

通过对矩阵的分解来降维。

A=U\*M\*V

其中，A是一个n\*n的方阵 U是一个m\*k的矩阵 M是一个k\*k的对角矩阵 对角线上的元素是A的奇异值（奇异值即特征值的开方） V是一个k\*n的矩阵

找到奇异值总量占据90%的那些奇异值--》特征值--》特征向量--》特征矩阵

**SVD （奇异值分解）算法实现流程：**

1. 利用 SVD 从数据中构建一个主题空间

2. 在该空间下处理问题

**SVD （奇异值分解）算法特点：**

1.优点：简化数据，去除噪声，优化算法的结果。

2.缺点：数据的转换可能难以理解。

3.适用数据范围：数值型。

**SVD （奇异值分解）算法代码实现：**

from sklearn.decomposition import TruncatedSVD

import pandas as pd

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

def datemap(datas):

# 1.要知道数据集有多少非数字列

for c in [c for c in datas.columns if type(datas[c][0])==str]:

# 2.要知道每一列中有多少不重复的值

index=0

for d in datas[c].drop\_duplicates():

# 3.把每一种值赋值为一个数字（每次变化）

datas[c][datas[c].str.strip()==d.strip()]=index

index+=1

return datas

if \_\_name\_\_ == '\_\_main\_\_':

filename=r'E:\AI137班\机器学习\资料\adult.txt'

#test\_filename = r'E:\AI137班\机器学习\资料\adult\_test.txt'

X=pd.read\_table(filename,sep=',',usecols=range(14))

Y = pd.read\_table(filename, sep=',', usecols=(14,))

# test\_X = pd.read\_table(test\_filename, sep=',', usecols=range(14))

# test\_Y = pd.read\_table(test\_filename, sep=',', usecols=(14,))

X=datemap(X)

Y=datemap(Y)

svd=TruncatedSVD(n\_components=10)

X=svd.fit\_transform(X)

# print(X)

sc=MinMaxScaler()

X=sc.fit\_transform(X)

gbn = SVC()

for i in range(10):

tarin\_X,test\_X,tarin\_Y,test\_Y=train\_test\_split(X,Y,test\_size=0.1)

# test\_X = datemap(test\_X)

# test\_Y = datemap(test\_Y)

# gbn=LinearRegression()

gbn.fit(tarin\_X.astype('float'),tarin\_Y.astype('int')) # 将y的值的类型换为int

print(gbn.score(test\_X.astype('float'),test\_Y.astype('int')))

**六、正则化**

正则化 --- 用于克服过拟合问题。

正则化过程中通过添加一个L1（LASSO）或L2（Ridge）规范到权重向量w（通过给定算法学习到的参数）上以惩罚损失项：

 这里的λ是正则项，N(w)是L1 或L2 规范。

包括：L1 绝对值的差距，L2 平方差和

**1. L1 绝对值的差距**

L1正则化的损失函数fff是是不光滑的，在x=0处不可导，最优化理论告诉我们，函数的最优点在导数为0处或者非规则点（不可导的点）处取到。X=0有可能作为函数的最优点，即使X=0不是可导的。事实上，当λ足够大的时候（强正则影响），最优点确实会在X=0处取到。

**2. L2 平方差和**

L2正则化的损失函数ddd是光滑的，这意味着它的最优点是一个固定的点（在损失函数导数为0处），如果我们增大，那么这个点的函数值会变得很小。但它不会是0，除非f’(0)=0。

在高维情形下：如果一个特征是不重要的，那么它对于损失函数的影响就很小，即这个特征的权值即使很大，但对于损失函数的影响很小，但对于正则项的影响就很大了。此时在正则项的作用下，就会把这个特征给turn off掉（L1即为取0过滤掉，L2即为取一个很小的权重值）。

**七、其他-问题：**

1. 样本包括特征和目标变量
2. 为什么要做机器学习？

人类的学习过程：

**数据的积累+大脑的转化=得到对某件事物的认识或者提高了某项能力**

机器的学习过程？

**大数据+算法的转化=得到对某件事物的认识或者提高了某项能力（by hbw）**

为什么要做机器学习？

**因为机器学习可以解决一些传统编程无法解决的问题(by hbw)**

1. 梯度下降 --- 批梯度下降 --- 随机梯度下降 --- 小批的梯度下降

深度学习的优化算法，说白了就是梯度下降。每次的参数更新有两种方式。

第一种，遍历全部数据集算一次损失函数，然后算函数对各个参数的梯度，更新梯度。这种方法每更新一次参数都要把数据集里的所有样本都看一遍，计算量开销大，计算速度慢，不支持在线学习，这称为Batch gradient descent，批梯度下降。

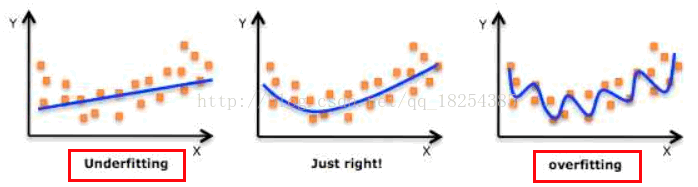
另一种，每看一个数据就算一下损失函数，然后求梯度更新参数，这个称为随机梯度下降，stochastic gradient descent。这个方法速度比较快，但是收敛性能不太好，可能在最优点附近晃来晃去，hit不到最优点。两次参数的更新也有可能互相抵消掉，造成目标函数震荡的比较剧烈。

为了克服两种方法的缺点，现在一般采用的是一种折中手段，mini-batch gradient decent，小批的梯度下降，这种方法把数据分为若干个批，按批来更新参数，这样，一个批中的一组数据共同决定了本次梯度的方向，下降起来就不容易跑偏，减少了随机性。另一方面因为批的样本数与整个数据集相比小了很多，计算量也不是很大。

1. **导数消亡(vanishing gradient):**

DAG: 有向非循环图形（directed acyclic graph, or DAG）  
导数消亡(vanishing gradient):   
 理论研究表明，在神经元总数一定的前提下，与传统的浅层神经网络（只有一个隐藏层）相比，深层神经网络具有更强大的学习能力[4]，能够自动提取出具有高层抽象含义的特征来解决复杂的机器学习问题。但深层神经网络的学习也有一些非常难以克服的难点。首先，神经网络训练的目标函数是非凸的，这意味着网络参数的训练并不能保证一定会收敛于全局最小点，而有可能在某个局部最小点停止。对于深层神经网络，传统的后向传播算法(back propagation)会遇到“导数消亡”(vanishing gradient)问题[5]，使得靠近输入端的隐藏层参数不能得到有效的训练，更容易陷入局部最小点，这导致深层神经网络的实际学习效果常常不如浅层神经网络，虽然它在理论上应该具有更强的学习能力。第二个难点是深层神经网络的学习会带来很大的计算量，学习的过程经常会持续数星期甚至数月。

1. 过拟合 欠拟合（https://blog.csdn.net/qq\_32742009/article/details/81629210）



（1）过拟合（over-fitting）：是所建的机器学习模型或者是深度学习模型在训练样本中表现得过于优越，导致在验证数据集以及测试数据集中表现不佳。

造成过拟合的本质原因是模型学习的太过精密，导致连训练集中的样本噪声也一丝不差的训练进入了模型。

（2）欠拟合（under-fitting），与过拟合恰好相反，模型学习的太过粗糙，连训练集中的样本数据特征关系（数据分布）都没有学出来。

**解决过拟合的方法主要有以下几种：**

（1）数据层面：

·数据集扩增（Data augmentation），获取更多的数据。

·特征工程，筛选组合得到更高质量的特征。

（2）模型层面：

·选择较为简单的模型

·集成学习，Bagging策略组合模型降低模型方差。

·加入正则项，如L1、L2正则项，以及树模型的剪枝策略，XGBoost中的正则项惩罚（叶子节点值+叶子节点个数）。

（3）更多方法：

·早停（Early stopping），在模型的训练精度已经到达一定的需求时停止训练，以防止模型学习过多的样本噪声。

·加入噪声，给定训练样本集更多的样本噪声，使得模型不易完全拟合这些噪声，从而只在大程度上的训练学习我们想要的数据特征关系。

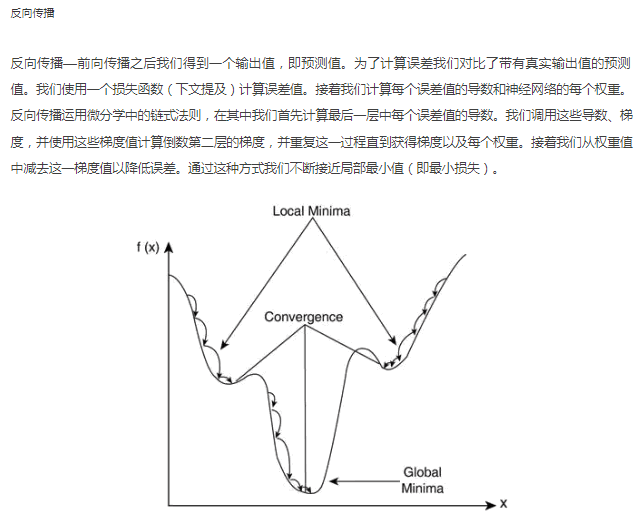
·dropout，在深度学习中，我们经常会使用dropout的方法来防止过拟合，dropout实际上借鉴来bagging的思想。

·正则化，常用的正则化方法就是加入**L1（Lasso）**与**L2（Ridge）**正则项。

·BN（Batch Normalization），BN每一次训练中所组成的Mini-Batch类似于Bagging策略，不同的Mini-Batch训练出来的BN参数也不同。

·权重衰减（Weight Deacy），有时我们也会称L2正则化为Weight Deacy，因为L2正则化会使得权重偏向于0.Weight Deacy实际上是使得模型在训练后期，权重的变化变得很慢很慢，从而使得模型不至于在迭代后期转而去学习更多的样本噪声。常用的权重衰减方法有滑动平均（Moving Average）

1. 反向传播

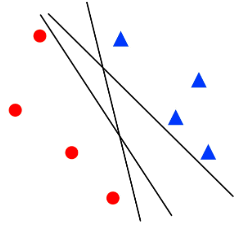


1. SVM的前身，感知机

感知机可以看做是低配版的线性SVM，从数学上可以证明：

在线性可分的两类数据中，感知机可以在有限步骤中计算出一条直线（或超平面）将这两类完全分开。

如果这两类距离越近，所需的步骤就越多。此时，感知机只保证给出一个解，但是解不唯一，如下图所示：



7.1 对二分类问题的具体描述

训练样本x∈Rnx∈Rn，标签y∈−1,1y∈−1,1，对于线性分类器来说：

参数: w∈Rnw∈Rn and b∈Rb∈R

决策边界（Decision boundary）：w⋅x+b=0w⋅x+b=0

对于一个新的点xx做分类时，预测标签为sign(w⋅x+b)sign(w⋅x+b)

参考上面的描述，在分类正确的情况下，如果一个点xx的标签为y=1y=1，预测值w⋅x+b>0w⋅x+b>0，分类为1；标签为-1，预测值小于0，分类为-1. 那么可以使用y(w⋅x+b)>0y(w⋅x+b)>0来统一表示分类正确的情况，反之可以使用y(w⋅x+b)<0y(w⋅x+b)<0来表示分类错误的情况。

7.2 代价函数

在分类正确时，即y(w⋅x+b)>0y(w⋅x+b)>0，loss=0loss=0;

在分类错误时，即y(w⋅x+b)≤0y(w⋅x+b)≤0，loss=−y(w⋅x+b)loss=−y(w⋅x+b).

7.3 算法的流程

利用随机梯度下降的方式训练模型，每次只使用一个样本，根据代价函数的梯度更新参数，

step1: 初始化w=0,b=0w=0,b=0;

step2: 循环从训练集取样本，每次一个

           if y(w⋅x+b)≤0y(w⋅x+b)≤0（该样本分类错误）:

               w = w + yx

               b = b + y

从流程上来看，每次取出一个样本点训练模型，而且只在分错的情况下更新参数，最终所有样本都分类正确时，模型训练过程结束。

1. 激活函数的作用

激活函数（Activation functions）对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。它们将非线性特性引入到我们的网络中。其主要目的是将A-NN模型中一个节点的输入信号转换成一个输出信号。该输出信号现在被用作堆叠中下一个层的输入。

如果我们不运用激活函数的话，则输出信号将仅仅是一个简单的线性函数。线性函数一个一级多项式。现如今，线性方程是很容易解决的，但是它们的复杂性有限，并且从数据中学习复杂函数映射的能力更小。一个没有激活函数的神经网络将只不过是一个线性回归模型（Linear regression Model）罢了，它功率有限，并且大多数情况下执行得并不好。我们希望我们的神经网络不仅仅可以学习和计算线性函数，而且还要比这复杂得多。同样是因为没有激活函数，我们的神经网络将无法学习和[模拟](https://www.baidu.com/s?wd=%E6%A8%A1%E6%8B%9F&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/zouzhen_id/article/details/_blank)其他复杂类型的数据，例如图像、视频、音频、语音等。这就是为什么我们要使用人工神经网络技术，诸如深度学习（Deep learning），来理解一些复杂的事情，一些相互之间具有很多隐藏层的非线性问题，而这也可以帮助我们了解复杂的数据。

那么为什么我们需要非线性函数？

非线性函数是那些一级以上的函数，而且当绘制非线性函数时它们具有曲率。现在我们需要一个可以学习和表示几乎任何东西的神经网络模型，以及可以将输入映射到输出的任意复杂函数。神经网络被认为是通用函数近似器（Universal Function Approximators）。这意味着他们可以计算和学习任何函数。几乎我们可以想到的任何过程都可以表示为神经网络中的函数计算。

而这一切都归结于这一点，我们需要应用激活函数f（x），以便使网络更加强大，增加它的能力，使它可以学习复杂的事物，复杂的表单数据，以及表示输入输出之间非线性的复杂的任意函数映射。因此，使用非线性激活函数，我们便能够从输入输出之间生成非线性映射。

激活函数的另一个重要特征是：它应该是可以区分的。我们需要这样做，以便在网络中向后推进以计算相对于权重的误差（丢失）梯度时执行反向优化策略，然后相应地使用梯度下降或任何其他优化技术优化权重以减少误差。

只要永远记住要做：

“输入时间权重，添加偏差和激活函数”

1. 激活函数的种类？

（1）Sigmoid函数或者Logistic函数

（2）Tanh — Hyperbolic tangent（双曲正切函数）

（3）ReLu -Rectified linear units（线性修正单元）

其中，

（1）Sigmoid激活函数：它是一个f（x）= 1/1 + exp（-x）形式的激活函数。它的值区间在0和1之间，是一个S形曲线。

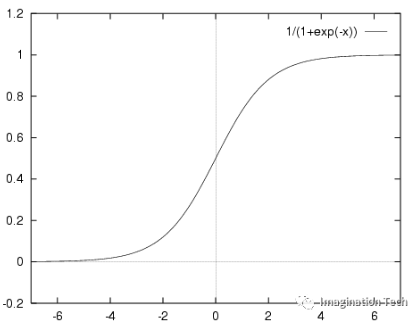
它很容易理解和应用，但使其不受欢迎的主要原因是：

- 梯度消失问题

- 其次，它的输出不是以0为中心。它的梯度更新在不同的方向上且走得太远。 0<output <1，使优化更加困难。<="" p="">

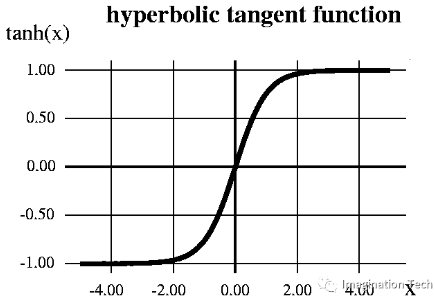
- Sigmoids函数饱和且kill掉梯度。

- Sigmoids函数收敛缓慢。



1. 解决上述问题。

双曲正切函数——Tanh：其数学公式是f（x）= 1 - exp（-2x）/ 1 + exp（-2x）。现在它的输出是以0中心的，因为它的值区间在-1到1之间，即-1<output <1。="" 因此，在该方法中优化更容易一些，从而其在实践应用中总是优于sigmoid函数。="" 但它依然存在着梯度消失问题。<="" p="">



1. 如何处理和纠正梯度消失问题?

ReLu -Rectified linear units（线性修正单元）：其实在过去几年中它就已经非常受欢迎了。最近证明，相较于Tanh函数，它的收敛性提高了6倍。只要R（x）= max（0，x），即如果x <0，R（x）= 0，如果x> = 0，则R（x）= x。因此，只看这个函数的数学形式，我们就可以看到它非常简单、有效。其实很多时候我们都会注意到，在机器学习和计算机科学领域，最简单、相容的技术和方法才是首选，才是表现最好的。因此，它可以避免和纠正梯度消失问题。现如今，几乎所有深度学习模型现在都使用ReLu函数。

但它的局限性在于它只能在神经网络模型的隐藏层中使用。

因此，对于输出层，我们应该使用Softmax函数来处理分类问题从而计算类的概率。而对于回归问题，它只要简单地使用线性函数就可以了。

ReLu函数的另一个问题是，一些梯度在训练过程中可能很脆弱，甚至可能会死亡。它可以导致权重更新，这将使其永远不会在任何数据点上激活。简单地说ReLu可能会导致死亡神经元。

为了解决这个问题，我们引进了另一个被称为Leaky ReLu的修改函数，让它来解决死亡神经元的问题。它引入了一个小斜坡从而保持更新值具有活力。

然后，我们还有另一个变体，它形成于ReLu函数和Leaky ReLu函数的结合，我们称之为Maxout函数。

