Collabrative Filtering(协同过滤算法，用于推荐）

整体流程可见：

<https://blog.csdn.net/xiaokang123456kao/article/details/74735992>

这一篇可以细看：

收集用户偏好（将“用户行为”——转化为“偏好”）

不同行为分组： 用户对某个商品一般是有一系列行为，都能反应偏好，显式反馈（购买）,隐式反馈（点击，查看，浏览等）

加权：对不同的偏好，赋权值，一般隐式反馈>隐式反馈

减噪：用传统“数据挖掘”方法，减少错、脏数据等

归一：无量纲

找到相似物品或偏好

计算相似度——选取相似集（如K近邻）

预测、推荐

回归、ALS等预测，排序取TopN

一般分类：

基于用户（user-based CF）:

基于物品（item-based CF）:

基于用户：

寻找用户-用户，之间的相似性————缺点，用户可能购买物品占比总物品低，无法计算

，比较稀疏，很难找到相似度高的用户，因为大家的购物篮都很不一样（电商品类要比超市多）

基于物品：

寻找物品-物品，之间的相似性————物品相似度，基于历史数据提前算好。

（实际是分析， 用户买过的商品，和未买过的商品————即用户显示过偏好的物品，和未显示过偏好的物品， 例如电商的评分，电影打分等）

相似性度量（3种）：

欧式距离——

皮尔逊相关系数

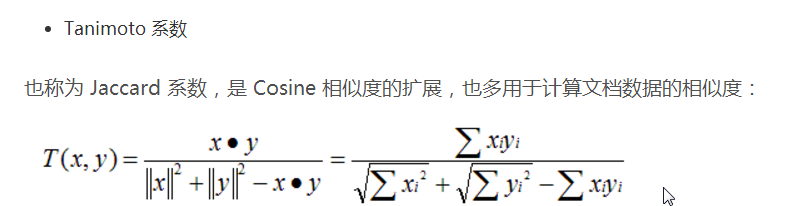
余弦相似度（计算向量夹角）

计算完相似度，要根据用户买过的商品A，寻找一系列“相似商品集合”Set。

此处（一般用K-nearest方法）找K个最近邻的。

根据提前计算过的相似性表， 预测、推荐（商品推荐的打分，排序）。

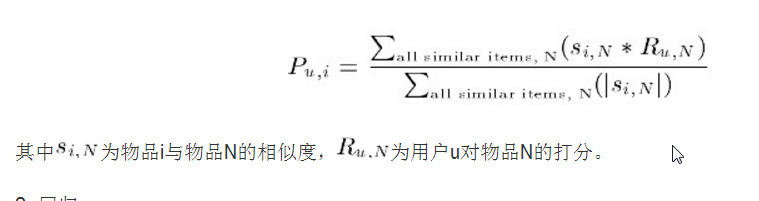
jacard系数



预测：

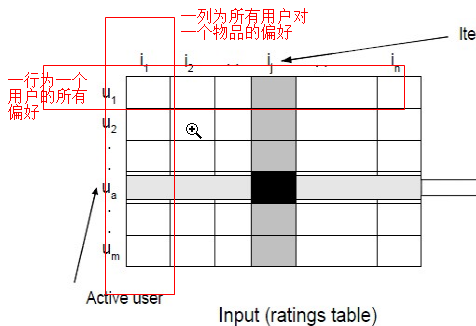
两种方式：

加权： A的偏好 =



其中Pu,i就是偏好。

貌似是用“回归”作预测————就是作预测（应该是这样， 我们知道了“用户-偏好”矩阵，）



那么，其实可以建模回归：（有可能是ALS方法）

比如，y= 用户已评分过的物品， x = 用户未评分过的物品 ，建立回归

（具体再看资料）

计算出用户对不同“物品“”的“偏好”后：

就可以，对“偏好”排序，即推荐：

推荐方式： TopN

假设用户有一系列行为（各有偏好，如评分就是一种偏好），对行为（偏好）作预测，并推荐

总结：CF的3步曲：

计算（偏好）相似度——可以算归类，相似度要归一化（求导数，或最大值归一，或其他归一方法，从而无量纲）

预测（偏好）——（没打分的，我们根据其他用户的数据，根据用户已有行为及相似度或建模，作打分）

推荐——（计算出偏好后，选择偏好最高的）

有点经济学的意思：

可以看看ALS作推荐。

另一篇讲CF的：

<https://blog.csdn.net/xiaokang123456kao/article/details/74735992>

基于用户的CF,很简单：

1.用户的偏好（形成一组向量）；

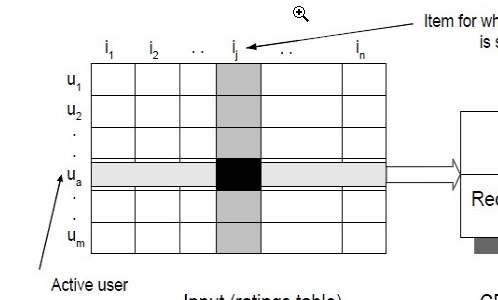
2.计算目标用户A和其他用户的相似度，找出K最近邻（即取K个最近的邻居用户）—K最近邻法。 K作为阀值

3.根据物品的“相似度集合”的每个元素作简单加权，作为权重。 根据所有用户对物品A的偏好，加权求出 用户U1未评分物品A的偏好——对所有未评分物品，都如此求未评分偏好。

4.排序，取topN,推荐给用户。

CF简单讲是解决这类问题：

下面是“用户-偏好”矩阵， 白色为已显示偏好， 求灰色部分（未显示偏好，即预测偏好）：



一篇ALS的博客：

<https://blog.csdn.net/buptdavid/article/details/78970906>

基本和sparkML的CF观点一致：

http://spark.apache.org/docs/2.3.0/ml-collaborative-filtering.html

讲了ALS的原理，有关显示反馈，隐式反馈，冷启动等问题：

显示反馈：例如直接评分，暴力简单，直接反应偏好

隐式反馈：点击，浏览，间接反应偏好， 但是不会给出负反馈

冷启动：当有新用户、新物品时，某个用户的偏好完全为0，就不好去作推荐什么的了，无法“计算”用户的偏好。

sparkML的协同过滤，使用的是ALS方法。

输入为“user-item” matrix，——即用户列user，物品列item，评分列ratings，时间列timestamp, 输入使用（implicit feedback，即隐式反馈数据集）

spark的CF里：

冷启动参数：coldStartStrategy————默认为“nan”,可以设为“drop”表示放弃该行空数据

一篇sparkML调用CF的文章,各种参数的设置:

https://blog.csdn.net/liulingyuan6/article/details/53489390

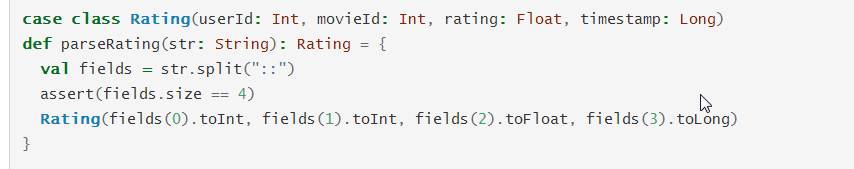
有篇著名的论文：

[Large-Scale Parallel Collaborative Filtering for the Netflix Prize](http://dx.doi.org/10.1007/978-3-540-68880-8_32)

使用ALS-WS

spark构造dataset的方法：

文本数据切割，再用case class类构造。



关于ALS的原理（交替最小二乘法）：

<https://blog.csdn.net/oucpowerman/article/details/49847979>

推荐系统：

1.“user-item” matrix得到的是一个“稀疏矩阵”，高维——

2.利用矩阵分解（一个m\*n维矩阵，使用SVD分解，从而降维，获取两个信息略减的U和V矩阵，分别存储着降维后的user和item的信息，一般信息也是有损的）降维———

3.建立损失函数模型，使损失最小——变成优化问题

4.使用“ALS”或“随机梯度下降（SGD），求最优解

SVD分解特别适用于“不是方阵的一般矩阵分解”,原理：

<https://blog.csdn.net/bitcarmanlee/article/details/52068118>，好好看

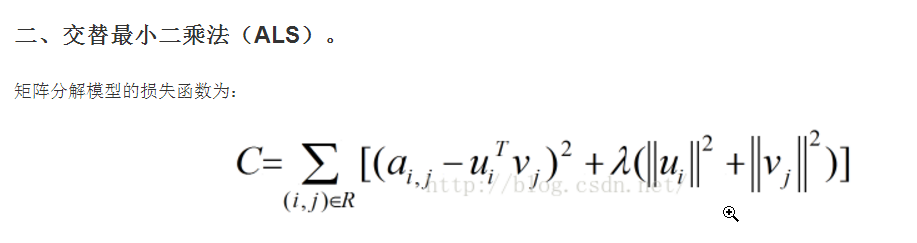
SVD分解公式： A = U\*Σ\*V

U和V都是方阵，Σ是酉矩阵（m\*n阶，只有对角线上有值）

Σ有个牛逼性质：如果对角线上元素排序， 后面的对角元素下降很快，到后面几乎为0，这样我们可以忽略这些值，只去前r个元素， 这样A ≈ U R V。 R是r\*r阶对角方阵。



这样就起到了降维作用了，而且相对于原数据规模，损失不大。



几种矩阵分解：

QR、Jordan、SVD、LU（三角分解）

<https://blog.csdn.net/bitcarmanlee/article/details/52662518>

# 线性回归

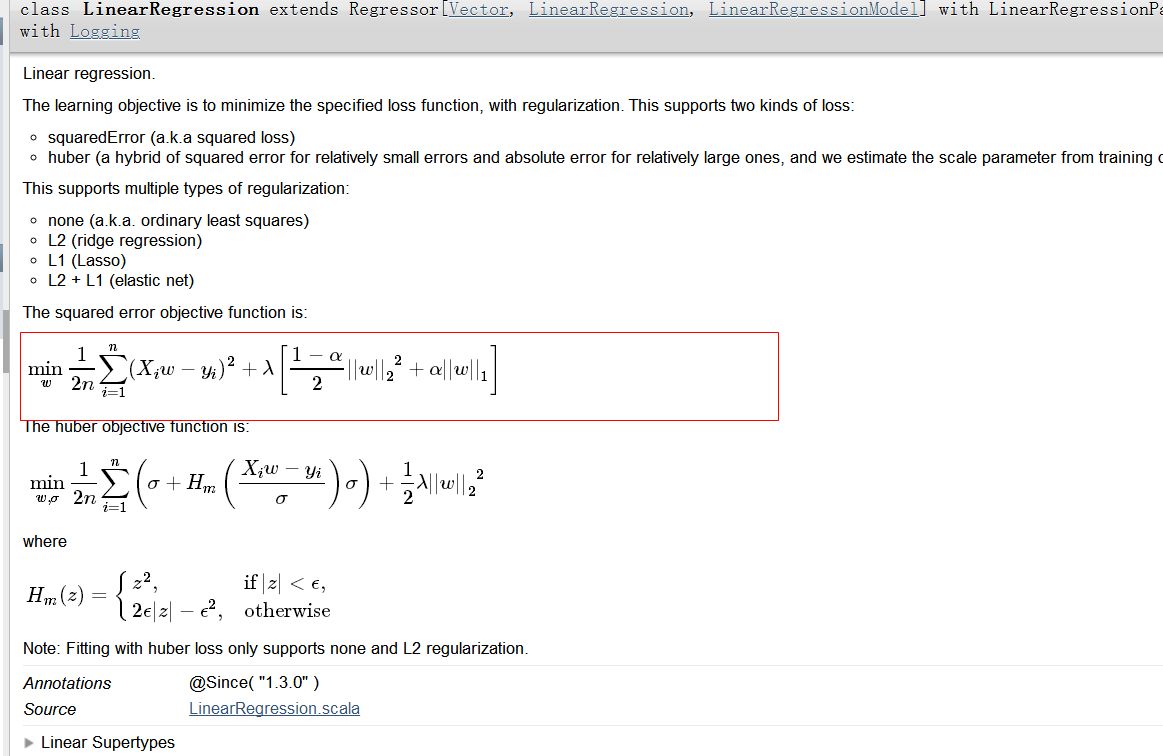
## 理论

## 公式

关于huber回归，见：

<https://blog.csdn.net/qq_29540745/article/details/53400606>

主要用于分类，是RMSE的一种改进，确保鲁棒性



* 1. 参数解释

在ML中，见源码对参数说明：

<https://github.com/apache/spark/blob/v2.3.1/mllib/src/main/scala/org/apache/spark/ml/regression/LinearRegression.scala>

weigetCol的作用：

默认所有特征变量我的权重为1，此时是普通的最小二乘。

若用户设置后，变成WLS，即加权最小二乘

epsilon:

是胡伯回归的参数，epsilon越大时，越接近最小二乘回归（无正则项），越小越接近（Lasso回归）



* 1. 优化-模式识别：

spark ML文档说明：

<http://spark.apache.org/docs/2.2.0/ml-tuning.html>

在模型形式确定的情形下：比如使用Linear Regression时，针对“ train test集的划分” 和 “参数组合” 两个部分进行“模式识别”，寻找最好的那个。

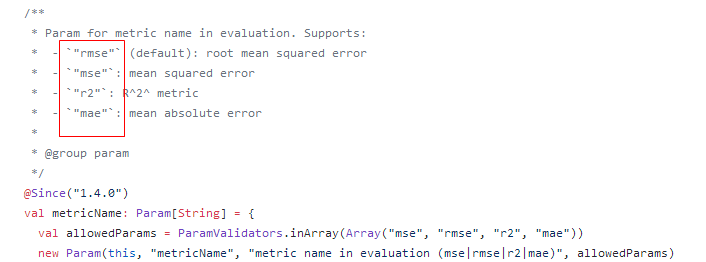
优化需要提供两个部分：

Estimator(ParamGrid就是由其中的参数产生)

Evaluator（评估器）——一般回归和分类各有自己的评估器，评估器可以自己设置参数，不设置的话用默认的即可（评估器的参数较少）

例如RegressionEvaluator:只有1个核心的参数MetricName:

提供了如下4种：



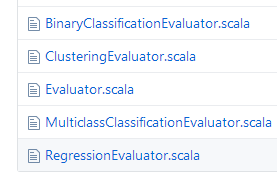
* 1. 评估器

位置：

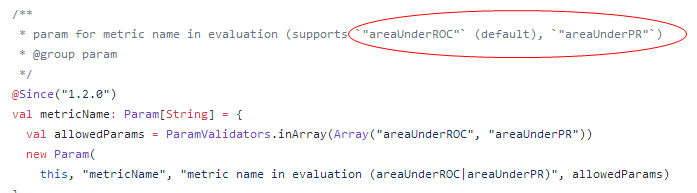
[**spark**](https://github.com/apache/spark/tree/v2.3.1)/[mllib](https://github.com/apache/spark/tree/v2.3.1/mllib)/[src](https://github.com/apache/spark/tree/v2.3.1/mllib/src)/[main](https://github.com/apache/spark/tree/v2.3.1/mllib/src/main)/[scala](https://github.com/apache/spark/tree/v2.3.1/mllib/src/main/scala)/[org](https://github.com/apache/spark/tree/v2.3.1/mllib/src/main/scala/org)/[apache](https://github.com/apache/spark/tree/v2.3.1/mllib/src/main/scala/org/apache)/[spark](https://github.com/apache/spark/tree/v2.3.1/mllib/src/main/scala/org/apache/spark)/[ml](https://github.com/apache/spark/tree/v2.3.1/mllib/src/main/scala/org/apache/spark/ml)/**evaluation/** [RegressionEvaluator.scala](https://github.com/apache/spark/blob/v2.3.1/mllib/src/main/scala/org/apache/spark/ml/evaluation/RegressionEvaluator.scala)

目前spark ML仅有5种评估器（4种实现类）：

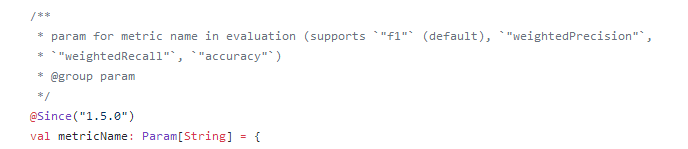
分类、聚类、多分类、回归、抽象类评估器



评估器的使用：一般内部都设置好了计算过程，对于某个评估器（比如回归评估器），有n种MetricName,即评估方法类型，即各种评价指标。选择其一个，基本上就会自动计算（找出最小或最大的那个嘛）

二分类评估其的MetricName则为如下（使用ROC曲线评估）：  


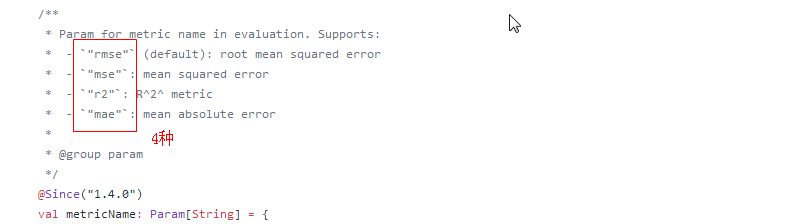
多分类评估器：



聚类评估器：



回归评估器：



在scikit-learn中，也是4种，不过略有不同：

见：<https://www.cnblogs.com/nolonely/p/7009001.html>

有个Explained\_Variance\_Score，还包括和ML相同的3种：mae,mse,r\*\*2

* 1. d
  2. d

1. 逻辑回归
   1. 公式
   2. 参数解释
   3. d
   4. d
   5. d
   6. d
2. 支持向量机
   1. 公式
   2. 参数解释
   3. d
   4. d
   5. d
   6. d
   7. 基本统计量
   8. 管道模型
   9. 分类
   10. 回归
   11. 聚类
   12. 协同过滤
   13. 关联分析
   14. d
   15. d
   16. d
   17. 模式识别&优化
   18. d
   19. d
   20. d
   21. d