常用机器学习算法

图灵：楼兰

五、机器学习-典型算法实现

这一节课，我们来了解一下常用的机器学习算法。目的是结合上一节的特征工程，让大家能够去kaggle上逛一逛，对机器学习入个门。然后我们再来用机器学习的方式来设计推荐系统。

机器学习适用的数据为具有特征值(属性，label)和目标值(结果,point)的数据集。通过从历史数据集中学习经验，建立模型，从而达到预测新特征值对应的目标值的效果。因此在数据方面，越见多识广的数据集(样本集越大越全)，越能进行更可信的预测(越准确的预测)。

算法分类：

- 按有无目标值值分类：1、监督学习：有目标值 2、无监督学习：无目标值
- 按目标值类型分类；1、分类算法：目标值为一组有限且固定的序列 2、回归算法：目标值为一个连续的数值空间

1、KNN算法

定义：

如果一个样本在特征空间中的K个最相似(距离最近)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。人以类聚，物以群分。

距离计算公式：

欧式距离(平方根距离)、曼哈顿距离(绝对值距离)，明科夫斯基距离(以上两种距离均是明科夫斯基距离的特例)

适用案例：

iris，根据鸢尾花的一些特征判断一个鸢尾花所属的种类

适用于小数据场景，K-W数据量级别的样本。

算法优缺点：

- 优点：简单、易于理解，易于实现，无需训练
- 缺点：1、懒惰算法，对预测样本分类时才进行计算，计算量大，内存开销大。
  2、必须指定K值，K值选择会极大程度影响分类的准确度。
  关于K值选取：K值过小，容易受到异常数据的影响。而K值过大，容易受样本不均衡的影响。

特征工程处理：

需尽量保证各个维度的数据公平性。无量纲化-标准化处理。尽量保证各个维度的数据公平性。

skLearn API:

```
    sklearn.neighbors.KNeighborsClassifier(n_neighbor=5,algrithm='auto')
        n_neighbors: int 可选，默认5 ，  K值
        algorithm : {'auto','ball_tree','kd_tree','brute'} .可选。用于计算最近的算法。有ball_tree、kd_tree。不同的实现方式会影响效率，但不影响结果。一般用auto，会自己根据fit方法的值来选择合适的算法。
```

spark Demo:

未找到。

> 从这个示例重点理解机器学习的基础处理流程。然后理解对K值参数的调优对结果的影响。

2、朴素贝叶斯分类算法

朴素：假定了特征与特征之间相互独立，没有影响。

贝叶斯：贝叶斯概率计算公式。

原理：

朴素的概率计算：30%的男人是好人，80%的老人是好人， 那男老人有24%的概率是好人。

> 概率基础知识：独立的定义 P(A,B) = P(A)*P(B)
>
> 全概率公式：
> P(A)=P(A|B1)*P(B1)+P(A|B2)*P(B2)+P(A|B3)*P(B3)+P(A|B4)*P(B4)+
> 。。。。

应用场景：

文本分类、评论区分好差评

优缺点：

优点：发源于古典数学理论，有稳定的分类效率。

对缺失数据不敏感，算法也比较简单，常用语文本分类。

分类速度快，准确度相对较高。

缺点：由于使用了样本属性独立的假设，当特征属性之间有关联时，其效果就不太好。

基于概率论的简单统计，样本数量越大越好。

Sklearn API:

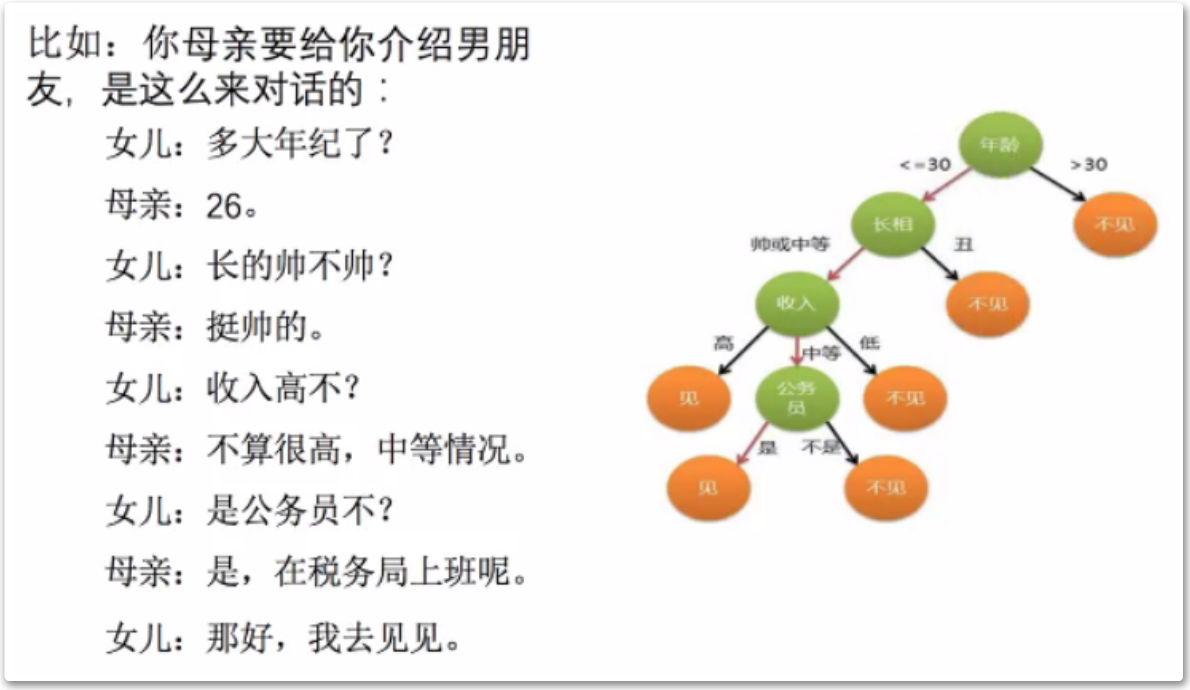
```
    sklearn.naive_bayes.MuitinomialNB(alpha = 1.0)
    alpha:拉普拉斯平滑系数，一般用1 。用来防止计算出的概率为0.使用方式是在分子和分母上一起添加平滑系数。
```

spark 示例：

JavaNaiveBayesExample

3、决策树

决策树是用来解决分类问题的。决策树模型就是一个多层的if-else结构。



分类原理：

决策树的关键就是判断的关键特征的先后顺序，要能尽量快速过滤掉大部分不符合标准的情况。

决策树的划分依据之一：信息熵 entropy，信息增益。信息熵可认为是信息的混乱程度。而信息增益可以理解为按某一个属性进行划分后的信息熵增益。决策树的划分方式就是在每一个节点选择信息增益最大的属性进行划分。

关于信息熵和信息增益的计算，在数学上有明确的计算公式，但是这里就不去过多推演了。

SkLearn API

```
sklearn.tree.DecisionTreClassifier(criterion='gini',max_depth=None,random_state=None)
    决策树分类器
    criterion: 默认是'gini'系数，也可以选择信息增益的熵'entropy'
    max_depth:树的深度大小
    random_state:随机数种子
```

> 熵的计算比较复杂，默认会使用相对比较简单的gini系数。gini系数可以理解为是信息熵的一个近似结果。

spark Demo:

JavaDecisionTreeClassificationExample、
JavaDecisionTreeRegressionExample

优缺点：

优点： 决策树模型的可解释能力强，不需要进行数据归一。模型就是一个多层的If-else结构，比较容易理解。
缺点：每个决策边界只能涉及到一个特征，不太容易扩展到更复杂的情况，整个决策树容易过大。树的深度过大就容易产生过拟合。

过拟合的调整方式是配置maxDepth参数，修改树的深度。改进方法是进行剪枝，防止整个树过于庞大。另一种比较好的方式就是采用随机森林

4、随机森林

通过建立几个模型组合来解决单一预测问题。其原理就是生成多个分类器(模型)，各自独立地学习和做出预测。这样预测最后结合成组合预测，一般都优于任何一个单一分类器做出的预测。其输出的类别就是取多数的结果。

原理：

BootStrap随机有放回抽样。从训练集中随机取一个样本，放入新训练集。然后从原训练集中再随机抽取(已抽取的样本放回原训练集)。这样，每棵树的训练集都是独有的。
相当于将数据分散成不同的树。随机的预测思想就是这些树中有“正确”的树，也有“错误”的树。而“错误”的树的学习结果会互相抵消，最终“正确”的树就会脱颖而出。

sklearn API：

```
sklearn.ensemble.RandomForestClassifier(n_estimators=10,criterion='gini',max_depth=None,bootstrap=True,random_state=None,min_samples_split=2)
    n_estimators: Int 默认10 森林里的树木数量
    criteria:  String  默认gini . 分隔特征的测量方法  entropy
    max_depth: Integer或者None,可选。树的最大深度
    max_features="auto" 每个决策树的最大特征数量
        auto,  sqrt  log2   None
        auto和sqrt是一个意思，开根号。 None取跟原样本一样的特征树。
    bootstrap: boolean 默认true 是否在构建树时使用放回抽样
    min_samples_split:节点划分最少样本数
    min_samples_leaf:叶子节点的最小样本数
```

超参数：n_estimator, max_depth,min_samples_split,min_samples_leaf。

Spark Demo:

JavaRandomForestClassifierExample

优缺点：

在所有分类算法中，具有较好的准确率。能够有效的运行在大数据集上，处理具有高维特征的输入样本，而且不需要降维。能够评估各个特征在分类问题上的重要性。

5、线性回归

回归问题：目标值是一组连续的数据。找到一种函数关系，来表示特征值与目标值之间的关系。

而线性回归就是以一个多元一次函数的方式来尽量的逼近所有的目标点。

函数形式： $y=f(x)=w_1*x_1+w_2*x_2+w_3*x_3+.....+b$。 如果是二维 $y=w_1*x_1+b$的话，大家应该非常熟悉，在坐标系上就是一条线，所以叫线性回归。

线性回归问题的关键就是要找出一组最适合的参数值w(形式为一个矩阵)和一个偏移量b。

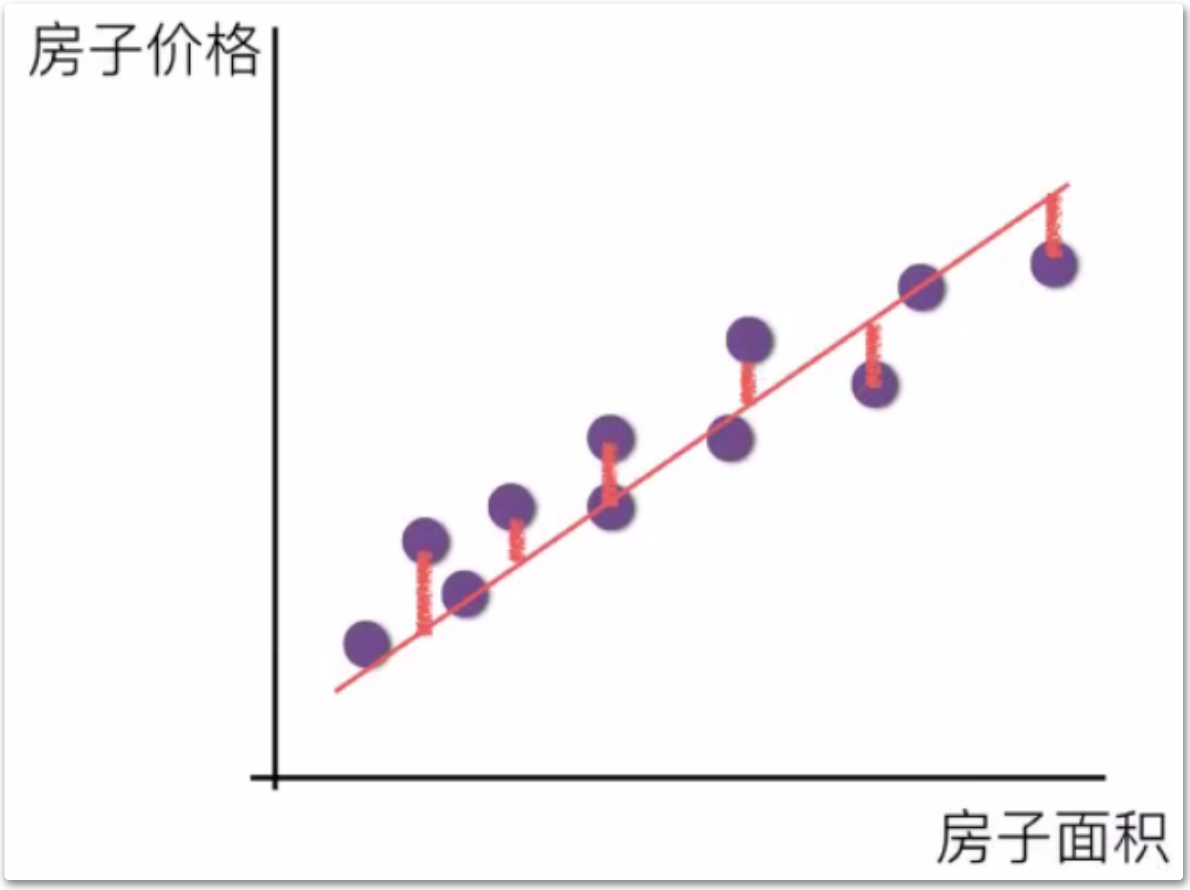
原理：

用一个多元一次方程来描述特征值与目标值之间的关系，线性关系。

使用场景：波士顿房价预测Demo

目标：

模型参数能够尽量准确的预测目标值。--损失函数最小。



SKlearn API：

```
--正规方程线性回归
sklearn.linear_model.LinearRegression(fit_interfcept=True)
- 通过正规方程优化
- fit_intercept：  是否计算偏置--参数序列最后的b
- LinearRegression.coef_ ：  回归系数
- LinearRegression.intercept_ ：  偏置

--随机梯度下降线性回归
sklearn.linear_model.SGDRegressor(loss="squard_loss",fit_intercept=True,learning_rate='invscaling',eta0=0.01)
- SGDRegressor类实现了随机梯度下降学习，支持不同的loss函数和正则化惩罚项来拟合线性回归模型
- loss：损失类型  squared_loss 普通最小二乘法
- fit_intercept：是否计算偏置
- learning_rate：string,可选项：
    constant ：eta=eta0
    optimal：eta=1.0/(alpha*(t+t0)) --default
    invscaling.：eta = eta0/pow(t,power_t)
    常用constant
  返回结果
```

```
 SGDRegressor.coef_ ：回归系数
 SGDRegressor.intercept_ ：偏置

--计算均方误差
sklearn.metrics.mean_squared_error(y_true,y_predict)
return  浮点数结果
```

SparkDemo:

JavaLinearSVCExample，计算正规方程。

JavaLinearRegressionWithElasticNetExample：包含了误差计算。这个ElasticNet就是线性回归用于优化损失函数的一种方式。

正规方程与线性回归

在线性回归问题中，机器学习的目标就是不断减少损失函数。而优化损失函数的方法有两种：

- 一种是像我们计算二元一次方程组一样，直接用公式去计算最佳的一组解。称为正规方程。这种方式不需要学习，直接求解。但是运算在大数据量下的计算会非常复杂，通常只适用于小数据量。
- 另一种是先假定一组解，然后不断试错，慢慢逼近正确答案。称之为梯度下降。

几种梯度下降的优化方法：

GD： Grandient Descent ; 原始的梯度下降方式。

SGD：Stochastic Grandient Descent：随机梯度下降。比较高效，节省时间。缺点是需要许多超参数，对特征标准化敏感。

SAG：Stochastic Average Gradient 。 随机平均梯度法。比SGD的收敛速度更快

> spark示例： JavaSVMWithSGDExample

模型评估

计算损失函数的方法也是有很多的，比如均方误差RMSE，MSE，R2等。这些参数都可以用来对线性回归模型进行评测。

过拟合与欠拟合

这也是机器学习过程中最为核心的问题。正规方程直接计算出最佳的结果，是不是就是最好的？其实也不是，数据之间的规律并不是稳定的，一般就不存在绝对的规律。机器学习的目的其实是要能够去对未来的数据进行预测，是一种模糊的行为。正规方程还有一个最重要的问题，就是他通常在训练集上表现良好，但是到验证集上表现就会差很多。这样的模型泛化能力不够，不能很好的体现未来数据的特征。这种问题就是**过拟合现象**，即机器学习从数据集中学到的规律过多。就像我们平常说的书呆子，学少了不好，学太多了同样也不好。

另外还有一种情况，如果数据的样本比较少，那机器学习学到的规律也会太不靠谱了。这就称为**欠拟合现象**。通常表现为算法在训练集和测试集上的表现都不太好。这就像我们常说的学渣，就是学习还不够。

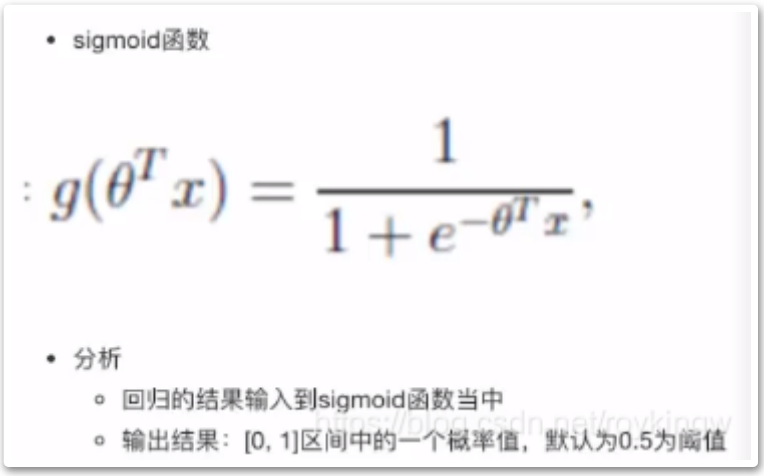
过拟合和欠拟合现象是机器学习过程中绕不开的问题。而算法工程师很多的工作就是要综合评测算法和参数，在过拟合和欠拟合之间寻找最佳解。通常，对于欠拟合现象，可以通过加数据、加特征等手段来优化。而过拟合的优化就比较麻烦，通常需要做一些针对性的特征工程。

6：逻辑回归与二分类

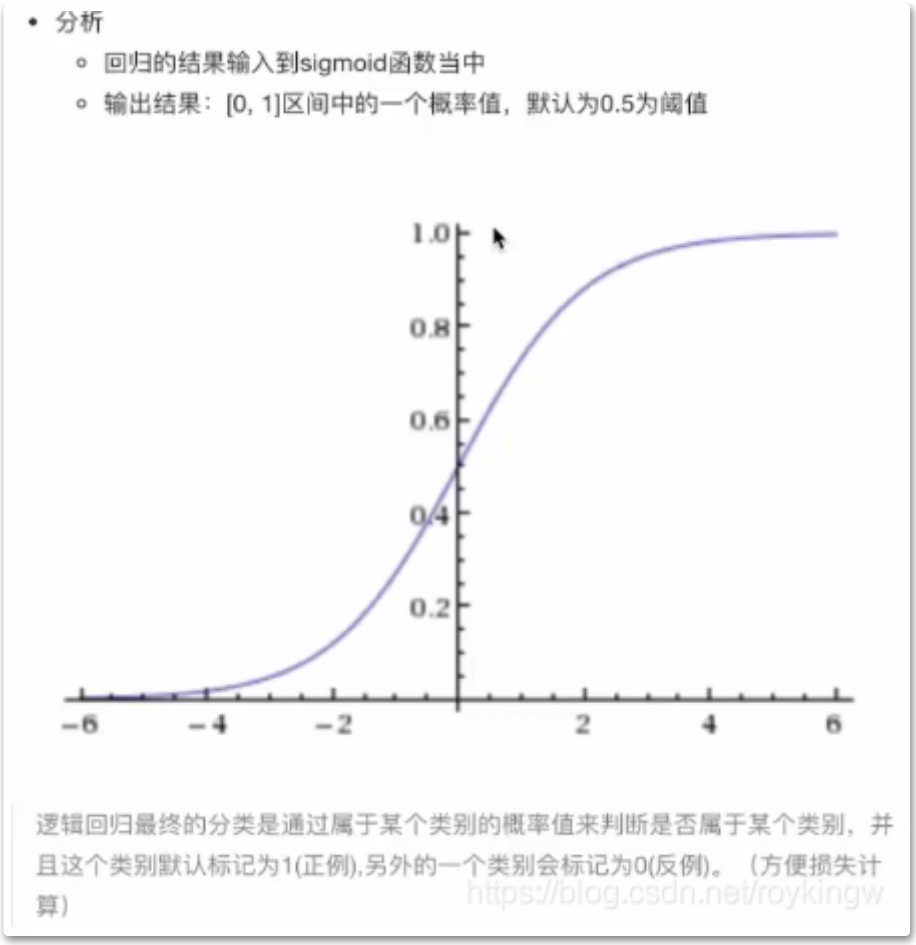
二分类问题： 是否垃圾邮件、是否金融诈骗、是否虚假帐号。。。
这里即是用逻辑回归来解决二分类问题。

逻辑回归原理：

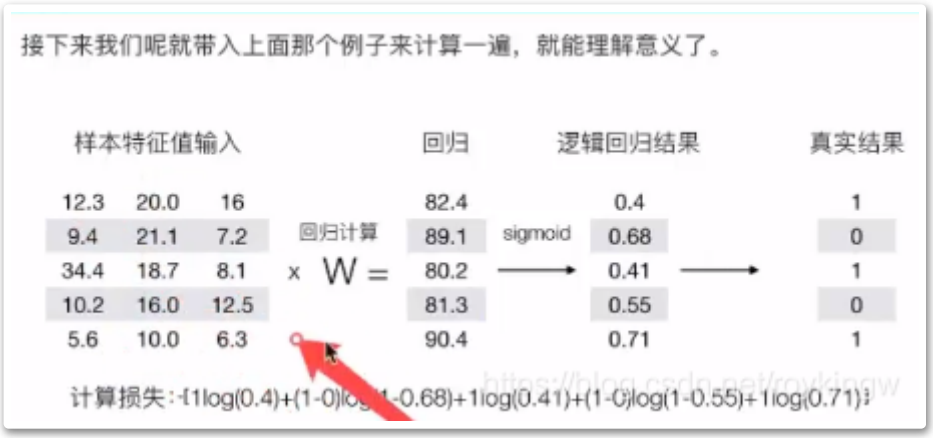
先对一组数据建立线性回归模型，h(w) = w1*x1 + w2*x2+...+b
然后用线性回归的输出作为逻辑回归的输入，将特征值映射到一个分类中，作为预测的目标值。例如：sigmoid激活函数：



线性回归的输出作为逻辑回归的输入



例如，看下图的计算示例，逻辑回归的结果可以认为是样本的二分类概率。然后，同样通过损失函数来计算逻辑回归的模型性能。



逻辑回归API：

```
sklearn.linear_model.LogisticRegrsssion(solver='liblinear',penalty='l2',C=1.0)
- solver: 优化求解模式。 默认 liblinear ， 还有sag 随机平均梯度下降
- penalty: 正则化的种类。 l1 l2
- C: 正则化力度
```

SparkDemo:

JavaLogisticRegressionSummaryExample，

JavaLogisticRegressionWithLBFGSExample

逻辑回归模型评估：

逻辑回归的结果只是一个二分类的概率，如有80%的可能是垃圾邮件。然而这种概率结果其实是很虚的，用模型评估拿到也是一个概率，闭上眼睛都选C也是一种概率。那要怎么评估模型的可信度呢？

混淆矩阵、精确率(Precision)、召回率(Recall)

在二分类任务中，预测结果与正确标记之间存在四种不同的组合，构成混淆矩阵(这个矩阵其实在多分类问题中也能建立，只是更加复杂)，通过混淆矩阵，可以对分类结果从多个不同的方面来进行评价。

真实结果\预测结果	正例	假例
正例	真正例TP	为反例FN
假例	伪正例FP	真反例TN

精确率Precision： 预测结果为正例样本中真实为正例的比例：

一批西瓜中，预测为好西瓜的瓜有多少真正是好西瓜。体现模型的准确性。--要把好西瓜挑出来

召回率Recall：真实为正例的样本中预测结果也为正例的比例：对正样本的区分能力。

一批西瓜中，所有真正为好西瓜的西瓜中有多少是被正确预测为好西瓜。体现模型对正样本的区分能力。--要把坏西瓜扔掉

另外，还有其他更复杂的评估标准。如F1-Score，反映了模型的稳健型

$$F1 = \frac{2TP}{2TP + FN + FP} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

混淆矩阵计算API：

```
`sklearn.metrics.classification_report(y_true,y_pred,labels=[],target_name=None)`
- `y_true ： 真实目标值数组`
- `y_pred: 估计器预测目标值`
- `labels: 指定类别对应的数字`
- `target_names: 目标类别名称`
- `return:  每个类别精确率与召回率`
  - `precision 精确率`
  - `recall 召回率`
  - `f1-scoreL 稳健度`
  - `support 样本数`
```

有了这些指标后，能够一定程度上评估模型的健康度。但是光有这些指标还不太够，例如在样本不均衡，正样本太多，负样本太少时，这些指标评估结果就不太可信。而为了能更准确的评估二分类模型，就引入了ROC曲线和AOC指标。

ROC曲线和AUC指标

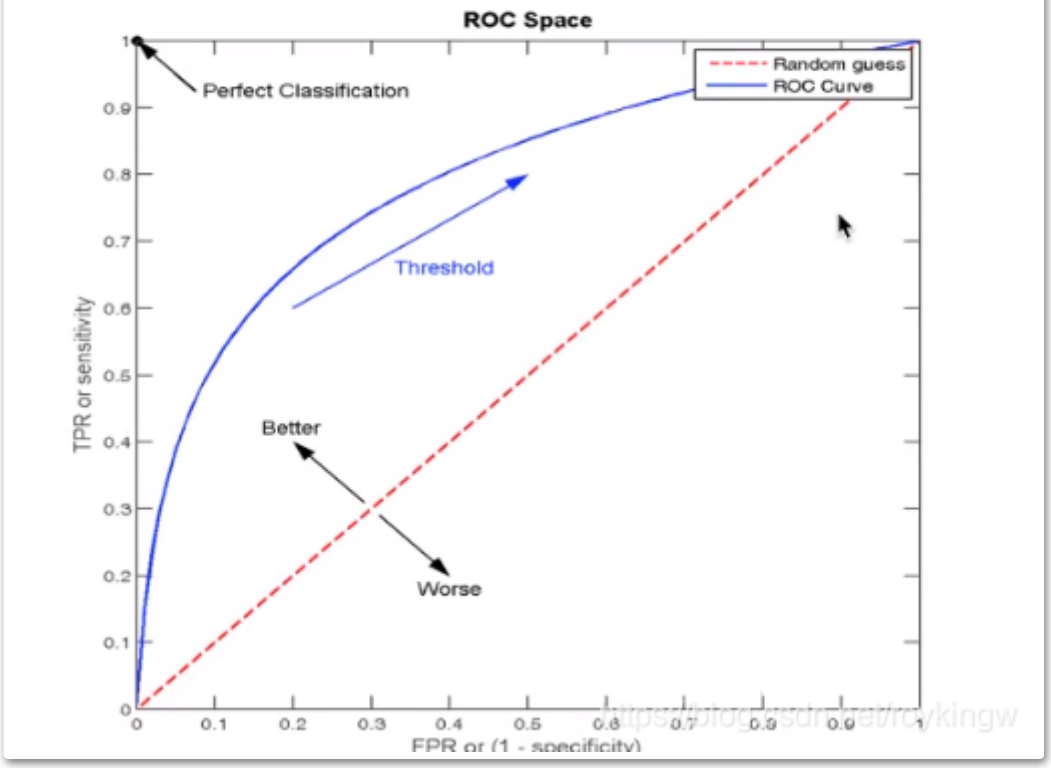
首先需要了解TPRate和FPRate

- TPRate = TP / (TP+FN): 所有真实类别为1的样本中，预测类别也为1的比例
- FPRate = FP/(FP + TN):所有真实类别为0的样本中，预测类别为1的比例

有这两个数据后，对于每一个分类器，就可以建立一条ROC曲线

2 ROC曲线

- ROC曲线的横轴就是FPRate，纵轴就是TPRate，当二者相等时，表示的意义则是：对于不论真实类别是1还是0的样本，分类器预测为1的概率是相等的，此时AUC为0.5



而AUC指标就可以认为是ROC曲线下方的图形面积。

因此，

- AUC指标的概率意义是随机取一对正负样本，正样本得分大于负样本的概率。
- AUC指标的最小值是0.5，最大值是1，取值越大越好
- AUC=1，就是完美分类器，采用这个预测模型时，不管设定什么阈值都能的出完美预测结果。但是，在绝大多数预测的场合，都不存在完美分类器。
- 0.5<AUC<1,优于随机猜测。这个分类器(模型)妥善设定阈值的话，能有预测价值。
- 如果AUC<0.5，就会采用1-AUC来表示AUC的值，代表反向预测。
- 同时，AUC指标也能用于比较多个不同的分类器的性能。

AUC指标计算API：

```
    `sklearn.metrics.roc_auc_score(y_true,y_score)`
- `计算ROC曲线面积，即AUC值`
- `y_true: 每个样本的真是类别，必须为0-反例，1-正例 标记`
- `y_score:预测得分，可以是正类的估计概率、可信值或者分类器方法的返回值`
```

Spark计算AUC示例：

见JavaLBFGSExample中一段示例代码

```
// Get evaluation metrics.
    BinaryClassificationMetrics metrics =
      new BinaryClassificationMetrics(scoreAndLabels.rdd());
    double auROC = metrics.areaUnderROC();
```

AUC总结

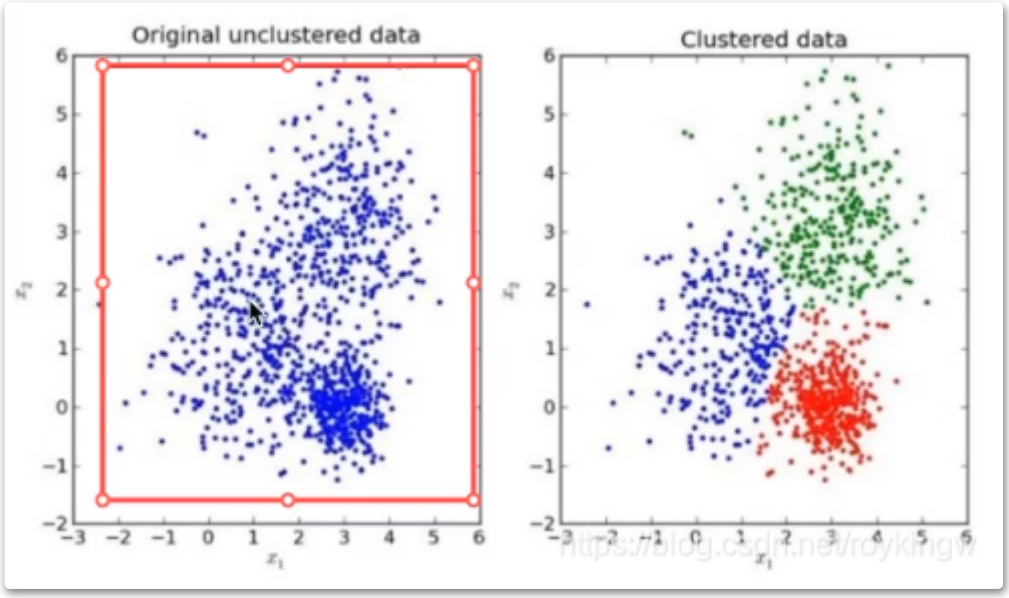
- AUC只能用来评估二分类问题
- AUC非常适合评价样本不均衡时的分类器性能。

7： 无监督学习-K-means算法

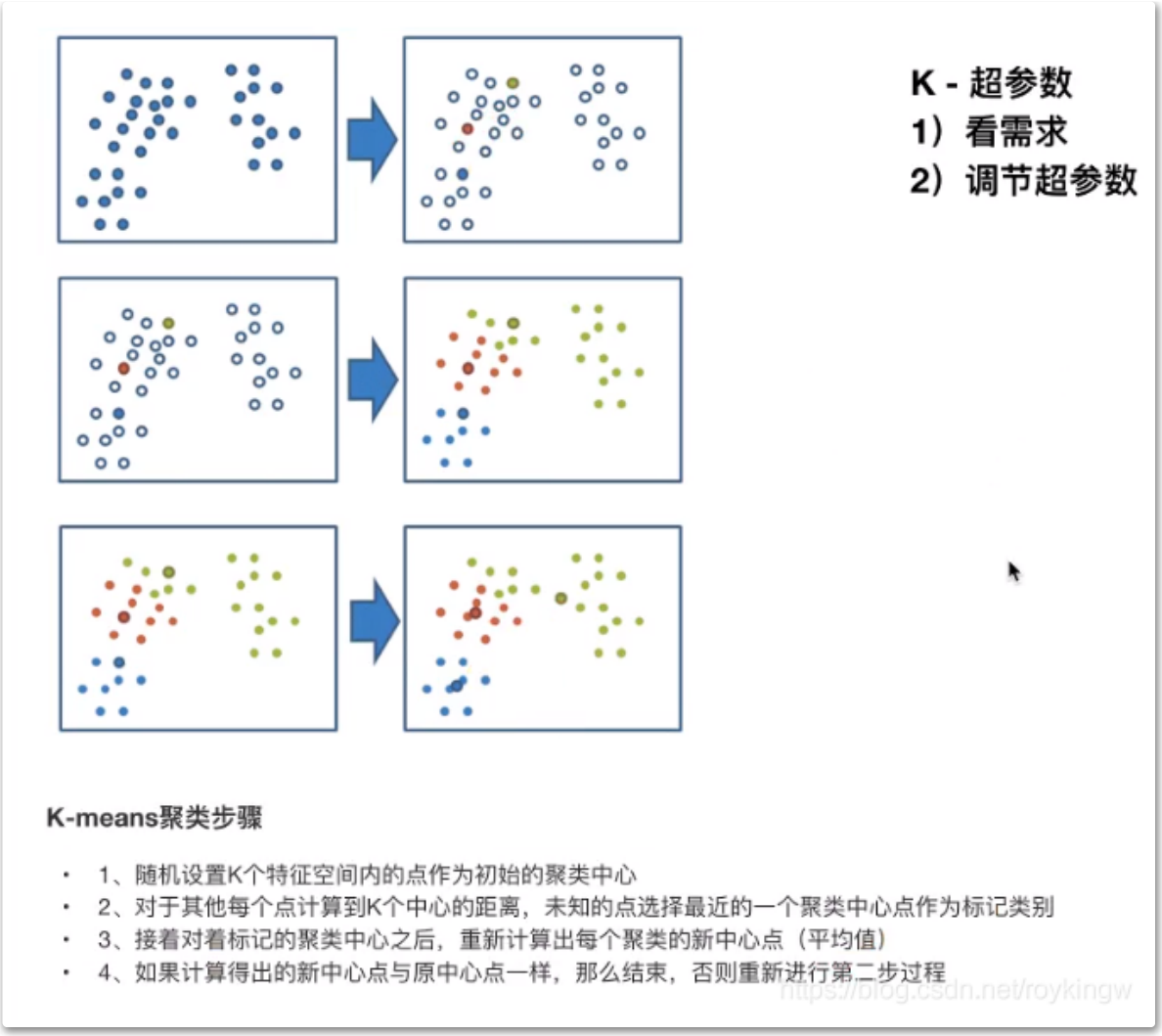
无监督学习：无目标值
示例场景：一家广告平台需要根据相似的人口学特征和客户消费习惯将目标客户分成不同的小组，以便广告客户可以通过有关联的广告解除到他们的目标客户

算法：K-Means聚类， PCA(主成分分析法)降维

以下KMeans效果图： 将一群点随机分成三类



而下面这个图说明了KMeans的计算步骤：



KMeans API:

kMeans实现走预估器代码流程

```
sklearn.cluster.KMeans(n_clusters=8, init = 'k-means++')
n_clusters:开始的聚类中心数量
init : 初始化方法。
labels_ : 默认标记的类型，可以和真实值比较(不是值比较)
```

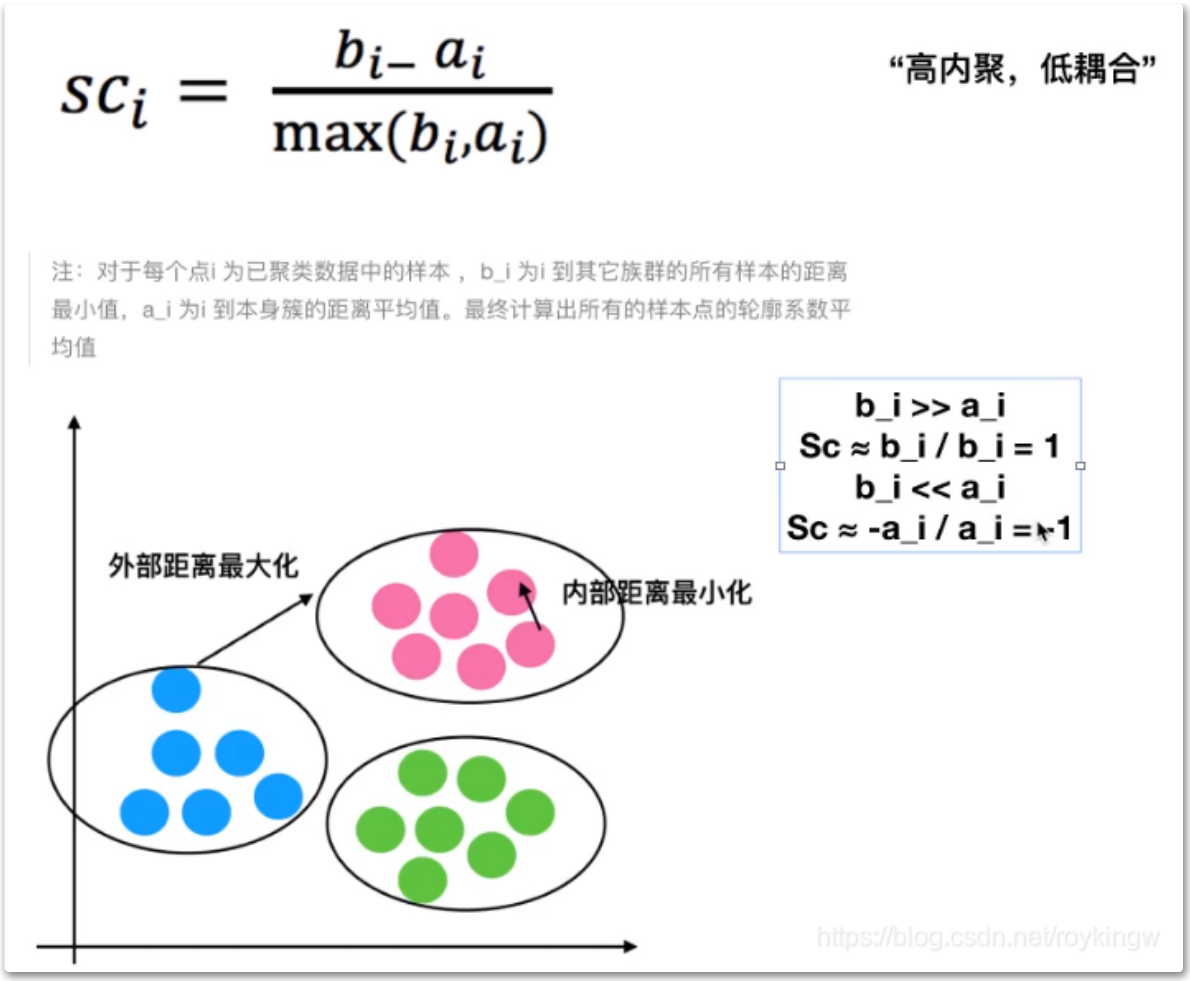
SparkDemo:

ml.JavaKMeansExample

mllib.JavaKMeansExample

kMeans性能评估:

KMeans采用轮廓系数来进行评估，轮廓系数在-1，1之间，越接近1，分类效果越好，越接近-1 ，分类效果越差。



轮廓系数API：

```
sklearn.metrics.silhouette_score(X,labels)`
计算所有样本的平均轮廓系数`
X：特征值`
labels：被聚类标记的目标值
```

在Spark的ml包下的JavaKMeansExample中也有计算轮廓系数的方式。

```
ClusteringEvaluator evaluator = new ClusteringEvaluator();
double silhouette = evaluator.evaluate(predictions);
System.out.println("Silhouette with squared euclidean distance = " +
silhouette);
```

KMeans特点：

特点： KMeans采用迭代式算法，直观易懂并且非常实用
缺点：如果样本分布不均匀，且一开始随机选取的聚类中心过于局部集中 ， 就容易被收敛到局部最优解。解决办法：可以多次聚类，多随机选取几次初始中心点来解决。

应用场景：聚类一般用在分类之前。没有目标值，先用聚类计算出一部分目标值，再用目标值进行分类。

总结

到这里为止，机器学习一些经典的工具和算法就给大家介绍完了。这一部分主要是给大家在机器学习领域入个门，对于一些比较基础的机器学习问题，不说一定就能做出来，至少有一些想法了把。

在最开始介绍推荐系统问题时提到过，我们之间简单的按照物品销售量排名，其实也算是一个推荐系统，但是他的坏处是没有考虑到个体的差异，也就是没有利用到推荐矩阵中已有的历史记录。而通过这些机器学习的方法，就可以更深入的挖掘已有的数据中的潜在规律，制定更好的推荐策略。下一章节我们就会回归到推荐系统的问题，看看如何用机器学习来优化推荐系统。

最后，其实我们这里介绍到的机器学习，还是属于传统的机器学习部分，大都属于之前介绍过的符号主义。他们有一个比较共同的特点，就是机器学习的过程其实都是一些严格的数学计算，也就是说，只要我们理论知识足够扎实，这些机器学习算法的整个过程和结论，都是我们可以完全掌握的。但是，其实还有一大类的问题，已经超出了我们理论能够理解的范畴。比如AlphaGo下围棋，如果科学家能够完全掌握AlphaGo学习到的算法，那就完全可以自己学习AlphaGo，超过所有专业围棋手了对吧。还有比如现在很火的DeepMind，已经在星际争霸等游戏中开始用机器学习来进行模拟改进了。在这样一些领域，我们的这些机器学习算法就非常乏力了。这时候就需要更有力的机器学习算法才行，这就是**深度学习**。也就是之前给大家介绍过的花书的内容。这些就有待大家深入探索了。

> 文档：C3_常用机器学习算法.md
> 链接：http://note.youdao.com/noteshare?id=389b2ed393a9c213ccc27947376d511f&sub=898E47E8C4214A0D9D9A3E8DE8B8B8F1