Machine Learning

1. Common Interview Questions

1.1 What kind of models are sensitive to missing values?

涉及到距离度量的模型，对缺失数据比较敏感，比如SVM和KNN。

线性模型的代价函数往往涉及距离的计算，计算预测值和真实值只差，这很容易导致对缺失值敏感。

神经网络的鲁棒性强，对缺失值不是非常敏感，但一般没有那么多数据可使用。

贝叶斯模型对于缺失数据比较稳定，数据量小时建议首先。

总结，对于有缺失值的数据在经过缺失值处理后

1）数据量小，用朴素贝叶斯

2）数据量适中或者较大，用树模型，优先XGboost

3）数据量较大，可以用神经网络

4）避免使用用距离度量相关的模型，如KNN和SVM

1.2 Normalization VS Standardization

标准化（standardization）是按照特征矩阵的列处理数据，其通过z-score的方法，将样本的特征值转换到同一量纲下，公式为（X-mean)/std，其中, mean是平均值，std是方差。
对标准化的几何理解就是：先将坐标轴零轴平移到均值这条线上，然后进行缩放，即平移加缩放。这样，对于每个属性（每列）来说，所有数据都集中在0附近，方差为1。

归一化 (normalization)是依照特征矩阵的行处理数据，其目的在于样本向量在点乘运算或是其他核函数计算相似性时，拥有统一的标准，也就是都转化成单位向量。公式为(X-min)/(max-min)，其中max 和min分别为该属性的最大值和最小值。他就是一种缩放。缩放后的范围在0-1间。

1.2.1 Normalization approaches

1）线性函数转换：$y = (x - min)/(max\ -\ min)$

2）对数函数转换：$y = log_{10}x$

3）反余切函数转换：$y = arctan(x)^{2}/\pi$

4）减均值，除方差：$y = (x - mean)/std$

1.3 Machine Learning Pipeline

1) 明确问题。明确问题是机器学习的第一步。ML的训练过程通常非常耗时，胡乱尝试的时间成本很高。这里的抽象成数学问题，指的是明确我们可以获得什么样的数据，目标是一个分类还是回归或者是聚类问题。

2）获取数据。数据决定了ML结果的上限，而算法只是尽可能得逼近这个这个上限。数据要有代表性，否则必定过拟合。而对于分类问题，数据偏斜不能太严重，不同类别的数据量不要有数量级的差别。而且要对数据的量级有评估，多少样本，多少个特征，可以估算其对内存的消耗成都，判断训练过程中是否能放得下。如果放不下就考虑改进算法或使用降维技巧，如果实际量实在太大，就需要考虑分布式了（cloud computing)。

3）特征预处理与特征选择

4）训练模型及调优。虽然现在大量的算法都能封装成黑盒直接调包使用，但真正考验技术的调整这些算法的超参数，使得结果变得更好。这需要我们对算法的原理有深入的理解。理解越深，越能发现问题症结，提出良好的调优方案。

5）模型诊断。如何确定模型调优的思路？这就需要有对模型进行诊断的技术。过拟合、欠拟合的判断是模型诊断中至关重要的一步。常见方法有交叉验证、绘制learning curve等。过拟合的基本调优思路是降低模型复杂度，增加数据量；欠拟合的基本调优思路是增加模型复杂度，提高特征数量和质量。诊断后的模型需要进行调优，调优后的模型要重新诊断，这是个反复迭代不断逼近的过程，需要不断尝试，进而达到最优。

6）一般来说，模型融合后都能使得效果有一定的提升，常见的融合方法有boosting和bagging。工程上，主要提升算法准确度的方法是分别在模型的前端（特征清洗和预处理，不同采样模式）与后端（模型融合）上下功夫。而直接调参的工作不会很多，毕竟大量数据训练起来太慢，而且效果难以保证。

7）上线运行。工程上是结果导向的，模型在线上运行的效果直接决定模型的成效。不单纯包括准确度、误差等，还有运行速度（时间复杂度）、资源消耗程度（空间复杂度）、稳定性是否可接受等。

1.4 No free lunch theorem

没有免费的午餐定理：对于训练样本，不同的算法A/B在不同的测试样本中有不同的表现，这表示：对于A，在某些问题上它的表现比B更好，但必然存在一些问题，用B比A处理效果更好。

但是没有免费午餐定理假设所有问题出现几率相同，实际应用中，不同的场景中，会有不同的问题分布，所以在优化算法中，针对具体问题进行问题才是优化算法的核心所在。

1.5 Common loss functions

损失函数用来评价真实值与预测值不一样的程度，损失函数越好，通常模型的性能越好，不同的模型对应不一样的损失函数。

Log loss, Hinge loss,指数损失函数，0-1损失函数以及最小二乘法。

最小二乘法 (Ordinary Least Squares)是一种数学优化技术。函数表示为 $min \sum_{i=1}^{n}(y_m - y_i)^2$，使误差平方和达到最小以寻求估值的方法，就是最小二乘估计。

均方误差MSE是一种加权最小二乘，它的权值是概率。函数表示为 $\frac{1}{n} min \sum_{i=1}^{n}(y_m - y_i)^2$ 。

注意：如果变量个数大于观察值个数，OLS不是好的选择，因为在高维数据集中，我们不能计算唯一的最小二乘法系数估计，方差变成无穷大。相应的，我们可以用惩罚回归的方法，如L1,L2回归，用来缩小系数以减小方差。当OLS具有较大方差的时候，L2回归最有效。

1.6 Covariance and correlation

相关性是协方差的标准化格式。协方差本身难以作比较。比如计算工资和年龄的协方差，因为他们有不同的度量，所以会得到不能做比较的协方差。

$$\Sigma_{ij} = \operatorname{cov}(X_i, X_j) = \mathrm{E}[\,(X_i - \mu_i)(X_j - \mu_j)\,] = \mathrm{E}[\,X_i X_j\,] - \mu_i \mu_j$$

为了解决这个问题，我们计算相关性的达到一个介于-1和1之间的值，就可以忽略它们各自不同的度量。

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

可以用协方差分析来捕获连续变量和分类变量之间的相关性。若你变大，同时我也变大，说明两个变量是同向变化的，这时协方差为正；你变大，但同时我变小，那么协方差为负。

1.7 Linear and Non-linear classifiers

线性分类器的可解释性强，计算复杂度较低，不足之处是模型的拟合效果相对较弱。当特征比数据量大的时候，通常使用线性分类器，因为数据在高维空间内的分布相对稀疏，通常可以直接线性划分。
非线性分类器的拟合能力较强，不足之处是数据量不足容易过拟合、计算复杂度高、可解释性不好。当维度极低的时候，低维空间可能很多特征都跑到一起去了，导致线性不可分，我们就用非线性分类器。
常见的线性分类器有：Logistic Regression, Perception, Linear regression
常见的非线性分类器有：RF, DT, GBDT, NN
SVM两种都有，看是线性核还是高斯核。

1.8 Feature Engineering

特征工程主要包括数据与特征处理、特征选择和降维三部分。

1.8.1 Preprocessing

1）数据选择、清洗、采样。
- 数据格式化
- 数据清洗：填充缺失值，去除脏数据及异常值，以及缺失量太大的特征
- 采样

2）不同数据的特征处理
- 数值型：归一化/标准化、log等变化、统计值(min/min/mean/std等）、离散化、分桶等
- 类别型：one-hot编码等
- 时间型：提取出连续值的持续时间和间隔时间；提取离散值的年、月、日等信息
- 文本型：tf-idf等
- 统计型：加减平均、分位线、次序、比例等

意义：
- 对数据进行预处理，可以提高数据质量，提高挖掘质量。对数据进行清洗可以填充缺失值、光滑噪声数据，识别和删除异常值，保证数据的一致性
- 使用正确的采样方法可以解决因数据不平衡带来的预测偏差

- 对不同的数据类型进行不同的特征处理有助于提高特征的可用性。比如对数值型数据进行归一化可以将数据转为同一量纲下。对类别型数据，可以用one-hot编码将类别数据数字化，数字化后可以用来计算距离、相似性等；可从时间型数据中提取更多的时间特征，例如年、月、日等，这些特征对于业务场景以及模型的预测往往有很大帮助。统计型特征处理有利于从业务场景中挖掘更丰富的信息。

如果缺失值多余30%的数据集，我们可以先用目标变量检查他们的分布，如果发现任何模式，我们将保留这些缺失值并给一个新的分类，同时删除其他缺失值，如果没有模式，就直接删除。

1.8.2 Feature selection

特征选择是一个重要的数据预处理的过程，主要原因有两个：减少特征数量、降维，去除共线性，使模型泛化能力加强，减少过拟合；增强对特征和特征值之间的理解。主要方法有以下：

1）Filter

利用方差、皮尔逊相关系数、卡方检验、互信息等方法过滤特征，衡量特征与目标，以及特征与特征值之间的关系

2）正则化。L1正则化可以生成稀疏模型，对应权值为零的特征可以删除

3）随机森林。随机森林和GBDT有自带的特征选择能力

如何去除多重共线性？

在处理多元线性回归问题时，变量间由于存在高度相关性而使回归估计还不准确，造成冗余，导致过拟合，故而需要去除共线性。一般有如下方法：

1）相关性分析：创建一个皮尔森相关系数矩阵，用来识别和去除那些具有75%（阈值主观决定）以上的相关性的变量，它不会因为特征值加或者减去一个数变化。此系数只能检验线性相关，不能检验非线性先关

2）利用方差膨胀因子VIF来检查多重共线性的存在。$VIF = 1/(1-R_i^2)$，其中R_i为自变量x_i对其余自变量作回归分析的负相关系数。VIF<=4表明没有多重共线性，VIF>=10意味着严重的多重共线性。

在变量数不多时，样本数不是很大时，用上述方法可行，但是变量数达到上千时，VIF的计算需要建立上千个回归模型，将耗费很长时间，这时我们可以从模型角度直接规避共线问题。

3）PCA等降维法

4）L1、L2 正则项：以损失部分信息、降低精度为代价获得回归系数更为符合实际、更可靠的回归方法，随着惩罚项系数的增大，共线性的影响会越来越小。

1.8.3 Dimensionality reduction

通过PCA或者LDA，将较高纬度的样本空间映射到较低维度的样本空间，从而达到降维的目的，减少模型的训练时间，提高模型的计算性能。

PCA - Principal components analysis（主成分分析法）

假设我们的数据有两个维度：x, y。那么它的协方差矩阵如下：

$$\begin{bmatrix} \sigma_x^2 & \sigma(x,y) \\ \sigma(x,y) & \sigma_y^2 \end{bmatrix}$$

其中，σ_x^2 是维度x的变异，σ_y^2 是维度y的变异，$\sigma(x,y)$ 是维度x和y之间的协方差，代表x和y之间的共变程度或相关性。

$$\sigma_x^2 = \frac{1}{n-1}\sum_{i-1}^{n}(x_i - \bar{x})^2$$

$$\sigma_y^2 = \frac{1}{n-1}\sum_{i-1}^{n}(y_i - \bar{y})^2$$

$$\sigma(x,y) = \frac{1}{n-1}\sum_{i-1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

SVD是PCA降维的求解方法， 若将数据转换成零均值，SVD和PCA将会有一样的投射。

数据的全部信息包含在协方差矩阵所描述的全部变异中。其中 σ_α^2 代表数据点在维度x上投影点的分散程度，也就数据点在x维度的方差，σ_y^2 代表数据点在维度y上投影点的分散程度；方差越大，数据越分散，也就意味着信息量越大，信号越强，该特征越有区分度。协方差代表维度x和维度y之间的相关程度，协方差越大，噪声越大，信息的冗余程度越高。

PCA的主要目的是从现有的特征中重建新的特征，新的特征剔除原有特征的冗余信息，因此更有区分度，也即是最大化方差；而新的特征基于原有特征，它能够重建原有特征，新特征之间的相关性要尽量小，即最小化协方差（组内差异小组间差异大），主成分必须是正交的，并且归一化的（模长尾1）。

PCA的一大好处是无参数限制，在计算中不需要人为设定参数或者根据任何经验对模型计算进行干预，最后的结果只与数据有关，与用户独立，但这同时也是缺点，如果用户对观测对象有一定的先验知识，掌握了数据的一些特征，却无法通过参数化等方法对处理过程进行干预，可能会得不到预期的效果。

在PCA中，旋转变换是有必要的，因为它把由主成分捕获的方差之间的差异最大化，使得主成分更容易解释。

LDA（Linear Discriminant Analysis 线性判别法）

也叫Fisher判别分析。它的基本思想是：在训练时，将训练样本投影到某条直线上，这条直线使得同类样本的投影点尽可能接近，异类样本投影点尽可能远离。在预测时，将待预测数投影到训练时学习到的直线上，根据投影点的位置来判断所属于的类别。LDA将而为特征向量投射到一维空间，推广到多分类任务中，LDA可以将M维特征向量投影到M-1维空间。

LDA的缺陷在于：

1）当样本数量远小于样本的特征维数，样本与样本间的距离变大使得距离度量实效，使LDA算法中的类内、类间离散度矩阵奇异，不能得到最优的投影方向，在人脸识别领域中表现尤为突出。

2）LDA不适合对非高斯分布的样本进行降维

3）LDA可能过拟合数据

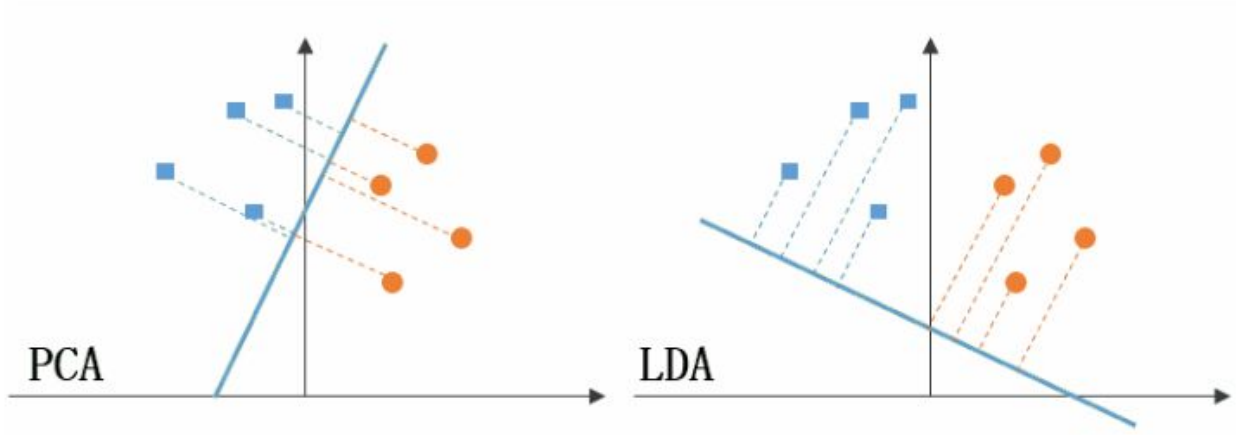
4）LDA在样本分类信息依赖方差而不是均值的时候，降维效果不好，因为LDA需要保证不同类别的数据投影中心尽可能远。

PCA vs LDA

1）出发思想不同。PCA主要是从特征的协方差角度，去找到比较好的投影方式，即选择样本点投影具有最大方差的方向；而LDA则更多的是为了考虑分类标签信息，寻求投影后不同类别之间数据点距离更大化以及同一类别数据点距离最小化，即选择分类性能最好的方向。

2）学习模式不同。PCA是无监督学习，因此大多数场景只作为数据处理的一部分，需要与其他算法结合使用，如将PCA与聚类、判别分析、回归分析等组合使用；LDA是监督学习，本身除了可以降维还可以进行预测，即可以与其他模型一起使用，还可以独立使用。

3）降维后可用维度不同。LDA降维后最多生成C-1维子空间（分类标签数-1），因此LDA与原始维度数量N无关，只与数据标签分类数量有关；而PCA最多有n维度可用，即最大可以选择全部可用维度。



如上图，PCA所做的是将整组数据整体映射到最方便表示这组数据的坐标轴上，映射时没有利用任何数据内部的分类信息。因此，虽然PCA后的数据在表示上更加方便（降低了维数并能最大限度的保持原有信息），但在分类上也许会变得更加困难；而LDA充分利用了数据的分类信息，将两组数据映射到另一个坐标轴上，使得数据更容易区分，减少了运算量)

LDA vs LR

两者都是线性分类器，但LDA的优势在于：

1）当类别的区分度高的时候，LR的参数估计不够稳定，这点在LDA中不存在

2）如果样本量n比较小，而且每一类因变量Y中自变量X近似服从正态分布，那么LDA比LR更稳定

3）多分类问题，LDA效果更好

t-SNE

t-SNE是非线性的降维算法，可以实现高维数据在低维空间的可视化映射，学习的是非参数映射，但是涉及大量的条件概率、梯度下降等计算，时间和空间的复杂度是平方级的，比较耗资源。t-SNE几乎可以用于所有高维数据集，广泛用于图像处理、NLP、基因组数据和语音处理。可以将word2vec的词向量进行可视化。

1.9 Common optimizers

给定一个具有参数 θ 的目标函数，我们要找到一个 θ 是的目标函数取得最大值或最小值。优化算法就是帮助我们找到这个 θ 的算法。

从优化方法上分为：

1）批量梯度下降（BGD）

优点：可以一定程度上解决最优解的问题

缺点：每次迭代使用全部数据集进行训练，运行速度慢，内存可能不够；选择合适的学习率比较难

2）随机梯度下降（SGD）

优点：每次使用一个数据进行训练，训练速度快，无内存问题

缺点：容易震荡，可能达不到最优解

3）Mini-batch梯度下降

优点：综合了批量梯度下降和随机梯度下降的优缺点，提取的一个中和方法

梯度下降法的共同缺点

$$\theta=\theta-\eta\cdot\nabla J(\theta)$$

η 为 learning rate

∇J(θ) 为 代价函数J(θ)对参数θ的梯度

1）对学习率的设置很敏感，太小则训练的太慢，太大则容易使目标函数发散掉

2）针对不同的参数，学习率都是一样的。这对于稀疏数据尤为不便，因为我们更想对那些经常出现的数据采用较小的步长，而对于较罕见的数据采用更大的步长。

3）梯度下降的本质是寻找不动点（目标函数对参数的一阶导为0的点），而这种不动点主要是极大值、极小值、鞍点。高维非凸空间存在大量鞍点，使得梯度下降法极其容易陷入鞍点，且长时间出不来。

4）Momentum

Momentum主要用来解决SGD的参数高幅震荡的问题，为梯度下降法添加了短期记忆。加速参数在主要方向的变化，减弱参数在非主要方向的变化。参数更新方式：

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

和共轭梯度法的作用类似，通过使用历史搜索方法对当前梯度方向的修正来抵消在非主要方向上的来回震荡。

缺点：下坡的过程中动量越来越大，在最低点的速度太大，可能又冲上坡导致错过极小点。

5）Nesterov accelerated gradient (NAG）

对Momentum算法进行的改进。给算法增加了预见能力，事先估计出下一个参数处的梯度，用于对当前计算的梯度进行校正。避免前进太快，提高灵敏度。参数更新方式：

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

6）Adagrad

首次出现了adaptive learning rate adjustment。也就是不同的参数有不同的学习率。梯度大的参数步长小一些，梯度小的参数步长大一些，适合稀疏数据。

缺点：依赖于全局学习率。并且随着时间的增长，学习率越来越小，学习速度越来越慢

7）RMSProp

优点：解决了Adagrad后期学习速度过慢的问题

缺点：依赖于全局学习率

8）Adam

以上方法都只用到了梯度的平方和信息，Adam不仅用了梯度的平方和，也利用了梯度和。结合了Adagrad善于处理稀疏梯度和RMSProp善于处理非平稳目标的优点；为不同的参数计算不同的自适应学习率；也适用于大多非凸优化-适用于大数据集和高维空间。

1.10 Hyperparameter tuning

1.10.1 Grid search

通过查找搜索范围内的所有点，来确定最优值。一般通过给出较大的搜索范围以及较小的步长，但是，它非常消耗计算资源，尤其是需要调优的超参数比较多的时候，时间复杂度是指数级的。此外，由于给出的超参数组合比较多，因此一般会固定多数参数，分步对1-2个参数进行调解，这样能减少时间但是难以自动化运行，而且由于目标和参数一般为非凸，因此容易陷入局部最小值。

1.10.2 Random search

与网格搜索相比，随机搜索并未尝试所有参数值，而是从指定的分布中采样固定数量的参数设置。它的理论依据是：如果随机样本数据集足够大，那么也可以找到全局最优解，或是他们的近似值。通过对搜索范围随机取样，随机搜索一般比网格搜索更快，但是和网格搜索一样，结果无法保证。它的使用方法和网格搜索完全一致。

1.10.3 Bayesian optimization

以上两种方法能针对单独超参数组合模型进行训练，并评估各自的性能，每个模型都是独立的，因此易于进行并行计算，但是每个模型独立也代表着模型之间不具有指导意义，前一模型的计算结构并不能影响后一模型的超参选择。
而贝叶斯优化法的思想就是参考了上一次参数的信息，从而更好的调整当前的参数。其先给定优化的目标函数，通过不断添加样本点来更新目标函数的后验分布，是个高斯过程，直到后验分布基本贴合于真实分布。

它与常规的网格搜索或随机搜索的区别是：
1）贝叶斯调参采用高斯过程，考虑之前的参数信息，不断地更新先验；而网格搜索未考虑之前的参数信息
2）贝叶斯调参迭代次数少，速度快；而网格搜索速度慢，参数多时容易导致维度爆炸
3）贝叶斯调参针对非凸问题依然稳健；而网格搜索针对非凸问题容易得到局部最优解。

当训练一个时间复杂度不高的模型时，可以直接用网格搜索或是随机搜索，但是如果是训练一个成本很高的模型，比如一个需要花几天时间训练的深度神经网络模型，就不适合用网格搜索了，而是用贝叶斯优化比较好。

高斯过程

高斯过程是观测值出现在一个连续域（如时间或空间）的统计模型。在高斯过程中，连续输入空间中每个点都与一个正态分布的随机变量相关联，此外，这些随机变量的每个有限集合都有一个多元正态分布。高斯分布是所有那些（无限多个）随机变量的联合分布，正因如此，它是连续域的分布。一个高斯过程可以被mean和covariance function共同唯一决定。我们知道一个高斯分布可以被mean和variance共同唯一决定，一个多元高斯分布可以被mean和covariance matrix共同唯一决定。

Mean function决定样本出现的整体位置，即基准线，若是0就是在y=0的基准线，ML中在数据预处理时归零中心是常常必做的。而covariance function就是ML中的核函数，它捕捉了不同输入点之间的关系，并反映在之后样本的位置上，这样就可以利用点与点之间的关系，以从输入的训练数据预测未知点的值。不同的核函数光滑性、严格周期性等不一样。

1.10.4 Cross validation

在用以上方法确定了最佳参数后，再根据实际的模型在验证集上的表现做一些微调，在树模型中，对于过拟合优先调整树深和树的数量，对于欠拟合相反。

1.11 Assumptions in Linear Regression

线性回归模型 $Y = a + bX + \mu$，其中X为解释变量，Y为被解释变量，a和b为参数，μ为随机误差项。

线性回归基本假设：

1）随机误差项期望值或平均值为0（期望是平均值随样本趋于无穷的极限）。

2）随机误差项服从正态分布

3）随机误差项彼此不相关

4）对于解释变量的所有观测值，随机误差项有相同的方差

5）解释变量是确定性变量，不是随机变量，与随机误差项彼此之间相互独立

6）解释变量之间不存在完全线性关系。

1.12 Product development pipeline

1) 问题的定义：在工业界中，通常是用botton-up或者top-down的方式明确问题，top-down指由产品经理提出需求，去找团队完成需求，由团队针对这个项目做POC (Proof of concept)，在此基础上对真实用户进行A/B 测试，如果效果不错就转化为产品特征；Bottom-up是你有一个创新点，然后做实验进行验证，再设计成小demo给产品经理看，产品经理觉得OK就再去做UX的优化，最终形成产品特征。

2) 数据处理：数据挖掘和数据处理

3) 模型选择：前期调研，寻找同类问题用到的主流模型，或是直接使用通用模型，比如NER用bi-LSTM+CRF, 文本分类用fasttext，或者都直接使用BERT。选择比较高效的模型。先做出一个baseline model，评价此model是否可行，然后再进行微调。微调主要分为两部分：对模型结构上的微调，比如保留一个大模型的前面若干层，对最后1-2层进行微调；在模型的基础上加一些人工规则。

4）评价指标：工业界的评价指标主要是和用户行为相关的评价方式，如展示位点击转化率CTR、用户的总观看时间等。

- **Academic Research vs. Product Development**

	Academic Research	Product Development
Problem definition	clear, unambiguous	vague, open, cross-domain
Data	clean, formatted	dirty
Model	plentiful, innovative, elegant	efficient, scenario-specific
Metrics	fixed, short cycled	product-oriented, long cycled
Engineering	one-time develop	research pipeline

2. Logistic Regression

2.1 Hypothesis Function

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid函数将所有实数投影到0-1的区间，符合二分类问题的要求。函数在某点的输出为给定x和已知参数，y为positive时候的概率。

Decision Boundary

$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

2.2 Cost Function

如果用linear regression 的损失函数，带入sigmoid function后得到的损失函数不是凸函数，会存在多个local mininum。所以要找到一个凸函数，保证能够找到global minimun。

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

可以简化为：

也叫cross-entropy loss， 交叉熵。

损失函数L(Y， P(Y|X))表达的是样本X在分类Y的情况下，使概率P(Y|X)达到最大值（就是利用已知样本分布，找到最有可能导致这种分布的参数值，或者说什么样的参数才能使我们观测到目前这组数据的概率最大化）。由于log函数是单调递增的，所以logP(Y|X)也会达到最大值，因此在前面加上负号后，最大化P(Y|X)就等于最小化L了。

Cross-entropy loss vs 似然函数

1）区别：Cross-entropy loss是损失函数，代表真实值与预测值的差别，越大代表越不相近，而似然函数本质是衡量在某个参数下，整体的估计和真实情况一样的概率，越大代表越相近。

2）联系：交叉熵函数可以由最大似然函数在伯努利分布的条件下推导出来，或者说最小化交叉熵函数就是最大化对数似然函数。

2.3 Gradient Descent

2.3.1 Batch Gradient Descent

可以通过GD对损失函数进行优化。不断对参数同时进行以下迭代，直至收敛，即达到了最优解。参数迭代的公式和Linear Regression一样，但要注意其中的hypothesis function是不同的。

$$\begin{aligned} &Repeat\ \{ \\ &\quad \theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \\ &\} \end{aligned}$$

有一些更高级的优化算法，比如共轭梯度，BFGS，和 L-BFGS，他们比GD跟复杂，但是不需要设置学习率，且更快，所以在进行大规模数据训练的时候，建议使用他们。

通过GD优化cost function的过程中，要同时更新参数。如果cost function是个凸函数，那么优化过程永远可以达到全局最小值，如果不是凸函数，只能找到local optimal。

为了加快梯度下降算法收敛的速度，可以将每个特征值归一化到近似的范围内，这是因为参数会在小范围内快速下降，而在大范围内缓慢下降，因此当变量非常不均匀时，会无效率地振荡到最优解。一般采取特征缩放和mean normalization的方式。特征缩放直接将输入值都除以他们的范围，mean noralization要先减去平均值。

$$x_i := \frac{x_i - \mu_i}{s_i}$$

对于线性回归算法，计算参数还有另一个方法： Normal Equation。它可以利用数学的方法不需要通过迭代一次性算出参数值，公式为 $\theta = (X^T X)^{-1} X^T y$ 。并且不需要进行特征缩放。但是它在分类算法中无法使用。它和GD的区别如下：

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

如果 $X^T X$ 不可逆，一般有两个原因：1.存在冗余特征值，即两个特征值高度相关；2.特征值太多。建议检查特征值，删除部分和目标相关度低的特征值，或者使用正则化。

梯度下降法的问题和挑战：

1）梯度的计算。在机器学习和参数统计估计问题中目标函数经常是求和函数的形式，当样本量极大时，梯度的计算就变得非常耗时耗力。

2）学习率的选择：如果学习率很小，收敛速度会很慢，但是可以收敛到最优解。如果学习率很大，很可能错过最优解，并且随着迭代次数的增加，损失函数的值反而变大。通常设置可调的学习率，先大加快收敛，后小使函数收敛到最优解。

针对第一个问题，提出了SGD和Mini-batch GD的优化方法。

2.3.2 Stochastic Gradient Descent

BGD在更新回归系数的时候需要遍历整个数组，是一种批处理的方法，这样训练数据非常庞大时，会出现如下问题：收敛速度非常慢，尤其到了接近最优解的时候；如果误差曲面上有多个局部最小值，不能保证收敛过程会到达全局最小值。

为了解决以上问题，我们一般会采用SGD，根据单独的训练样本来更新权值。对比一下BGD和SGD的参数更新过程：

BGD：

参数更新为：$$\theta_j^{'} = \theta_j + \frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$$

其中，i是样本编号下标，j是样本维数下标，m是样本数目，n是特征数目。所以更新一个参数θj需要遍历整个样本集。

SGD：

$$\theta_j^{'} = \theta_j + (y^i - h_\theta(x^i))x_j^i$$

在SGD中，更新一个θj只需要一个样本就可以。

2.3.3 Mini Batch Gradient Descent

每次梯度计算使用一个小批量样本，梯度样本相对SGD更为稳定，可以很好地利用县城的高度优化的矩阵运算工具。

注意，梯度下降法并不一定是全局下降最快的方向，它只是目标函数在当前点的切平面上下降最快的方向，而牛顿法（运用了泰勒展开二阶）才一般是下降最快的方向。

梯度下降和牛顿法的区别：

1）从收敛速度看，牛顿法是二阶收敛，梯度下降是一阶收敛，前者收敛速度更快。但牛顿法仍然是局部算法，只是在局部上看得更细致，梯度法仅考虑方向，牛顿法不但考虑了方向还兼顾了步子的大小，其对步长的估计使用的是二阶逼近。

2）牛顿法每一步都需要求解目标函数的Hessian矩阵（多元函数的二阶偏导数构成方阵，描述了函数的局部曲率）的逆矩阵，计算比较复杂，计算和存储都是问题。在使用小批量的情形下，牛顿法对二阶导数的估计噪音太大，在目标函数非凸时，牛顿法更容易收敛到鞍点。

拟牛顿法的本质思想是改善牛顿法每次需要求解复杂的Hessian矩阵的逆矩阵的缺陷，它使用正定矩阵来近似Hessian矩阵的逆，从而简化了算法的复杂度。拟牛顿法不需要二阶导数的信息，所以有时候比牛顿法更有效。常用的拟牛顿法有BFGS算法和DFP算法。

共轭梯度法是介于梯度下降法与牛顿法之间的一个方法，它仅需要利用一阶导数，但克服了梯度下降法收敛速度慢的缺点，又避免了牛顿法需要存储和计算海森矩阵并求逆的缺点。它的基本思想是把共轭性与梯度下降法结合，利用已知点的梯度构造一组共轭方向，并沿这组方向进行搜索，求出目标函数的极小值。

2.4 Multiclass classification

One vs rest algorithm: 假设我们的target有N个类别的，我们可以把分类问题看成N个二分类问题，通过LR找到N个二分类器，将每对类别的target分开。在对新的数据进行预测的时候，将其归于能够最大化假设函数的类别。

$$
\begin{aligned}
&y \in \{0, 1 \ldots n\} \\
&h_\theta^{(0)}(x) = P(y = 0|x; \theta) \\
&h_\theta^{(1)}(x) = P(y = 1|x; \theta) \\
&\cdots \\
&h_\theta^{(n)}(x) = P(y = n|x; \theta) \\
&\text{prediction} = \max_i(h_\theta^{(i)}(x))
\end{aligned}
$$

这同样可以用于SVM的多分类问题。

2.5 Overfitting and Underfitting

欠拟合是指模型没有学到数据内在关系，不能很好的预测目标。

过拟合即模型过度拟合了训练数据的内在关系，能够很好的预测训练集，但是在测试集上表现很差。

2.5.1 Bias and Variance

Bias偏差度量了学习算法的期望预测和真实结果的偏离程度，刻画了学习算法本身的拟合能力。
Variance方差度量了同样大小的训练集的变动所导致的学习性能的变化，刻画了数据扰动所产生的影响，度量学习算法的泛化能力。

2.5.2 Cross Validation

将数据分成三份：训练集、交叉验证集和测试集。交叉验证集在模型选择、超参数选择、正则项参数选择和评价模型中都很有用。

在分类问题中，最好采用分层抽样将数据集分开，这样可以保证类别的比例。如果数据集足够大，采用随机抽样分成子集也可以。但如果数据量不够大，随机抽样很可能导致数据偏差，影响预测准确性。

1）简单交叉验证

利用训练集训练模型；

利用交叉验证集在每一种模型上测试并计算损失值

选择损失值小的模型

但是，在增加CV set后，训练集的数据量更少了，这样模型无法更好的体现整体数据，所以可以用K-Fold CV来解决。

2）K-Fold交叉验证

假设训练集为S，将训练集分为K份: {S1, S2, …...Sk}

然后每次从集合中拿出K-1份进行训练

利用集合中剩下的那份来进行测试并计算损失值

最后得到k次测试得到的损失值，并选择平均损失值最小的模型

3）LOOCV

Leave one out cross validation

只用一个数据作为测试集，其他都是训练集用于训练模型和调参，最终训练N个模型（N为总的数据量），每次得到一个MSE，最终test MSE 是将这个N个MSE取平均。它不受测试集合训练划分方法的影响，因为每个数据都单独做过测试集，同时，用了N-1个数据训练模型，也几乎用到了所有数据，故而偏差很小。不过缺点在于计算量巨大。

LOOCV是一种特殊的K-fold，其中K=N

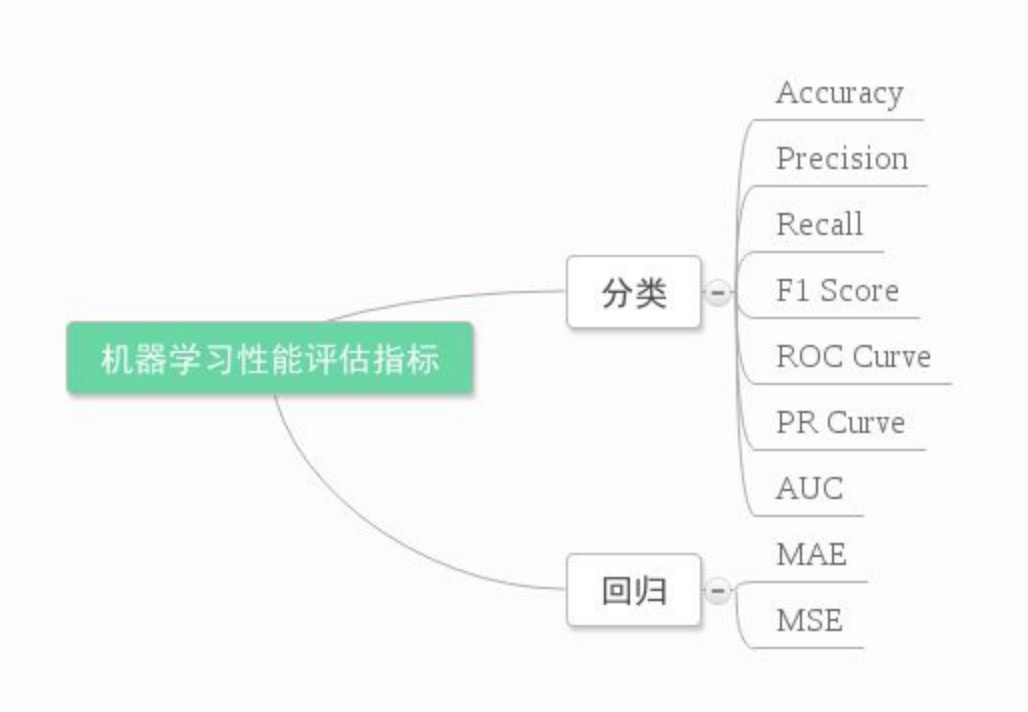
Cross Validation in Time-Series data

对于时间序列问题，无法使用K-fold或是LOOCV，因为第四年或第五年的模式可能和第三年不同而对数据集的重复采样会将分离这些趋势，我们最终可能是对过去几年进行了验证，相反，我们可以采用如下的五倍正向链接策略：

```
fold 1 : training [1], test [2]
fold 2 : training [1 2], test [3]
fold 3 : training [1 2 3], test [4]
fold 4 : training [1 2 3 4], test [5]
fold 5 : training [1 2 3 4 5], test [6]
```

1，2，3，4，5，6代表的是年份。

2.5.3 Evaluation metrics



为何不常用accuracy衡量分类模型？

因为很多模型对分类问题的预测结果都是概率，如果要计算accuracy，需要先把概率转化为类别，这就需要手动设置阈值，如果对一个样本的预测概率高于这个阈值，就把这个样本放进一个类别里面，低于这个阈值，就放进另一个类别里，所以阈值的大小很大程度影响了accuracy的计算，而是用AUC或者logloss就可以避免这个问题。

另外，针对不平衡样本的问题，准确率不可靠，我们可以采用以下几种指标：

精确率(precision)定义为：

$$P = \frac{TP}{TP + FP}$$

表示被分为正例的示例中实际为正例的比例。

召回率Recall是覆盖面的度量，度量多少个正例被分为正例。Recall = TP / (TP+FN) = TP/P = Sensitive

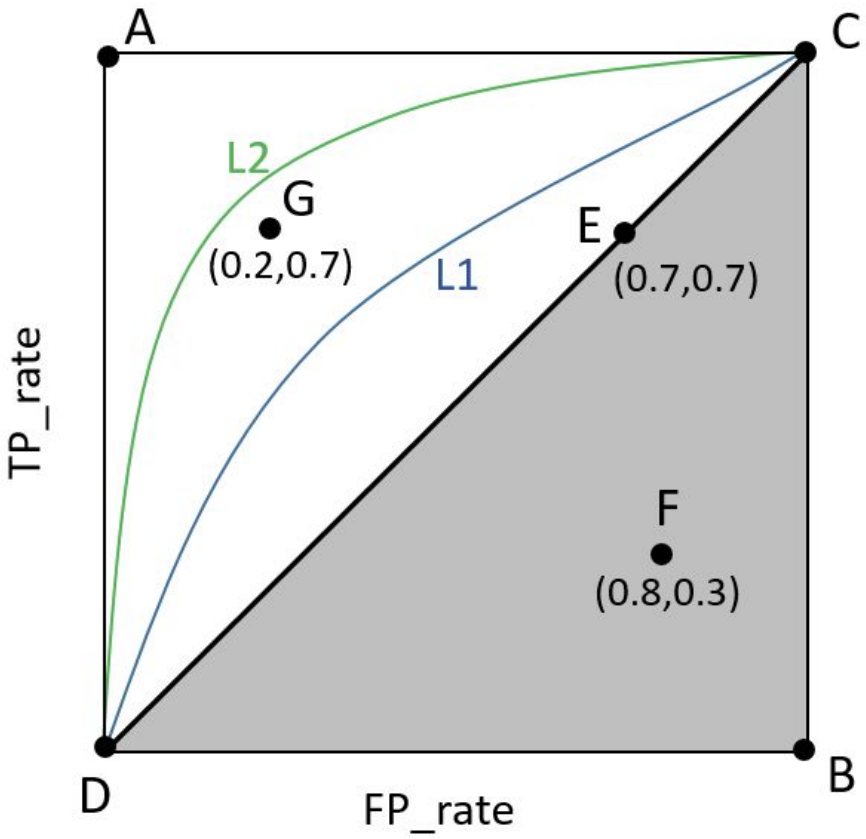
在二分类问题中，提高阈值，会导致预测为正的样本数降低，导致recall的分子变小，分母不变，召回率降低，而Precision的分母分子都减小，精确率变化不确定。

F1 score是Recall和Precision的加权调和平均：F1 = 2*P*R/(P+R)，F1的值在0-1之间，越高说明试验方法比较有效。

ROC曲线是以FP rate和TP rate为轴的曲线，ROC曲线下的面积叫做AUC。

第一类和第二类错误

在混淆矩阵中，如果我们把一个值归为阳性，但事实它为阴性，发生第一类错误(假阳性FP)；而当我们把一个值归为阴性，但它事实为阳性时，就发生第二类错误（假阴性FN)。



其中: $TP_{rate} = \frac{TP}{P_c}, FP_rate = \frac{FP}{N_c}$

（1）曲线与FP_rate轴围成的面积（记作AUC）越大，说明性能越好，即图上L2曲线对应的性能优于曲线L1对应的性能。即：曲线越靠近A点（左上方）性能越好，曲线越靠近B点（右下方）曲线性能越差。

（2）A点是最完美的performance点，B处是性能最差点。

（3）位于C-D线上的点说明算法性能和random猜测是一样的–如C、D、E点。位于C-D之上（即曲线位于白色的三角形内）说明算法性能优于随机猜测–如G点，位于C-D之下（即曲线位于灰色的三角形内）说明算法性能差于随机猜测–如F点。

（4）虽然ROC曲线相比较于Precision和Recall等衡量指标更加合理，但是其在高不平衡数据条件下的的表现仍然过于理想，不能够很好的展示实际情况。

AUC的几何意义与源码：

AUC的几何意义是ROC曲线下的面积，其物理意义在于：给定正样本M个， 负样本N个，以及他们的预测概率，那么AUC的含义就是穷举所有的正负样本对，如果正样本的预测概率大于负样本的预测概率，那么+1， 如果相等，+0.5，如果正样本的预测概率小于负样本，就+0，最后把统计的值除以M*N就得到AUC。

```
[7] def AUC(label, pre):
      pos = [i for i in range(len(label)) if label[i] == 1]
      neg = [j for j in range(len(label))if label[j] == 0]

      auc = 0
      for i in pos:
        for j in neg:
          if pre[i] > pre[j]:
            auc += 1
          elif pre[i] == pre[j]:
            auc += 0.5
      return auc/(len(pos)*len(neg))

    if __name__ == '__main__':
      label = [1,1,1,0,0,0,1]
      pre = [0.6,0.7,0.8,0.3,0.5,0.99,0.67]

      print(AUC(label, pre))
```

```
0.6666666666666666
```

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, th = roc_curve(label, pre,pos_label = 1)
print('sklearn', auc(fpr, tpr))
```

```
sklearn 0.6666666666666667
```

R-squared vs Adjusted R-squared

线性回归问题中，R-squared用来衡量预测值与真实值之间的相似程度，表达式为：

$$R^2=1-\frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$$

上式中，分子表示真实值与预测值的平方差之和，类似于MSE，分母表示真实值与均值的平方差，类似于方差，一般来说，R-squared越大，表示模型拟合度越好。如果增加一个特征，R-squared可能变大也可能保持不变，两者不一定正相关。

Adjusted R-squared

$$R^2\_\text{adjusted} = 1 - \frac{(1-R^2)(n-1)}{n-p-1}$$

其中，n是样本数，p是特征数量。Adjusted R-squared抵消了样本数量对R-squared的影响，做到了真正的0-1。如果增加一个特征变量，若这个特征有意义，Adjusted R-squared就会增大，若是冗余特征，Adjusted R-squared就会减少。

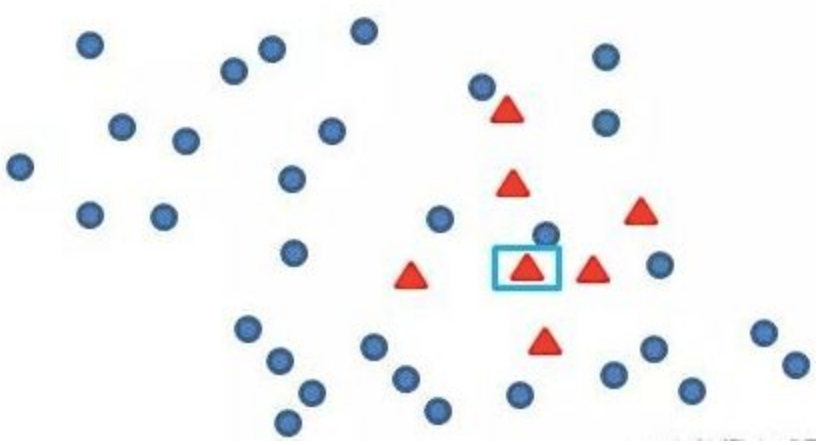
Imbalanced data

1) 过采样。对训练集里样本数量较少的类别进行过采样，合成新的样本来缓解类不平衡。

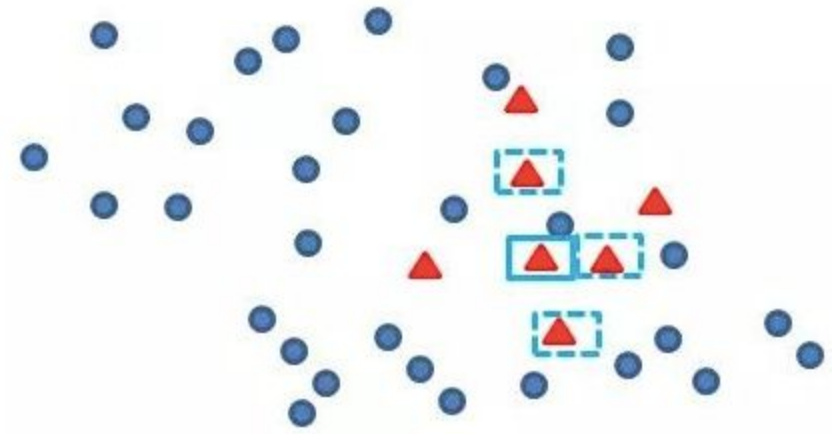
SMOTE为一经典的过采样算法。它是基于随机过采样算法的一种改进方案，由于随机过采样采取简单复制样本的策略来增加少数类样本，这样容易产生模型过拟合的问题，即使得模型学习到的信息过于特别(Specific)而不够泛化(General)，SMOTE算法的基本思想是对少数类样本进行分析并根据少数类样本人工合成新样本添加到数据集中，采用了KNN技术。模拟生成新样本的步骤如下：采样KNN算法，算出每个少数类样本的K个近邻；从K个近邻中随机挑选出N个样本进行随机线性插值；构造心的少数类样本；将新样本与原数据合成，产生新的训练集。

SMOTE的思想可以用如下的图解表示：

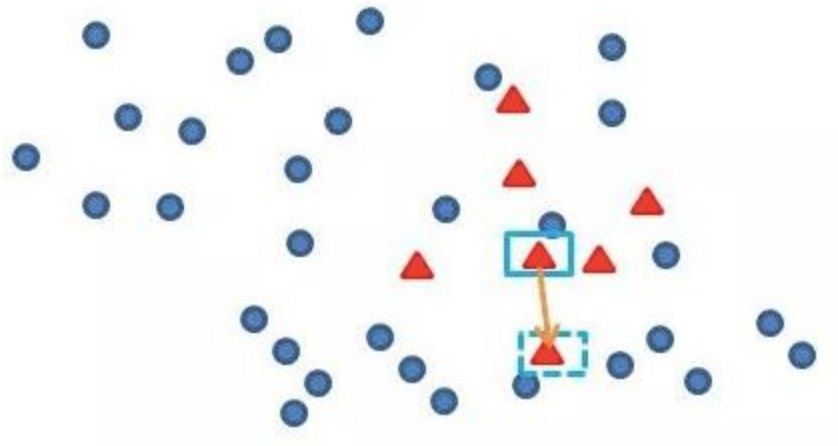
1.首先随机选定n个少类的样本，找出初始扩展的少类样本。



2.找出最靠近它的m个少类样本。



3.再任选最邻近的m个少类样本中的任意一点，在这两点上任选一点，这点就是新增的数据样本。



2）欠采样。对训练集里样本数量较多的类别进行欠采样，抛弃一些样本来缓解类不平衡，但这会导致某些隐含的信息被丢失。

3）进行特殊的加权，如在Adaboost或者SVM中

4）改变评价指标，用AUC/ROC进行评价

5）采用bagging/boosting等集成方法

2.5.4 Address

问题	数据	特征	模型
Underfitting	清洗数据	1. 增加特征 2. 删除噪音特征	1. 调低正则项的惩罚参数 2. 换更“复杂”的模型（如把线性模型换为非线性模型） 3. 多个模型级联或组合
Overfitting	增加数据	1. 进行特征选择 2. 降维（如对特征进行聚类、主题模型进行处理等）	1. 提高正则项的惩罚参数 2. 减少训练迭代次数 3. 换更“简单”的模型（如把非线性模型换为线性模型）

Regularization

正则化可以看做是损失函数的惩罚项。所谓惩罚是对损失函数中的某些参数做一些限制，以简化复杂模型。

L1 Norm

权值向量中各个元素的绝对值之和。Lasso回归。可以用来产生稀疏权值矩阵，用于特征选择。

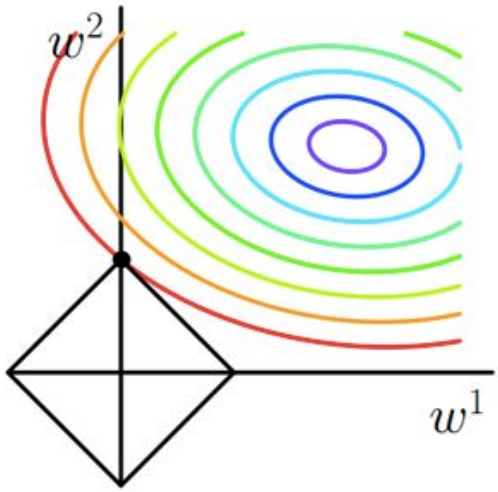
如果机器学习中特征数量很多，在预测或者分类时，那么多特征难以选择，如果带入这些特征得到的模型是个稀疏模型，表示只有少数特征对模型有贡献，绝大多数没有贡献或是贡献微小（因为他们前面的系数是0或者很小，就算去掉也没什么影响），所以我们可以只关注系数是非零值的特征。

那么为何L1正则化可以产生稀疏矩阵呢？

$$J = J_0 + \alpha \sum_w |w|$$

假设有如下带L1正则化的损失函数：

其中J_0是原始的损失函数，加号后面的一项是L1正则化项，α是正则化系数。注意到L1正则化是权值的**绝对值之和**，J是带有绝对值符号的函数，因此J是不完全可微的。机器学习的任务就是要通过一些方法（比如梯度下降）求出损失函数的最小值。当我们在原始损失函数J_0后添加L1正则化项时，相当于对J_0做了一个约束。令$L = \alpha \sum_w |w|$，则$J = J_0 + L$，此时我们的任务变成**在L约束下求出J_0取最小值的解。考虑二维的情况**，即只有两个权值w^1和w^2，此时$L = |w^1| + |w^2|$。对于梯度下降法，求解J_0的过程可以画出等值线，同时L1正则化的函数L也可以在w^1w^2的二维平面上画出来。如下图：



在图中，当J_0等值线与L图形首次相交的地方就是最优解。上图中J_0与L在L的一个顶点处相交，这个顶点就是最优解。注意到这个顶点的值是$(w^1, w^2) = (0, w)$。可以直观想象，因为L函数有很多『突出的角』（二维情况下四个，多维情况下更多），J_0与这些角接触的机率会远大于与L其它部位接触的机率（这是很直觉的想象，突出的角比直线的边离等值线更近写），而在这些角上，会有很多权值等于0（因为角就在坐标轴上），这就是为什么L1正则化可以产生稀疏模型，进而可以用于特征选择。

而正则化前面的系数α，可以控制L图形的大小。α越小，L的图形越大（上图中的黑色方框）；α越大，L的图形就越小，可以小到黑色方框只超出原点范围一点点，这是最优点的值$(w1, w2) = (0, w)$中的w可以取到很小的值。

L2 Norm

Ridge回归。用来防止过拟合。

$$l_2 : \Omega(w) = ||w||_2^2 = \sum_i w_i^2$$

同样可以画出他们在二维平面上的图形，如下：

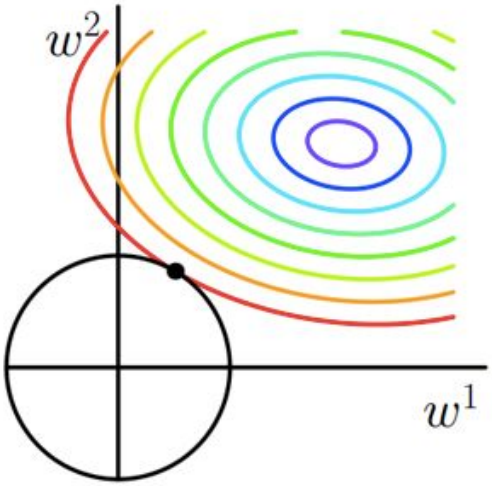


图2 L2正则化

二维平面下L2正则化的函数图形是个圆（绝对值的平方和，是个圆），与方形相比，被磨去了棱角。因此J_0与L相交时使得w^1或w^2等于零的机率小了许多（这个也是一个很直观的想象），这就是为什么L2正则化不具有稀疏性的原因，因为不太可能出现多数w都为0的情况。

从贝叶斯学派的角度看，正则化其实是对模型的参数设定一个先验，L1是laplace先验，L2是高斯先验。先验是优化的起跑线，有先验的好处就是可以在较小的数据集中有良好的泛化能力。L2先验趋向于0周围，而L1先验趋向于0本身。

如何选择L1或者L2回归？

由于L1回归同时做变量选择和参数收缩，而L2回归只做参数收缩，并最终在模型中包含所有的系数。在有相关变量时，L2回归是首选。此外，L2回归在用最小二乘估计有更高的方差的情况下效果最好。注意在LR中同时加入L1和L2并不会提高精度。

2.6 Pros and Cons

优点：

1）输出值并不是一个离散值或是一个类别，而是每个观测样本相关的概率列表，我们可以不同的标准和常用的指标性能分析这个概率分数，并得到一个阈值，从而确定分类。在金融行业，普遍运用于评分卡中。
2）在时间和内存的需求上很高效。
3）对outlier不是很敏感，多重共线问题可以用L2正则化来解决。
4）LR被广泛运用到了工业问题上。
5）简单易于理解，可以直接看到各个特征的权重。
6）LR将各个维度特征的线性组合进行了非线性处理，即使用sigmoid函数将输入值压缩到（0，1）之间，这样避免了不同量纲下某些特征主导模型的现象，所以不需要对数据进行标准化。

缺点：
1）当特征值或是变量很多时，逻辑回归性能不是很好
2）对于非线性特征，需要进行转换
3）难以处理多维分类问题
4）特征处理复杂，需要归一化和较多的特征工程。

2.7 Discretization

在工业界，很少直接将连续值作为逻辑回归模型的特征输入，而是将连续特征离散化为一系列0、1特征交给逻辑回归模型，这样做的优势有以下几点：
1) 离散特征的增加和减少都很容易，易于模型的快速迭代；
2) 稀疏向量内积乘法运算速度快，计算结果方便存储，容易扩展；
3) 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰；
4) 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；
5) 离散化后可以进行特征交叉，由M+N个变量变为M*N个变量，进一步引入非线性，提升表达能力；
6) 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问；
7) 特征离散化以后，起到了简化了逻辑回归模型的作用，降低了模型过拟合的风险。

李沐曾经说过：模型是使用离散特征还是连续特征，其实是一个“海量离散特征+简单模型” 同 “少量连续特征+复杂模型”的权衡。既可以离散化用线性模型，也可以用连续特征加深度学习。就看是喜欢折腾特征还是折腾模型了。

2.8 Parallelization

LR的并行化主要是对目标函数的梯度计算并行化。目标函数的梯度向量计算中只需要进行向量间的点乘和相加，可以将每个迭代过程拆分成相互独立的计算步骤，由不同的节点进行独立计算，然后归并计算结果。

3. Neural Networks

3.1 Perception

感知机是一个二类分类的线性分类器，是支持向量机和神经网络的基础。它假设数据是线性可分的，目标是通过梯度下降法，极小化损失函数，最后找到一个分割超平面，可以将数据划分成两个类别。

决策函数 如下：

$$f(x) = sign(w \cdot x + b)$$

其中 w 是权值（weight）参数，b 是偏置项（bias）。对 n 维来说，线性方程 $w \cdot x + b = 0$ 对应特征空间的一个超平面，其中 w 是超平面的法向量，b 是超平面的截距。$w \cdot x$ 是求内积的意思，即各项对应相乘后求和。
在二维空间，即直角坐标系中，上面的超平面变成了高中学过的直线的解析式。

$$\begin{cases} w = (A,\ B),\ x = (x_1,\ x_2) = (x,\ y) \\ w \cdot x + b = 0 \rightarrow\ Ax + By + b = 0 \end{cases}$$

函数 $sign$ 就是符号函数，定义如下：

$$sign(t) = \begin{cases} +1, & t \geq 0 \\ -1, & t < 0 \end{cases}$$

即，给出特征空间的一个例子$x = \{x_1, x_2, x_3, \ldots, x_n\}$，通过上面的公式得出该输入的分类结果是 $f(x)$。对二类分类来说，要么是正例，要么是反例。

神经网络的强大之处是可以用来解决数据量很大的非线性的预测问题。

3.11 Loss function

0-1损失函数：

预测值和目标值不等时为1，否则为0，是非凸函数。

$$L(Y, f(X)) = \begin{cases} 1, Y \neq f(X) \\ 0, Y = f(X) \end{cases}$$

感知机使用的就是这种损失函数，但是相等这个条件太严格，因此可以放宽条件，即满足|Y-f(x)| < T 时认为相等。

$$L(Y, f(X)) = \begin{cases} 1, |Y - f(X)| \geq T \\ 0, |Y = f(X)| < T \end{cases}$$

目标函数为：

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

3.2 Forward propagation

前向传播就是通过上一层的节点以及对应的连接权值进行加权和运算，最终结果再加上一个偏置项（Bias），最后再通过一个非线性函数，比如Relu，Sigmoid，最后得到的结果就是本层节点的输出。不断通过这一方法一层层的运算，得到输出层结果。

对于前向传播来说，不管维度多高，其过程都可以用如下公式表示：

$$a^2 = \sigma(z^2) = \sigma(a^1 * W^2 + b^2)$$

其中，上标代表层数，星号表示卷积，b表示偏置项bias，σ表示激活函数。

3.3 Backward propagation

DL的模型所学到的信息都是存储在权值当中的。权值通过BP不断进行更新。

目的在于找到参数向量来最小化损失函数，它类似于LR里的GD。

BP的过程：

- 先做FP计算出 $a^{(l)}$ for l=2,3,...,L
- 计算输出层的误差, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
- . Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$

  $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

  ence we update our new Δ matrix.

  - $D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right)$, if j≠0.
  - $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$ If j=0

在实际操作的过程中，反向传播算法很难调试得到正确结果，有可能导致只有部分权重得到训练，可以通过梯度检验（Gradient Checking）的方式对求导结果进行数值检验，以确保求导代码正确。

梯度校验的数值校验公式：

$$g(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

NN的参数不可以全部初始化为0，因为如果都初始化为0，在反向传播的过程中，同一隐藏层的每个节点所学习到的特征是相同的，那么就失去了NN的意义，可以用随机初始化来解决。

在用sigmoid做激活函数的时候，为何用交叉熵损失函数而不是MSE？

这个问题通过求导，再分析两个误差函数的参数更新就可以发现原因。

对于MSE，常常定义为：

$$C = \frac{1}{2n}\sum_x (a-y)^2$$

其中，y是期望的输出，a为神经元的实际输出，在训练神经网络的时候我们用梯度下降的方式来更新w和b，即计算代价函数对w与b的导数，再更新参数，参数更新的公式为：

$$w = w - \eta\frac{\partial C}{\partial w} = w - \eta(a-y)\sigma'(z)x$$
$$b = b - \eta\frac{\partial C}{\partial b} = b - \eta(a-y)\sigma'(z)$$

而由于sigmoid的性质，在z很大的时候，$\sigma'(z)$取值很小，这回导致参数的更新非常慢

而对于cross-entropy loss, 它的定义为:

$$C = -\frac{1}{n}\sum_x [y\ln a + (1-y)\ln(1-a)]$$

参数更新过程同MSE，最终的参数更新公式为：

$$w = w - \eta\frac{\partial C}{\partial w} = w - \eta(a-y)x$$
$$b = b - \eta\frac{\partial C}{\partial b} = b - \eta(a-y)$$

它没有$\sigma'(z)$这一项，权重的更新只受（a-y）的影响，当误差大的时候，权重更新快，误差小的时候，权重更新慢，这是个很好的性质。

3.4 Pros and Cons

优点

1）分类准确率高

2）并行处理能力强

3）分布式存储和学习能力强

4）鲁棒性强，不易受噪声影响

缺点

1）需要大量参数

2）结果难以解释

3）训练时间过长

4. SVM

SVM可以广泛用于实际问题中，包含回归、聚类、分类、手写数字识别等。

做分类算法时，用w^T+b定义分类函数，于是求w、b，为寻最大间隔，引出1/2||w||^2，继而引入拉格朗日因子，化为对拉格朗日乘子a的求解（求解过程中会涉及到一系列最优化或凸二次规划等问题），如此，求w.b与求a等价，而a的求解可以用一种快速学习算法SMO，至于核函数，是为处理非线性情况，若直接映射到高维计算恐维度爆炸，故在低维计算，等效高维表现。在线性可分的情况下，支持向量在间隔边界上，在线性不可分的情况下，支持向量或者在间隔边界尚，或者在间隔边界与分离超平面之间，或者在分离超平面误分一侧。

凸包

当数据是线性可分的，凸包就表示两个组数据点的外边界。

一旦凸包建立，我们得到的最大间隔超平面(MMH)作为两个凸包之间的垂直平分线，MMH是能够最大界限地分开两个组的线。

4.1 Cost Function

Hinge loss

对于输出$y = \pm 1$，当前$\hat{y}$的损失为：

$$\ell(y) = \max(0, 1 - y \cdot \hat{y})$$

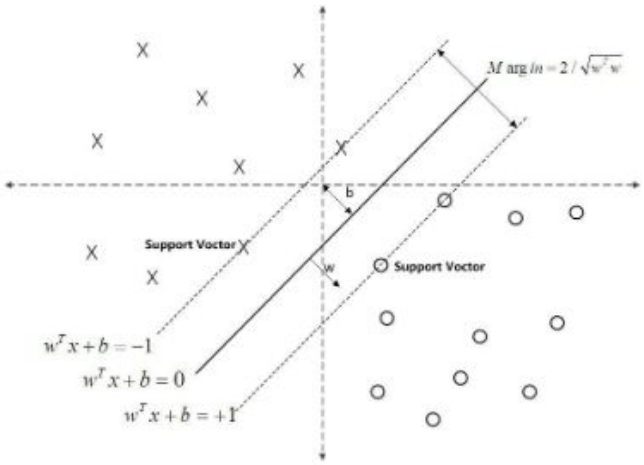
一般$\hat{y}$是预测值，在-1到1之间，y是目标值（-1或1）。其含义是，$\hat{y}$的值在-1和+1之间就可以，并不鼓励|$\hat{y}$|>1，即并不鼓励分类器过度自信，让某个正确分类的样本距离分割线超过1并不会有任何奖励，从而使分类器可以更专注于整体的误差。

经过演算可以得到目标函数为：

$$\min \frac{1}{2}\|w\|^2 \quad s.t., y_i(w^T x_i + b) \geq 1, i = 1, \ldots, n$$

相当于在相应的约束条件 $y_i(w^T x_i + b) \geq 1, i = 1, \ldots, n$ 下，最大化这个1/||w||值，而1/||w||便是几何间隔 $\tilde{\gamma}$。

如下图所示，中间的实线便是寻找到的最优超平面（Optimal Hyper Plane），其到两条虚线边界的距离相等，这个距离便是几何间隔 $\tilde{\gamma}$，两条虚线间隔边界之间的距离等于 $2\tilde{\gamma}$，而虚线**间隔边界上的点则是支持向量**。由于这些支持向量刚好在虚线间隔边界上，所以它们满足 $y(w^T x + b) = 1$（还记得我们把 functional margin 定为 1 了吗？上节中：处于方便推导和优化的目的，我们可以令 $\hat{\gamma}$ =1），而对于所有不是支持向量的点，则显然有 $y(w^T x + b) > 1$。



SVM中的代价函数参数决定着SVM能够在多大程度上适配训练数据，如果要一个平稳的决策平面，代价会比较低；如果需要将更多的数据正确分类，代价会比较高，可以理解为误分类的代价。Hingeloss的健壮性强，对噪音不敏感。

4.2 Dual SVM

因为现在的目标函数是二次的，约束条件是线性的，所以它是一个凸二次规划问题。这个问题可以用现成的QP (Quadratic Programming) 优化包进行求解。一言以蔽之：在一定的约束条件下，目标最优，损失最小。

此外，由于这个问题的特殊结构，还可以通过拉格朗日对偶性（Lagrange Duality）变换到对偶变量 (dual variable) 的优化问题，即通过求解与原问题等价的对偶问题（dual problem）得到原始问题的最优解，这就是线性可分条件下支持向量机的对偶算法，这样做的优点在于：一者对偶问题往往更容易求解，可以大大降低时间复杂度；二是可以自然的引入核函数，进而推广到非线性分类问题。

4.3 Kernel functions

在分类问题中，如果我们有无限多个特征值，很容易造成过拟合，我们使用SVM的kernel trick解决这个问题。

Kernel的本质两个函数的内积，对于非线性分类问题，SVM的处理方法是选择一个核函数，通过核函数将数据映射到高维空间，在高维空间把非线性问题转化为线性问题，SVM得到的超平面是高维空间的线性分类平面，其分类结果也视为低维空间的非线性分类结果。以此来解决在原始空间中线性不可分的问题。

具体来说，在线性不可分的情况下，支持向量机首先在低维空间中完成计算，然后通过核函数将输入空间映射到高维特征空间，最终在高维特征空间中构造出最优分离超平面，从而把平面上本身不好分的非线性数据分开。

而在我们遇到核函数之前，如果用原始的方法，那么在用线性学习器学习一个非线性关系，需要选择一个非线性特征集，并且将数据写成新的表达形式，这等价于应用一个固定的非线性映射，将数据映射到特征空间，在特征空间中使用线性学习器，因此，考虑的假设集是这种类型的函数：

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b,$$

这里ϕ：X->F是从输入空间到某个特征空间的映射，这意味着建立非线性学习器分为两步：

1. 首先使用一个非线性映射将数据变换到一个特征空间F，
2. 然后在特征空间使用线性学习器分类。

而由于对偶形式就是线性学习器的一个重要性质，这意味着假设可以表达为训练点的线性组合，因此决策规则可以用测试点和训练点的内积来表示：

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i \langle \boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}) \rangle + b.$$

如果有一种方式可以在特征空间中直接计算内积〈φ(x*i*· φ(x)〉，就像在原始输入点的函数中一样，就有可能将两个步骤融合到一起建立一个非线性的学习器，这样直接计算法的方法称为核函数方法：

核是一个函数K，对所有x，z(-X，满足 $K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle,$

$K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle,$ ，这里φ是从X到内积特征空间F的映射。

简单来说，第一种方式是先将样本映射到高维空间中，再根据内积的公式进行运算，但这样会导致映射的维度爆炸，难以计算，而用Kernel，是直接在原来的低维空间中进行运算，将转换和内积的过程一起操作，简化了计算步骤，就算映射到无穷维也没问题。

4.4.1 Linear kernel

安全（一般都先试线性核函数），快，好解释。但是无法处理非线性分类问题。

$$\kappa(x, x_i) = x \cdot x_i$$

线性核，主要用于线性可分的情况，我们可以看到特征空间到输入空间的维度是一样的，其参数少速度快，对于线性可分数据，其分类效果很理想，因此我们通常首先尝试用线性核函数来做分类，看看效果如何，如果不行再换别的

4.4.2 Polynomial kernel

比线性核函数的限制少，比较容易控制，但是难以计算，并且有多个超参数要选择。如果degree很小，才会用多项式核函数，但有时直接用对偶的线性核函数就可以解决。

$$\kappa(x, x_i) = ((x \cdot x_i) + 1)^d$$

多项式核函数可以实现将低维的输入空间映射到高纬的特征空间，但是多项式核函数的参数多，当多项式的阶数比较高的时候，核矩阵的元素值将趋于无穷大或者无穷小，计算复杂度会大到无法计算。

4.4.3 Gaussian kernel

无限多维转换，比前两个都强大，并且只有一个超参数要选择。但是不好解释，并且耗时较长，并可能导致过拟合。

$$\kappa(x, x_i) = exp(-\frac{||x - x_i||^2}{\delta^2})$$

高斯径向基函数是一种局部性强的核函数，其可以将一个样本映射到一个更高维的空间内，该核函数是应用最广的一个，无论大样本还是小样本都有比较好的性能，而且其相对于多项式核函数参数要少，因此大多数情况下在不知道用什么核函数的时候，优先使用高斯核函数。

Gamma是选择RFB函数作为kernel后，该函数自带的参数，隐含地决定了数据映射到新的特征空间后的分布。Gamma越大，支持向量越少；Gamma越小，模型越简单，平滑度越好，分类边界越不容易过拟合。支持向量的个数影响预测与训练的速度。

4.4.4 Sigmoid kernel

$$\kappa(x, x_i) = tanh(\eta < x, x_i > +\theta)$$

采用sigmoid核函数，支持向量机实现的就是一种多层神经网络。

因此，在选用核函数的时候，如果我们对我们的数据有一定的先验知识，就利用先验来选择符合数据分布的核函数；如果不知道的话，通常使用交叉验证的方法，来试用不同的核函数，误差最下的即为效果最好的核函数，或者也可以将多个核函数结合起来，形成混合核函数。在吴恩达的课上，也曾经给出过一系列的选择核函数的方法：

- 如果特征的数量大到和样本数量差不多，则选用LR或者线性核的SVM；
- 如果特征的数量小，样本的数量正常，则选用SVM+高斯核函数；
- 如果特征的数量小，而样本的数量很大，则需要手工添加一些特征从而变成第一种情况。

4.5 Soft Margin SVM

如果样本中有outlier，hard margin SVM会产生超平面严格将两类数据分开，outlier会明显影响margin的大小。于是我们引入soft margin SVM，允许数据点在一定程度上偏离一下超平面。

原本我们的约束问题是 $y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, n$ ，现在考虑到噪音问题，约束条件变成了 $y_i(w^T x_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n$

其中 $\xi_i \geq 0$ 称为松弛变量 (slack variable)，对应数据点 x_i 允许偏离的 functional margin 的量。当然，如果我们运行 ξ_i 任意大的话，那任意的超平面都是符合条件的了。所以，我们在原来的目标函数后面加上一项，使得这些 ξ_i 的总和也要最小：

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

其中 C 是一个参数，用于控制目标函数中两项（"寻找 margin 最大的超平面"和"保证数据点偏差量最小"）之间的权重。注意，其中 ξ 是需要优化的变量（之一），而 C 是一个事先确定好的常量。完整地写出来是这个样子：

$$\begin{aligned}
&\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\
&s.t., y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, \ldots, n \\
&\quad \xi_i \geq 0, i = 1, \ldots, n
\end{aligned}$$

C是惩罚系数。C越大，说明越不能容忍出现误差，C越小，更倾向于容忍误差，分类间隔越大，越容易欠拟合。

在选择模型时，可以根据cross validation对在不同参数下的模型进行对比后选择。

参考：https://blog.csdn.net/v_JULY_v/article/details/7624837

4.6 SVM VS Logistic Regression

4.6.1 Similarities

1）都是监督学习方法

2）都是线性分类学习方法

3）都是判别模型。

4.6.2 Differences

1）Loss function不同

LR是logloss， 而SVM是hinge loss

2）SVM不能产生概率，LR可以产生概率

LR本身就是基于概率的，所以它产生的结果代表了分成某一类的概率，而SVM则因为优化的目标不含有概率因素，所以其不能直接产生概率。【虽然现有的工具包，可以让SVM产生概率，但是那不是SVM原本自身产生的，而是在SVM基础上建立了一个别的模型，当其要输出概率的时候，还是转化为LR】

SVM甚至是SVR本质上都不是概率模型，因为其基于的假设就不是关于概率的。

3）SVM依赖于数据的测度，而LR不受影响

因为SVM是基于距离的，而LR是基于概率的，所以LR是不受数据不同维度测度不同的影响，而SVM因为要最小化12||w||2所以其依赖于不同维度测度的不同，如果差别较大需要做normalization，当然如果LR要加上正则化时，也是需要normalization一下的。

4）实际应用中的区别

根据经验来看，对于小规模数据集，SVM的效果要好于LR，但是大数据中，SVM的计算复杂度受到限制，而LR因为训练简单，可以在线训练，所以经常会被大量采用。

5）SVM 比LR的鲁棒性更强，但他俩对缺失值都敏感。

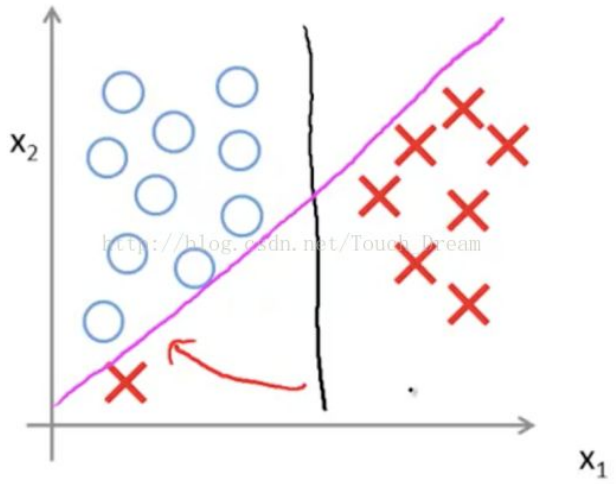
4.7 SVM vs Perceptron

相同点：两者都是监督学习的分类方法

不同点：

1）损失函数不同，感知机的损失函数是0-1损失函数，而SVM是hingeloss，感知机追求最大程度的正确划分，最小化错误，容易造成过拟合，效果如图中的紫线；SVM追求找到得到最大间隔的分类平面，在大致正确分类的同时，一定程度上避免过拟合，效果类似图中的黑线。

2）感知机使用的学习策略是梯度下降法，而SVM是用约束条件构造拉格朗日函数，然后求偏导令其为0求极值点。



4.8 Pros and Cons

优点

1）可以解决小样本下的机器学习问题

2）提高泛化性能

3）可以解决高维非线性问题，超高维的文本分类问题依然受欢迎

4）避免神经网络结构选择和局部极小的问题

缺点

1）对缺失值数据敏感

2）内存消耗大，难以解释

3）运行和调参麻烦

5. Decision Tree

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。可以把它看做是 if-else 规则的集合，也可以看做是特征空间上的条件概率分布。

那么如何选择根节点呢？

5.1 Entropy

5.1.1 Entropy

熵用来表示随机变量不确定性的度量，概率越小，熵越大，不确定性越大，把它搞清楚的所需要的信息量也就越大。信息的作用在于消除不确定性，NLP的大量问题就是要找到相关信息。

5.1.2 Joint Entropy

两个随机变量X，Y的联合分布，可以形成联合熵Joint Entropy，用H(X,Y)表示。

5.1.3 Conditional Entropy

在随机变量X发生的前提下，随机变量Y发生所新带来的熵定义为Y的条件熵，用H(Y|X)表示，用来衡量在已知随机变量X的条件下随机变量Y的不确定性，公式为： $H(Y|X)=\sum_{i=1}^{n} p_i H(Y|X=x_i)$

5.1.4 Information Gain

互信息，或者说信息增益，就是在得知信息Y的情况下，使得X的信息的不确定性减少的程度。即

$$I(X;Y)=H(X)-H(X|Y)$$

信息增益更大的特征具有更强的分类能力。
所以，所谓两个事件相关性的量化度量，就是在了解了其中一个Y的前提下，对消除另一个X不确定性所提供的信息量。互信息是一个取值在0 到min(H(X), H(Y)) 之间的函数，当X和Y完全相关时，它的取值是1，当二者完全无关时，取值为0。

互信息能够很好的解决机器翻译中歧义性的问题。比如，Bush可以表示灌木，也可以是布什总统的名字，我们可以在大量文本中找出和总统布什一起出现的互信息最大的一些词，比如总统、国会、华盛顿等，再用同样的方法找出和灌木丛一起出现的互信息最大的词，比如土壤、植物、野生等。有了这两组词，在翻译Bush时，看看上下文中哪类的词最多就可以了。

5.2 ID3

ID3算法的核心在于DT是以各个节点上用IG作为准则进行特征选择。具体做法为：

- 从根节点开始，对结点计算所有可能特征的IG，选择IG最大的特征作为结点的特征，并且用改特征的不同取值作为子节点。
- 对子节点递归地调用以上方法，构建决策树。
- 直到所有特征的IG都很小或者没有特征可选为止。

递归停止的条件，以下之一成立的时候停止：

- 节点中所有目标变量的值相同， 既然都已经是相同的值了自然没有必要在分裂了，直接返回这个值就好了
- 树的深度达到了预先指定的最大值 parameter: max_depth，或是节点的数据量小于预先定好的阈值 parameter: min_samples_split，即所谓的预剪枝
- 不纯度的减小量小于预先定好的阈值,也就是之进一步的分割数据并不能更好的降低数据的不纯度的时候就可以停止树分裂了

缺点：只能处理分类属性数据，划分过程中容易由于子集规模过小造成统计特征不充分而停止，其属性选择标准可能导致过度拟合。

5.3 C4.5

和ID3唯一的不同就是它是采用信息增益比选择树权。

信息增益作为划分训练数据集的特征，存在偏向于选择取值较多的特征的问题，用信息增益比可以很好的解决。

信息增益比：特征A对训练数据集D的信息增益比$g_R(D,A)$定义为其信息增益与训练数据集D关于特征A的值的熵$H_A(D)$之比

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$$

其中，$H_A(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$，n为特征A取值的个数。

输出变量只能是分类型，能够处理连续型和离散型属性数据，对缺失值的数据直接分配到概率最大的分支，会对生成的树进行后剪枝处理。但在构造过程中需对数据集进行多次的顺序扫描和排序，且产生的决策树不够稳定。

5.4 CART

Classification and regression tree。CART的不同在于它是一颗二叉树。

CART是利用基尼系数来划分树权的。它更加高效快速。

$$Gini(p) = \sum_{i=1}^{m} p_i(1-p_i) = 1 - \sum_{i-1}^{m} p_i^2$$

$$= 1 - \sum_{i-1}^{m} (\frac{|C_k|}{|D|})^2$$

基尼系数越小，表示选择该特征后熵下降最快，对分类模型效果更好，其和信息增益和信息增益率的选择指标是相反的。基尼系数主要是度量数据划分对训练数据集D的不纯度大小，所以在做特征选择时，选基尼系数最小的作为根节点。

输入和输出变量可为分类型或数值型，它在每一步的决策时只能是“是”或者“否”,采用代理测试处理缺失值，采用预剪枝和后剪枝相结合的方式剪枝。

参考：https://zhuanlan.zhihu.com/p/35423404

回归树与分类树的不同在于分类树用基尼系数作为决定根节点的计算标准，而回归树利用平方误差最小化。

5.5 Pruning

DT很容易产生过拟合的现象，缺乏泛化能力，因此需要进行剪枝来达到全局最优解。剪枝通过及消化DT整体的损失函数来实现。可以采用以下方法：
1）先剪枝：当熵减少的数量小于某一个阈值时，就停止分支的创建，这是一种贪心算法。常用方法是限制树的高度、叶子节点中的最少样本数量。

2）后剪枝：先创建完整的DT，然后再尝试消除多余的节点。一般XGBoost不会采用后剪枝，因为计算代价很高，合并一次叶节点就要计算一次测试集的表现，数据量大的情况下很耗时，而且也不必要，因为这样很容易过拟合。

5.6 Pros and cons

优点

1）易于理解，比LR直观，容易可视化

2）契合人类思考做决策的习惯

3）可以直接处理非数值型数据，不需要对dummy variables进行转化，甚至可以直接处理含缺失值的数据

缺点

1）对于各类别样本数量不一致的数据，信息增益偏向于那些具有更多数值的特征

2）不好处理变量中存在多种错综复杂的关系，比如金融数据分析

3）容易过拟合

决策树是如何处理缺失值的？

有两种情况：1）在训练样本的过程中，有一些样本属性值是缺失的，那么如何划分属性呢?

解决方法是，用所有未缺失的样本，和之前一样，计算每个属性的信息增益，但是这里的信息增益需要乘以一个系数（未缺失样本/总样本），然后再选择信息增益最大的的那个属性作为当前的最优划分。

2）在预测样本的过程中，有一些样本属性缺失，那么如何把它们归到决策树中呢？

解决方法是，如果样本x在划分属性a上的取值未知，则把这个样本同时划入所有子节点，但是所属每个子节点的概率不同。或者补充缺失值。

6. Ensemble learning

将已有的分类或回归算法通过一定方式组合起来，形成一个性能更加强大的分类器，更准确的说这是一种分类算法的组装方法。即将弱分类器组装成强分类器的方法，这些弱分类器通常有高偏差和低方差。

注意：只有当个基模型之间没有相关性的时候组合效果才会好，如果组合后没有提高精度，说明这些模型是相关的。

集成算法成功的关键在于保证弱分类器的多样性。

为何要集成：
1）模型选择。假设各个分类器之间具有一定的差异性（如不同的算法，或相同的算法配置不同的参数），会导致生成的分类的边界不同，也就是他们在决策的时候会范不同的错误，将他们结合后能得到更合理的边界，减少整体错误，实现更好的分类效果。
2）若数据集过大：可分为不同的子集，分别进行训练，然后再合成分类器；若数据集过小，可使用bootstrapping，从原来的样本集有放回的抽取m个子集，训练m个分类器，进行集成。
3）数据融合：当有多个不同的数据源，且每个数据源的特征集抽取方法都不同时，需要分别训练分类器然后再集成。

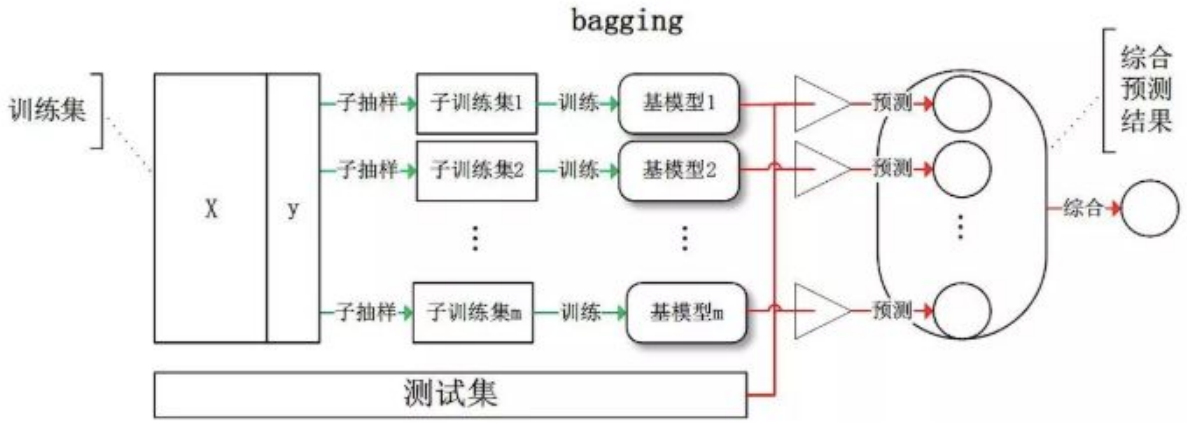
6.1 Bagging

处理分类问题时，通过投票进行统计，选出这些分类器输出值中被投票最多的分类作为结果。
处理回归问题时，通过求取各个分类器的输出值做平均，得到最终结果。

6.1.1 Bootstrap

Bootstrap是一种统计学里的抽样方法。就是一个在自身样本重采样（有放回）的方法来估计真实分布的问题。在小样本的时候效果好，但是样本如果很大，比如我们想统计大海里有多少条鱼，那么我们就算采样标记一万条也没用，因为实际数量太过庞大，去的样本相比于过于渺小，极端情况下，如果在放回后下一次抽样发现标记的鱼一条都没抽掉，就尴尬了。在小样本不知道样本分布的情况下，bootstrap方法最有用。

Bagging就是采用Bootstrap的抽样方法，选出n个训练样本（需要放回，因为别的分类器抽训练样本的时候也要用），然后在所有属性上，用这n个训练样本训练分类器（CART or SVM or …），重复以上两步m次，就可以得到m个分类器。最终将数据放在这m个分类器上跑，最后投票机制（多数服从少数）看最后属于哪一类。



6.1.2 Random Forest

随机森林中的随机包含两点：1）Bootstrap采样随机选择子样本；2）属性集中随机选择k个属性，在每个树节点分裂时，从这随机的K个属性中，通过DT的方法（IG，最大增益率，基尼基数等）选择最优的。

RF的构造过程：

1）用Random的方式构造一颗决策树

2）用1的方法构造很多颗决策树，每棵树都尽可能地生长而不进行剪枝，许多决策树构成一片森林，他们相互独立

3）测试数据进入每一个决策树，每棵树做自己的判断，然后进行投票选出最终所属类别（默认每棵树权重一致）

RF的优缺点

RF优点：

1）不容易过拟合，因为选择训练样本的时候就不是全部样本

2）处理高维数据集的能力强。它自带特征选择的能力，因此可以作为降维的一种方法

3）并行运算，速度较快

RF缺点：

1）较多的超参数需要提前设置

2）在解决回归问题的时候效果一般。因为它无法给出一个连续型的输出。

RF是如何处理缺失值的？

首先，给缺失值预设一些估计值，比如数值型特征，选择其余数据的中位数或众数作为当前的估计值，然后，根据估计的数值，建立随机森林，把所有的数据放进随机森林里面跑一遍。记录每一组数据在决策树中一步一步分类的路径，然后来判断哪组数据和缺失数据路径最相似，引入一个相似度矩阵，来记录数据之间的相似度，比如有N组数据，相似度矩阵大小就是N*N，如果缺失值是类别变量，通过权重投票得到新估计值，如果是数值型变量，通过加权平均得到新的估计值，如此迭代，直到得到稳定的估计值。

RF如何评估特征的重要性？

衡量变量重要性主要有两种方法：Decrease GINI and Decrease Accuracy

1) Decrease GINI: 对于回归问题，使用当前节点训练集的方差减去其左节点的方差以及右节点的方差。

2) Decrease Accuracy: 对于一棵树，我们可以用OOB样本得到测试误差1，然后随机改变OOB的第j列，保持其他列不变，对第j列进行随机的上下置换，得到误差2。然后用误差1-误差2来刻画变量j的重要性。基本思想就是：如果一个变量j足够重要，那么改变它会极大的增加测试误差，反之，如果改变它测试误差没有增大，则说明该变量不是那么重要。

RF中的OOB是如何计算的，有何优缺点？

Bagging的方法中Bootstrap每次约有1/3的样本不会出现在bootstrap所采集的样本集合中，当然也有就没有参加决策树的建立，把这1/3的数据称为袋外数据OOB，它可以用于取代测试集误差估计方法。

RF中参数对模型的影响

1）树深：一般来说，深度越大，拟合效果越好，但同时带来计算复杂度增加，速度减慢，并可能过拟合。一般情况下，在数据少或者特征少的时候可以不管这个值，如果样本量和特征很大才限制这个值

2）决策树的棵树

增加树会使得方差减少，但并不能改善偏差，太多的树反而可能造成欠拟合。

OOB误差的计算方法如下：

对于已经生成的RF，用袋外数据作为输入，假设有O个，这些随机分类树会给出相应的分类，假设当中有X个分类错误，那么袋外数据误差就是X/O，所以在RF中无需再进行交叉验证。

6.2 Boosting

Boosting的工作机制在于：先从初始训练集训练出一个分类器，再根据对这个分类器的表现对训练样本分布进行调整，使得先前在分类器做错的训练样本在后续受到更多的关注，然后基于调整后的样本分布来训练下一个分类器，如此重复，直到达到预期目标，然后将这些分类器按照权值相加。它本质上是一个加法模型，通过改变训练样本权重学习多个分类器并进行一些线性组合。

Boosting的最大好处在于，每一步的残差计算其实变相地增大了分错instance的权重，而已经分对的instance则都趋向于0。这样后面的树就能越来越专注那些前面被分错的instance。就像我们做互联网，总是先解决60%用户的需求凑合着，再解决35%用户的需求，最后才关注那5%人的需求，这样就能逐渐把产品做好，因为不同类型用户需求可能完全不同，需要分别独立分析。如果反过来做，或者刚上来就一定要做到尽善尽美，往往最终会竹篮打水一场空。

6.2.1 Adaboost

加法模型+指数损失函数+前项分布算法。从弱分类器出发反复训练，在其中不断调整数据权重或者是概率分布，同时提高前一轮被弱分类器误分的样本权值，最后用分类器进行投票表决（但分类器的重要性不同，误差小的分类型权重更大），其中的分类器可以根据需要采用任意合适的模型。

基本思想：
1）先赋予每个样本同样的权重
2）进行n次迭代，每次迭代后，对分类错误的样本加大权重（重采样），使得在下一次迭代过程中更加关注这些样本。

优点
1）不像DT那样容易过拟合，并且随着迭代次数的增加，会越接近真实分类
2）简单，不用做特征筛选

缺点：指数损失函数对outlier非常敏感，因为在每次的迭代过程会给outlier更大的权重。

Loss function

Adaboost的损失函数是指数函数。

学过Adaboost算法的人都知道，它是前向分步加法算法的特例，是一个加和模型，损失函数就是指数函数。在Adaboost中，经过m此迭代之后，可以得到fm(x)fm(x):

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

Adaboost每次迭代时的目的是为了找到最小化下列式子时的参数α和G:

$$\arg\min_{\alpha,G} = \sum_{i=1}^{N} exp[-y_i(f_{m-1}(x_i) + \alpha G(x_i))]$$

而指数损失函数(exp-loss）的标准形式如下

$$L(y, f(x)) = \exp[-yf(x)]$$

可以看出，Adaboost的目标式子就是指数损失，在给定n个样本的情况下，Adaboost的损失函数为：

$$L(y, f(x)) = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i f(x_i)]$$

6.2.2 GBDT

Gradient Boosting Decision Tree

将基分类器变成了CART树模型，基模型都是回归树，利用残差（负梯度，即损失函数的一阶导）学习概率。GBDT采用损失函数的梯度作为残差，并可以使用任何损失函数，只要损失函数是一阶连续可导的，这样一些比较健壮的损失函数就能得以应用，使模型抗噪能力增强。GBDT很容易出现过拟合问题，所以推荐GBDT的树深不要超过6，而RF可以在15以上。

GBDT的核心就在于，它每一次的计算都是为了减少上一次的残差，这个残差就是一个加预测值后能得真实值的累加量，进而在残差减少（负梯度）的方向上建立一个新的模型。比如A的真实年龄是18岁，但第一棵树的预测年龄是12岁，差了6岁，即残差为6岁。那么在第二棵树里我们把A的年龄设为6岁去学习，如果第二棵树真的能把A分到6岁的叶子节点，那累加两棵树的结论就是A的真实年龄；如果第二棵树的结论是5岁，则A仍然存在1岁的残差，第三棵树里A的年龄就变成1岁，继续学。

GBDT可以用于所有回归问题（线性和非线性），也可以用于二分类（设定阈值，大于阈值为正，反之为负）。

注意，在GBDT中，梯度下降的方法只考虑了一阶信息。

泰勒公式：

$$f(x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$
$$=f(x_0)+f^1(x_0)(x-x_0)+\frac{f^2(x_0)}{2}(x-x_0)^2+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$

Objective function

GBDT可以认为是由k个基模型组成的一个加法运算式：$\widehat{y_t}=\sum_{k=1}^{K}(f_k(x_k)),\ f_k\in F$ 。其中，F是指所有基模型组成的函数空间。

一个好的模型就是要在偏差和方差有一个好的平衡，而算法的损失函数正式代表了模型的偏差面，最小化损失函数，就是最小化模型的偏差，但同时我们要兼顾模型的方差，所以目标函数还包括抑制模型复杂度的正则项，因此目标函数可以写成：

$$Obj=\sum_{i-1}^{n}l(y_i,\hat{y}_i)+\sum_{k=1}^{K}\Omega(f_k)$$

而对于Boosting的算法，新模型的加入总是以优化目标函数为目的，我们以第t步的模型拟合为例，在这一步，模型对第i个样本 x_i 的预测为：

$\hat{y}_i^t=\hat{y}_i^{t-1}+f_t(x_i)$ ，此时目标函数可以被写为：

$$Obj^{(t)}=\sum_{i=1}^{n}l(y_i,\hat{y}_i^t)+\sum_{i=i}^{t}\Omega(f_i)$$
$$=\sum_{i=1}^{n}l\left(y_i,\hat{y}_i^{t-1}+f_t(x_i)\right)+\Omega(f_t)+constant$$

此时，优化目标函数，就相当于求得了 $f_t(x_i)$ 。

通过泰勒展开，将以上目标函数转化为：

$$Obj^{(t)}\approx\sum_{i=1}^{n}\left[g_if_t(x_i)+\frac{1}{2}h_if_t^2(x_i)\right]+\Omega(f_t)$$

其中 g_i 为损失函数的一阶导， h_i 为损失函数的二阶导，在GBDT中，只用到了一阶导，而在XGBoost中，用到了二阶导。

6.2.3 XGBoost

Extreme Gradient Boosting

Boosting算法最大的缺点在于：一是方差过高，容易过拟合；二是模型的构建过程是串行的，难以应用于大数据场景。而XGboost给出了boosting并行的计算思路，使之能够适应大数据场景，应用于大数据分布式集群环境中。

Loss function

XGBoost的目标函数包含损失函数和正则项两部分。损失函数是模型输出与样本真实值的差异度，差异度越低模型越好；正则项衡量模型的复杂度，复杂度越高模型越容易过拟合，所以正则项的值越低模型越好。

相对于GBDT，XGboost把二阶信息也用上了，根据泰勒展开，加入目标函数如下：

$$J(f_m) = \sum_{i=1}^{N} L\left(y_i, y_i^{m-1} + f_m(x_i)\right) + \Omega(f_m) + C$$

$$= \sum_{i=1}^{N} [L(y_i, y_i^{m-1}) + \frac{\partial L(y_i, y_i^{m-1})}{\partial y_i^{t-1}} * f_m(x_i) + \frac{1}{2} \frac{\partial^2 L(y_i, y_i^{m-1})}{\partial y_i^{t-1^2}} * f_m(x_i)^2]$$

$$+ \Omega(f_m) + C$$

因此能够更快的收敛。且在寻找最佳分割点的时候，可以引入并行计算，使得速度进一步提高。

XGboost是如何进行分裂的？

XGboost主要支出两种分裂节点的方法：贪心算法和近似算法。

1）贪心算法。

从树深为0开始：

1. 对每个节点枚举所有的可用特征；
2. 计算每个特征的信息增益；
3. 选择增益最大的特征作为分裂特征，用该特征的最佳分裂点作为分裂位置，在该节点分裂出左右两个新的叶节点，并为每个新节点关联对应的样本集；
4. 回到第一步，递归执行到满足特定条件为止。

2）贪心算法可以得到最优解，但当数据量太大时无法读入内存进行计算，近似算法主要针对贪心算法这一缺点给出了近似最优解。

XGboost为何要用泰勒二阶展开，有什么优势？

1）可扩展性：当目标函数是MSE时，展开是一阶项（残差）+二阶项的形式，而其他目标函数，比如log loss的展开就没有这种形式。为了统一，所以采用泰勒展开得到二阶项，这样就能把MSE推导的那套直接复用到其他自定义损失函数上。简单来说，就是为了统一损失函数求导的形式以支持自定义损失函数。MSE是最普遍的损失函数，且求导容易，求导后的形式也简单，所以理论上只要损失函数与MSE统一了，那么只要推导MSE就好了。

2）精确性：二阶信息本身能让梯度收敛得更快更准确。这一点在优化算法的牛顿法中已经证实。可以简单认为一阶导指引梯度方向，二阶导指引梯度变化方向。相对于GBDT的一阶泰勒展开，XGBoost采用了二阶泰勒展开，可以更精准得逼近真实的损失函数。

调参

在更改参数的时候，根据目标的不同目标函数可以设置为multi:softmax、binary:logistic、reg:logistic等。eta设置是衰减因子，也就是学习率，就是在步长前面乘以一个系数，这个值一般在0-1之间，在每次boosting　step后，我们都可以自动的得到新特征值的权重，eta将这些权重缩减从而使得boosting　　的过程更加的保守，不容易过拟合。max_depth是所用的树的最大深度，num_round是迭代计算次数。可以通过将tree_method设置成hist（运用了贪心优化算法）/gpu_hist，来加快运行速度。

XGBoost 如何防止过拟合？

一般而言有两种方法：

1）直接降低模型的复杂度：包含调整max_depth，　min_child_weight（如果一棵树分裂的过程中产生的叶子节点的所有特征的权重之和小于这个值，就停止分裂，这个值设置的越大，算法越保守）和gamma（也就是min_split_loss，进一步分裂成叶子节点所需要达到的最小损失减少，gamma越大，算法越保守，它的值在0到无穷大）

2) 调整学习效率参数shrinkage：类似于其他算法中的learning rate

3）引入正则项：包括树的叶子节点数量，叶子节点输出值的L1或L2范数

4）列（特征）采样：在迭代时只使用一部分特征来构建模型。列采样不仅缩短了计算时间，还提高了模型的预测效果。列采样的方法源于RF。

5）行（样本）采用：在迭代时只使用一部分样本来构建模型。

XGboost如何处理不平衡数据？

一般有通过两种方法：

1） 如果只关心预测的整体AUC，可以平衡通过scale_pos_weight这个参数来平衡正样本和负样本的权重

2） 如果关心预测正确的概率，那么我们不能够将数据集rebalance，要将参数max_delta_step(默认值一般是0，表示对叶子节点的输出没有限制）设置到一个很大的值（比如1）来帮助收敛。

XGboost如何处理缺失值？

在普通的GBDT中，对缺失值的处理是先手动对缺失值进行填充，然后当做有值的特征进行处理，但是这样人工填充不一定准确，而且没有什么理论依据。而XGboost分别尝试将缺失值数据划分到左叶子和右叶子，在计算其信息增益，选择增益更大的划分方案。如果训练中没有缺失值而预测时出现了缺失值，那么默认被分类到右子树。这种处理方式的问题在于当缺失值很多的情况下，比如缺失80%，那么xgb分裂的时候仅仅在20%的特征值上分裂，这是非常容易过拟合的。

XGboost 与GBDT的联系和区别

1）GBDT是机器学习算法，XGBoost是该算法的工程实现。

2）正则项：在使用CART作为基分类器时，XGBoost显式地加入了正则项来控制模型的复杂度，有利于防止过拟合，从而提高模型的泛化能力。

3）导数信息：GBDT在模型训练时只使用了代价函数的一阶导数信息，XGBoost对代价函数进行二阶泰勒展开，可以同时使用一阶和二阶导数。

4）基分类器：传统的GBDT采用CART作为基分类器，XGBoost支持多种类型的基分类器，比如线性分类器。

5）子采样：传统的GBDT在每轮迭代时使用全部的数据，XGBoost则采用了与随机森林相似的策略，支持对数据进行采样。

6）缺失值处理：传统GBDT没有设计对缺失值进行处理，XGBoost能够自动学习出缺失值的处理策略。

7）并行化：传统GBDT没有进行并行化设计，而XGBoost在样本间和特征间都可以采用并行计算。样本间并行表现在：XGBoost在每次迭代前，需要遍历所有样本，求解损失函数的一阶导和二阶导，由于不同样本间并无相关性，故可以对不同的样本使用并行计算；特征间表现在：在进行节点分裂时，需要遍历所有特征，此时可以对不同的特征进行并行计算，再汇总不同特征的判定结果，输出最优解。

8）XGBoost允许在每一轮boosting迭代中使用交叉验证。因此，可以方便的获得左右的boosting迭代次数，而GBDT只能用grid search检测有限个值。

XGboost的优缺点

优点：

1）精度更高：GBDT只用到一阶泰勒展开，而XGBoost对损失函数进行了二阶泰勒展开。XGBoost 引入二阶导一方面是为了增加精度，另一方面也是为了能够自定义损失函数，二阶泰勒展开可以近似大量损失函数；

2）灵活性更强：GBDT以 CART 作为基分类器，XGBoost 不仅支持 CART 还支持线性分类器，使用线性分类器的XGBoost相当于带和正则化项的Logistic Regression or Linear Regression。此外，XGBoost 工具支持自定义损失函数，只需函数支持一阶和二阶求导；

3) 正则化：XGBoost 在目标函数中加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、叶子节点权重的范式。正则项降低了模型的方差，使学习出来的模型更加简单，有助于防止过拟合，这也是XGBoost优于传统GBDT的一个特性。

4) Shrinkage（缩减）：相当于学习速率。XGBoost 在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。传统GBDT的实现也有学习速率；

5) 列抽样：XGBoost 借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算。这也是XGBoost异于传统GBDT的一个特性；

6) 缺失值处理：对于特征的值有缺失的样本，XGBoost 采用的稀疏感知算法可以自动学习出它的分裂方向；

7) XGBoost工具支持并行: XGBoost的并行不是tree粒度的并行，XGBoost也是一次迭代完才能进行下一次迭代的。XGBoost的并行是在特征粒度上的。决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost在训练之前，预先对数据进行了排序，然后保存为块结构，在块里面保存排序后的特征值及对应样本的引用，以便于获取样本的一阶、二阶导数值，后面的迭代中重复地使用这个结构，大大减小计算量，但注意这意味着保存原始特征之外还要保存原始特征的排序结果，消耗内存。这个块结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

8) 可并行的近似算法：树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以XGBoost还提出了一种可并行的近似算法，用于高效地生成候选的分割点。

缺点：每一次迭代的过程中，都需要遍历整个训练数据多次。如果将整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据会消耗很多时间。

6.2.4 LightGBM

优势在于：基于Histogram的决策树算法，带深度限制的leafwise 的叶子生长策略加快了计算并减少了过拟合的产生。

直方图算法的基本思想是先把连续的浮点特征值分箱。在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。缺点是直方图较为粗糙，会损失一定精度，但是在LightGBM的框架下，基学习器的精度损失可以通过引入更多的tree来弥补。

XGBoost 和LightGBM在节点分裂时候的区别

XGBoost是level-wise，而LightGBM是leaf-wise。level-wise对于同一层的非叶子节点，只要继续分裂能够产生正的增益就继续分裂下去，而leaf-wise更苛刻一些，同一层的非叶子节点，仅仅选择分裂增益最大的叶子节点进行分裂。

Level-wise过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度，不容易过拟合。但实际上Level-wise是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销，因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

Leaf-wise则是一种更为高效的策略，每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。因此同Level-wise相比，在分裂次数相同的情况下，Leaf-wise可以降低更多的误差，得到更好的精度。Leaf-wise的缺点是可能会长出比较深的决策树，产生过拟合。因此LightGBM在Leaf-wise之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。

因此，XGBoost控制树深更有效，LightGBM控制叶子数量更有效

6.3 Bagging vs Boosting

相同点：都是将已有的分类或是回归算法通过一定方式组合起来，行程一个性能更强的分类器，更准确的说是一种分类算法的组装方法，即将弱分类器组装成强分类器的方法。

不同点：

1） Bagging减少variance,而boosting减少bias和variance。其一是因为bagging采用了Bootstrap的采样方法，有放回地从一个训练集中选出数据，每轮的采集是相互独立的，可以有效地减掉variance，其次，最终将输出值平均/投票取得结果，也降低了variance。而boosting实际是个优化算法，每一轮的分类器都是基于降低上一轮结果的残差进行训练的，最终达到收敛。所以bias和variance都会减少。

Boosting每一轮的训练集是不变的，但训练集的分布会发生变化，因为每个样本在分类器中的权重会根据上一轮的分类结果进行调整。

2）样本权重：Bagging使用均匀抽样，每个样本的权重相等；Boosting根据每轮训练的错误率不断调整样本的权重，错误率越高，样本的权重越大。

3）预测函数：Bagging中通过投票的方式预测，所有预测函数的权重相同；Boosting中每个弱分类器都会有一个权重，对于那些分类误差小的分类器，将会有更大的权重。

4）运算方式：Bagging中所有预测函数可以并行生成，他们互相独立；Boosting中各个预测函数只能顺序生成，后一个模型参数需要前一轮的预测结果。

7. Naive Bayes

7.1 Bayes Theorem

7.1.1 Possibilities

条件概率（后验概率）即事件A在另一件事件B已发生的条件下发生的概率，表示为P(A|B)，读作“在条件B下A的概率”。条件概率的计算公式为：

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

联合概率表示两个事件共同发生的概率。A与B的联合概率就表示为 $P(A \cap B)$或者$P(A, B)$。

先验概率是某件事发生的概率。

7.1.2 Bayes Theorem

接着，考虑一个问题：P(A|B)是在B发生的情况下A发生的可能性。
首先，事件B发生之前，我们对事件A的发生有一个基本的概率判断，称为A的先验概率，用P(A)表示；
其次，事件B发生之后，我们对事件A的发生概率重新评估，称为A的后验概率，用P(A|B)表示；
类似的，事件A发生之前，我们对事件B的发生有一个基本的概率判断，称为B的先验概率，用P(B)表示；
同样，事件A发生之后，我们对事件B的发生概率重新评估，称为B的后验概率，用P(B|A)表示。
贝叶斯定理便是基于下述贝叶斯公式：

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

上述公式的推导其实非常简单，就是从条件概率推出。

7.2 Naive Bayes Classifier

贝叶斯分类器是一类分类算法的总称，这类算法均以贝叶斯定理为理论基础。贝叶斯分类器的分类原理是通过先验概率，利用贝叶斯公式计算出后验概率，选择最大后验概率所对应的分类结果。

$$P(Y_j|X) = \frac{P(X|Y_j)P(Y_j)}{\sum_{i=1}^{m} P(X|Y=Y_i)P(Y_i)}$$

其中，
$P(Y_j)$ 为先验概率；
$P(X|Y_j)$ 为似然函数；
$P(X)$ 为归一化项；
$P(Y_j|X)$ 为后验概率；

朴素贝叶斯算法的核心就是：利用最大似然估计，求取P(X|Yj),最终比较后验概率大小来判别类别。

7.2.1 MLE VS MAP

最大似然估计MLE是求参数 θ，使得似然函数 $P(x_0^{\theta})$ 最大，最大后验概率估计MAP则是想求 θ 使得 $P(x_0^{\theta})P(\theta)$ 最大。所以，MAP就是加入了模型参数本身的概率分布，或者说，MLE认为模型参数本身的概率分布是军印的，即改概率为一个固定值。

7.2.2 Pros and Cons

优点：

1）需要设置的超参数少，对缺失数据不敏感

2）有着坚实的数学基础，以及稳定的分类相率

缺点：

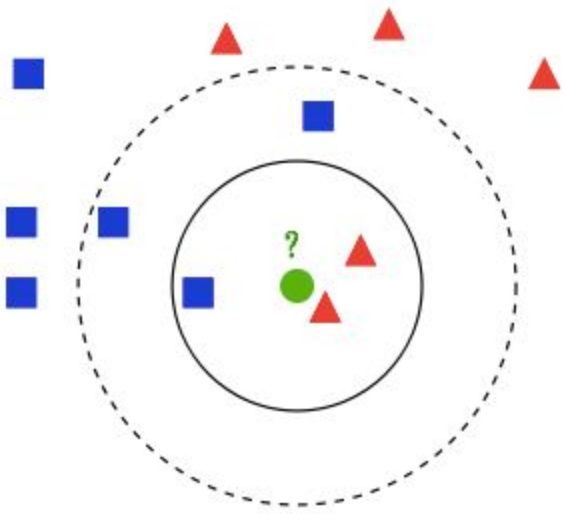
1）假设属性之间相互独立，故而称为“朴素”

2）需要知道先验概率

朴素的原则是假设各个特征属性之间是相互独立的，以此极大减少了参数量，在文本分类中，朴素贝叶斯的分类效果依然很好，即便各特征之间存在较强的相关性。因为特征之间的相关性可能在不同类别中均匀分布，不同特征的相关性可能相互抵消，因此，独立性假设会改变分类的后验概率大小，但不会改变其相对大小，所以不会影响分类结果。

8. KNN

KNN可以用于分类和回归。在做分类问题时，给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例，这K个实例的多数属于某个类，就把该输入实例分类到这个类中。在样本较少且有典型性时，分类效果最好。



比如上图，如果K=3，那么圆圈就被分类为红色三角形，而如果K=5， 圆圈就被分类为蓝色正方形。

KNN实际上是对训练数据集对特征向量空间进行划分，并将其作为分类的模型，K值的选择、距离度量、分类决策规则（一般为多数表决）是KNN的三个基本要素。KNN的算法训练边界一定不是直线，由于它是看周围最近的K个样本类别从而确定分类，所以边界一定是坑坑洼洼的。

KNN的主要问题在于如何确定K以及如何衡量样本之间的距离。

8.1 The number of K

K值的大小即为邻居的多少，K值如果取值较小，相当于用较小的领域中的训练实例进行预测，‘学习’近似误差会减少，只有与输入实例较近或者相似的训练实例才会对预测结果起作用，与此同时带来的问题是‘学习’的估计误差会增大，不易泛化，换句话说，K值的减小意味着整体模型变得复杂，容易发生过拟合；如果K值取值过大，就相当于用较大领域中的实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习近似误差会增大，K值的增大以为着整体模型变得简单。

在确定K值的时候，一般先选取一个较小的数值，采取交叉验证发来选取最优K值。

8.2 The distance between neighbours

距离的度量一般采用欧氏距离。

设特征空间 $\mathcal{X}$ 是 n 维实数向量空间 $\mathbf{R}^n$，$x_i, x_j \in \mathcal{X}$，$x_i = (x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(n)})^{\mathrm{T}}$，$x_j = (x_j^{(1)}, x_j^{(2)}, \cdots, x_j^{(n)})^{\mathrm{T}}$，$x_i, x_j$ 的 L_p 距离定义为

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} | x_i^{(l)} - x_j^{(l)} |^p \right)^{\frac{1}{p}} \tag{3.2}$$

这里 $p \geqslant 1$．当 $p = 2$ 时，称为欧氏距离(Euclidean distance)，即

$$L_2(x_i, x_j) = \left(\sum_{l=1}^{n} | x_i^{(l)} - x_j^{(l)} |^2 \right)^{\frac{1}{2}} \tag{3.3}$$

当 $p = 1$ 时，称为曼哈顿距离（Manhattan distance），即

$$L_1(x_i, x_j) = \sum_{l=1}^{n} | x_i^{(l)} - x_j^{(l)} | \tag{3.4}$$

8.3 Optimization

实现k近邻法时，主要考虑的问题是如何对训练数据进行快速k近邻搜索，这点在特征空间的维数大以及训练数据容量大时尤其重要。k近邻的最简单实现是线性扫描，这时需要计算输入实例与每个训练实例的距离，当训练集很大时，计算非常耗时，可以采用其他的结构存储训练数据，以减少计算距离的次数，比如kd-tree。

8.4 Pros and Cons

优点

1）思想简单，理论成熟，既可以用来做分类也可以用来做回归

2）可以用于做非线性分类

3）训练时间复杂度为O(N)

4）准确度高，对数据没有假设，对异常值不敏感

缺点

1）计算量大，需要大量的内存

2）对于样本分类不均衡问题，会产生误判

3）输出的可解释性不强

9. K Means

Kmeans的核心是物以类聚人以群分。

9.1 Process

过程如下：

1）首先输入K的值，即我们指定希望通过聚类得到K个分组

2）从数据集中随机抽取K个数据点作为质心

3）对数据集中的所有其他样本，计算他们与每个质心的距离，离哪个质心近，就分为哪一类

4）这时所有样本被分为了K类， 再通过计算每一类的中心点重新确认质心，重复第三步

5）如果新的质心和旧的质心之间的距离小于某一个阈值（表示重新计算质心的位置变化不大，趋于稳定，或者说收敛），我们就认为聚类已经达到了效果，算法终止。

K Means与EM

我们目的是将样本分成k个类，即求每个样例x的隐含类别y，然后利用隐含类别将x归类。由于我们事先不知道类别y，那么我们首先可以对每个样例假定一个y吧，但是怎么知道假定的对不对呢？怎么评价假定的好不好呢？

用样本的极大似然估计来度量，这里是就是x和y的联合分布P(x,y)了。如果找到的y能够使P(x,y)最大，那么我们找到的y就是样例x的最佳类别了，x顺手就聚类了。但是我们第一次指定的y不一定会让P(x,y)最大，而且P(x,y)还依赖于其他未知参数，当然在给定y的情况下，我们可以调整其他参数让P(x,y)最大。但是调整完参数后，我们发现有更好的y可以指定，那么我们重新指定y，然后再计算P(x,y)最大时的参数，反复迭代直至没有更好的y可以指定。

这样从K-means里我们可以看出它其实就是EM的体现，E步是确定隐含类别变量，M步更新其他参数来使J最小化。这里的隐含类别变量指定方法比较特殊，属于硬指定，从k个类别中硬选出一个给样例，而不是对每个类别赋予不同的概率。总体思想还是一个迭代优化过程，有目标函数，也有参数变量，只是多了个隐含变量，确定其他参数估计隐含变量，再确定隐含变量估计其他参数，直至目标函数达到最优。

9.2 The number of K

9.2.1 Demands

按照建模需求和目的确定聚类数目。比如，一家游戏公司想把所有玩家做聚类分析，分成顶级、高级、中级和菜鸟四类，那么K=4；如果房地产公司想把当地的商品房分成高中低三档，那么K=3。按需选择虽然合理，但未必能保证在做k-means时能够清晰地分界。

9.2.2 Elbow method

Elbow method的核心指标是SSE (sum of the squared error)。

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2$$

其中，Ci是第i个簇，p是Ci中的样本点，mi是Ci的质心（Ci中所有样本的均值），SSE是所有样本的聚类误差，代表了聚类效果的好坏。

手肘法的核心思想是：随着聚类数k的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和SSE自然会逐渐变小。并且，当k小于真实聚类数时，由于k的增大会大幅增加每个簇的聚合程度，故SSE的下降幅度会很大，而当k到达真实聚类数时，再增加k所得到的聚合程度回报会迅速变小，所以SSE的下降幅度会骤减，然后随着k值的继续增大而趋于平缓，也就是说SSE和k的关系图是一个手肘的形状，而这个肘部对应的k值就是数据的真实聚类数。

9.2.3 Silhouette score

轮廓系数法的核心指标是silhouette coefficient，某个样本点Xi的轮廓系数定义为：

$$S = \frac{b - a}{\max(a, b)}$$

其中，a是Xi与同簇的其他样本的平均距离，称为凝聚度，b是Xi与最近簇中所有样本的平均距离，称为分离度。

求出所有样本的轮廓系数后再求平均值就得到了平均轮廓系数。平均轮廓系数的取值范围为[-1,1]，且簇内样本的距离越近，簇间样本距离越远，平均轮廓系数越大，聚类效果越好。那么，很自然地，平均轮廓系数最大的k便是最佳聚类数。

需要注意的是，由于轮廓系数考虑了分离度b，也就是样本与最近簇中所有样本的平均距离，那么从定义来看，轮廓系数大，不一定是凝聚度a（样本与同簇的其他样本的平均距离）小，而可能是b和a都很大的情况下b相对a大的多，这么一来，a是有可能取得比较大的。a一大，样本与同簇的其他样本的平均距离就大，簇的紧凑程度就弱，那么簇内样本离质心的距离也大，从而导致SSE较大。所以，虽然轮廓系数引入了分离度b而限制了聚类划分的程度，但是同样会引来最优结果的SSE比较大的问题。

所以轮廓系数确定出的最优K值不一定是最优的，有时候还需要根据SSE去辅助选取，在一般情况下，手肘法还是首选。

9.3 Pros and Cons

优点：简单高效，易于理解，聚类效果好。
缺点及优化：

1）算法可能找到的是局部最优解，可以采用改进的二分K-means算法。

二分K-means算法：首先将整个数据集看成一个簇，然后进行一次k-means（k=2）算法将该簇一分为二，并计算每个簇的SSE，选择SSE最大的簇迭代上述过程再次一分为二，直至簇数达到用户指定k为止，此时可以达到全局最优。

2）算法的结果非常依赖初始随机选择的聚类中心的位置，可以通经过k-means++选择彼此距离尽可能远的k个点作为簇中心，通过多次执行该算法来减少初始中心敏感的影响。

3）聚类结果对k的取值依赖性大，简单常用的办法是：从一个起始值开始到一个最大值，每一个值运行kmeans聚类，通过手肘法或是轮廓系数法来确定最好的一次聚类结果下的K。

4）对outliers敏感。可以去除outliers后再聚类。

2）在大数据的条件下，会耗费大量的时间和内存。可以使用kd-tree（多维数组上的BST）或ball-tree（数据结构）

时间复杂度：O(tKn)，其中，t为迭代次数，K为簇的数目，n为数据集容量。

空间复杂度：O((m+K)n)，其中，K为簇的数目，m为记录数，n为维度。