

深度学习+项目问题

1. 项目后续如何优化？收获是什么？

2. 单张图片测试的速度？

使用GPU，一张图片的测试速度是0.032s

3. CNN中对结果影响比较大的参数？

学习率、batch-size、卷积核大小和数量、网络层数。

4. 如何进行fintuning，层数不相同的情况

层数不同也可以调试，一般将不想调试的网络层修改名字即可。

5. 反卷积网络是否有了解过？

6. 卷积核大小的选择

- 更小的卷积核可以减小参数，节省运算开销，虽然训练时间会变长，但是总体上参数和预测时间会减少，同时小的卷积核会提取到图片中局部的特征，而大的卷积核则可以提取到更加全局的信息。
- 两个3*3的卷积核可以达到一个5*5卷积核的作用，并且还能减少参数和计算时间。

7. 1*1 卷积核的作用

- 实现跨通道的交互和信息整合，实现多个feature map的线性组合，实现跨通道的信息整合
- 进行卷积核通道数的降维和升维
- 实现平滑操作
- 可以在保持feature map 尺寸不变（即不损失分辨率）的前提下大幅增加非线性特性，把网络做得很deep。

8. CNN对图像分类效果好的原因

图像存在局部相关性，而卷积运算可以获取这种空间相关性。

9. BatchNorm层的作用，为什么使用？用在哪里？什么时候使用？为什么能产生这样的效果？

（1）作用有两个，一个是逐层尺度归一，避免梯度消失和溢出；其次是加快收敛速度，可以防止过拟合；
（2）防止梯度弥散，在BN中，是通过将activation规范为均值和方差一致的手段使得原本会减小的activation的scale变大。

（3）BN应作用在非线性映射前，即对做规范化。

（4）在神经网络训练时遇到收敛速度很慢，或梯度爆炸等无法训练的状况时可以尝试BN来解决。另外，在一般使用情况下也可以加入BN来加快训练速度，提高模型精度。

（5）原因在于神经网络学习过程本质就是为了学习数据分布，一旦训练数据与测试数据的分布不同，那么网络的泛化能力也大大降低；另外一方面，一旦每批训练数据的分布各不相同(batch 梯度下降)，那么网络就要在每次迭代都去学习适应不同的分布，这样将会大大降低网络的训练速度，这也正是为什么我们需要对数据都要做一个归一化预处理的原因。

对于深度网络的训练是一个复杂的过程，只要网络的前面几层发生微小的改变，那么后面几层就会被累积放大下去。一旦网络某一层的输入数据的分布发生改变，那么这一层网络就需要去适应学习这个新的数据分布，所以如果

训练过程中，训练数据的分布一直在发生变化，那么将会影响网络的训练速度。

我们知道网络一旦train起来，那么参数就要发生更新，除了输入层的数据外(因为输入层数据，我们已经人为的为每个样本归一化)，后面网络每一层的输入数据分布是一直在发生变化的，因为在训练的时候，前面层训练参数的更新将导致后面层输入数据分布的变化。以网络第二层为例：网络的第二层输入，是由第一层的参数和input计算得到的，而第一层的参数在整个训练过程中一直在变化，因此必然会引起后面每一层输入数据分布的改变。我们把网络中间层在训练过程中，数据分布的改变称之为：“Internal Covariate Shift”。Paper所提出的算法，就是要解决在训练过程中，中间层数据分布发生改变的情况，于是就有了Batch Normalization，这个牛逼算法的诞生。

10. 全卷积网络介绍

FCN将传统CNN中的全连接层转化成一个个的卷积层。在传统的CNN结构中，前5层是卷积层，第6层和第7层分别是一个长度为4096的一维向量，第8层是长度为1000的一维向量，分别对应1000个类别的概率。FCN将这3层表示为卷积层，卷积核的大小(通道数，宽，高)分别为(4096,1,1)、(4096,1,1)、(1000,1,1)。所有的层都是卷积层，故称为全卷积网络。

可以发现，经过多次卷积(还有pooling)以后，得到的图像越来越小,分辨率越来越低(粗略的图像)，那么FCN是如何得到图像中每一个像素的类别的呢？为了从这个分辨率低的粗略图像恢复到原图的分辨率，FCN使用了上采样。例如经过5次卷积(和pooling)以后，图像的分辨率依次缩小了2, 4, 8, 16, 32倍。对于最后一层的输出图像，需要进行32倍的上采样，以得到原图一样的大小。

这个上采样是通过反卷积(deconvolution)实现的。对第5层的输出(32倍放大)反卷积到原图大小，得到的结果还是不够精确，一些细节无法恢复。于是Jonathan将第4层的输出和第3层的输出也依次反卷积，分别需要16倍和8倍上采样，结果就精细一些了。

与经典的CNN在卷积层之后使用全连接层得到固定长度的特征向量进行分类不同，FCN可以接受任意尺寸的输入图像，采用反卷积层对最后一个卷积层的feature map进行上采样，使它恢复到输入图像相同的尺寸，从而可以对每个像素都产生了一个预测，同时也保留了原始输入图像中的空间信息，最后在上采样的特征图上进行逐像素分类。论文中逐像素计算softmax分类的损失，相当于每一个像素对应一个训练样本。

与传统用CNN进行图像分割的方法相比，FCN有两大明显的优点：一是可以接受任意大小的输入图像，而不用要求所有的训练图像和测试图像具有同样的尺寸。二是更加高效，因为避免了由于使用像素块而带来的重复存储和计算卷积的问题。

同时FCN的缺点也比较明显：一是得到的结果还是不够精细。进行8倍上采样虽然比32倍的效果好了很多，但是上采样的结果还是比较模糊和平滑，对图像中的细节不敏感。二是对各个像素进行分类，没有充分考虑像素与像素之间的关系，忽略了在通常的基于像素分类的分割方法中使用的空间规整(spatial regularization)步骤，缺乏空间一致性。

传统的基于CNN的分割方法：为了对一个像素分类，使用该像素周围的一个图像块作为CNN的输入用于训练和预测。这种方法有几个缺点：一是存储开销很大。例如对每个像素使用的图像块的大小为15x15，然后不断滑动窗口，每次滑动的窗口给CNN进行判别分类，因此则所需的存储空间根据滑动窗口的次数和大小急剧上升。二是计算效率低下。相邻的像素块基本上是重复的，针对每个像素块逐个计算卷积，这种计算也有很大程度上的重复。三是像素块大小的限制了感知区域的大小。通常像素块的大小比整幅图像的大小小很多，只能提取一些局部的特征，从而导致分类的性能受到限制。

而全卷积网络(FCN)试图从抽象的特征中恢复出每个像素所属的类别。即从图像级别的分类进一步延伸到像素级别的分类。

11.BP算法？如果对 $wx+b$ 的反向计算是 $w^T * \epsilon$,为什么是 w 的转置？

- (1)
- (2) 维度匹配

12.Caffe中im2col函数的作用？

将卷积转为矩阵相乘，加快卷积运算的计算速度。

13. 卷积是如何实现的？

卷积核在卷积计算时没有“翻转”，而是与输入图片做滑动窗口“相关”计算。

14. 动量的作用

计算梯度时考虑历史梯度信息，使随机梯度下降更容易跳出局部最优，加速收敛

15 卷积本质是什么？卷积神经网络本质是什么？

(1)

(2)

16. 项目，与业界最好成绩相比？优化了哪些，或者相比更好的地方在哪里？

17. CNN中，计算速度主要取决于什么？

卷积运算，并且主要是channel这个维度。

18. 如何提高卷积运算速度，比如提高矩阵运算？

19. 手机实现CNN，主要使用哪个实现提速？

使用ARM的汇编指令。

20.如何增大卷积网络的感受野？

增大卷积核，加深网络，加pooling层

感受野的定义参考<http://blog.csdn.net/wonengguwozai/article/details/73133737>和
<https://zhuanlan.zhihu.com/p/22627224>。

21. 什么样的资料集不适合用深度学习？

- 数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势。
- 数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像 / 语音 / 自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。举个例子：预测一个人的健康状况，相关的参数会有年龄、职业、收入、家庭状况等各种元素，将这些元素打乱，并不会影响相关的结果。

22. Caffe为什么要用protobuf？

Protocol Buffers 是一种轻便高效的结构化数据存储格式，可以用于结构化数据串行化，或者说序列化。它很适合做数据存储或 RPC 数据交换格式。可用于通讯协议、数据存储等领域的语言无关、平台无关、可扩展的序列化结构数据格式。

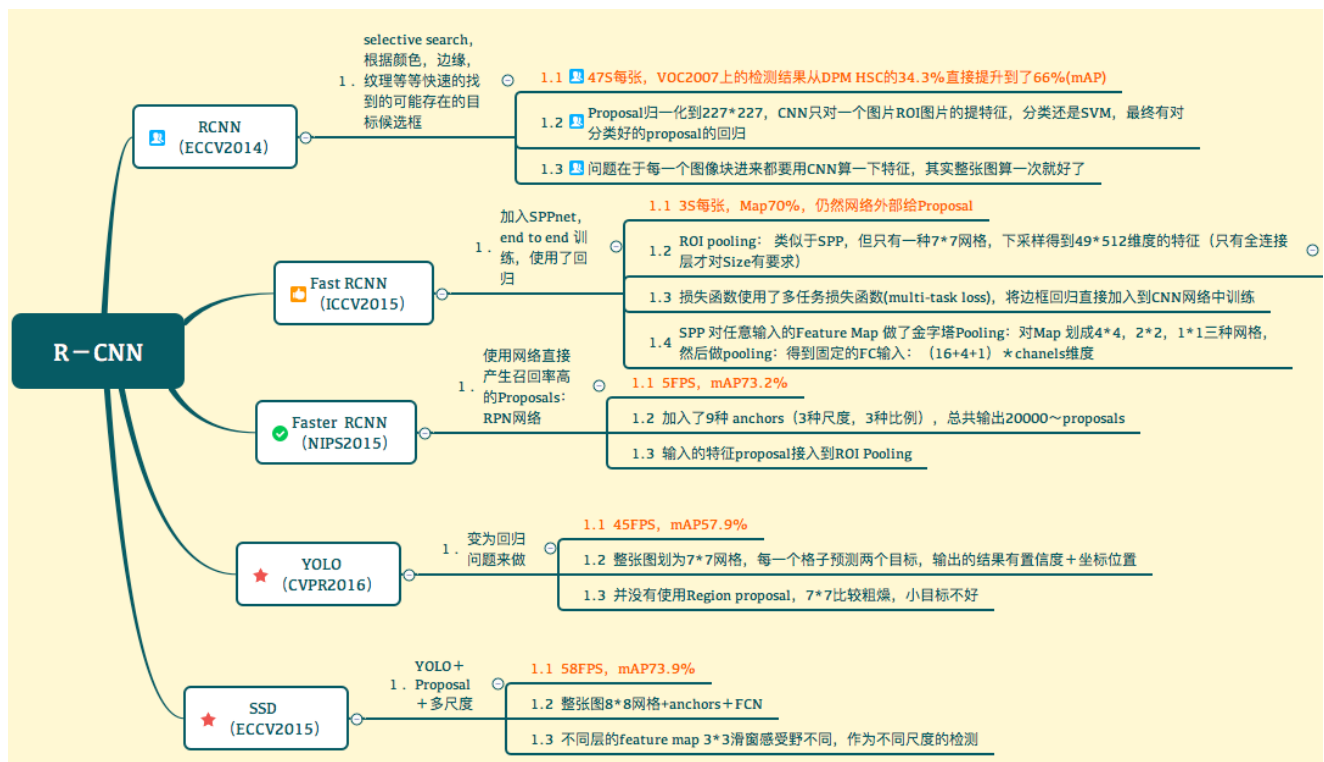
对于CNN网络而言，其主要有两部分组成：网络具体结构和网络的具体优化算法及参数。对于框架的使用者而言，用户只需输入两个描述文件即可得到对该网络的优化结果，这无疑是非常方便的。

Caffe框架选择使用谷歌的开源protobuf工具对这两部分进行描述，解析和存储，这一部分为caffe的实现节省了大量的代码。

参考文章[caffe数据格式（Google Protocol Buffers）](#)、[Caffe学习（十）protobuf及caffe.proto解析](#)、[Caffe 深度学习框架上手教程](#)。

23. rcnn/fast rcnn/fater rcnn区别？

参考文章[RCNN, Fast RCNN, Faster RCNN 总结](#)、[如何评价rcnn、fast-rcnn和faster-rcnn这一系列方法？](#)、[RCNN->SPP net -> Fast RCNN -> Faster RCNN](#)、[RCNN, Fast-RCNN, Faster-RCNN以及YOLO的区别](#)。



R-CNN的流程主要分为四步：

1. 输入图像；
2. 利用选择性搜索（Selective Search）等区域生成算法在输入图像中提取Region Proposal（大概2000个）；
3. 将第一步中产生的每个Region Proposal分别resize后（也即图中的warped region，文章中是归一化为227×227）作为CNN网络的输入；
4. CNN网络提取到经过resize的region proposal的特征送入每一类的SVM分类器，判断是否属于该类；

FRCNN针对**RCNN**在训练时是**multi-stage pipeline**和训练的过程中很耗费时间空间的问题进行改进。它主要是将深度网络和后面的**SVM**分类两个阶段整合到一起，使用一个新的网络直接做分类和回归。主要做以下改进：

1. 最后一个卷积层后加了一个ROI pooling layer。ROI pooling layer首先可以将image中的ROI定位到feature map，然后用一个单层的SPP layer将这个feature map patch池化为固定大小的feature之后再传入全连接层。
2. 损失函数使用了多任务损失函数(multi-task loss)，将边框回归直接加入到CNN网络中训练。

Fast-RCNN的速度瓶颈在**Region proposal**上。于是将**Region proposal**也交给**CNN**来做，提出了**Faster-RCNN**。Fater-RCNN中的region proposal network实质是一个Fast-RCNN，这个Fast-RCNN输入的是固定的（把一张图片划分成n*n个区域，每个区域给出9个不同ratio和scale的proposal），输出的是对输入的固定proposal是属于背景还是前景的判断和对齐位置的修正（regression）。Region proposal network的输出再输入第二个Fast-RCNN做更精细的分类和Boundingbox的位置修正。Fater-RCNN速度更快了，而且用VGG net作为feature extractor时在VOC2007上mAP能到73%。

24. Tensorflow 为什么用图？

TensorFlow是用数据流图(data flow graphs)技术来进行数值计算的。数据流图是描述有向图中的数值计算过程。有向图中的节点通常代表数学运算，但也可以表示数据的输入、输出和读写等操作；有向图中的边表示节点之间的某种联系，它负责传输多维数据(Tensors)。图中这些tensors的flow也就是TensorFlow的命名来源。节点可以被分配到多个计算设备上，可以异步和并行地执行操作。因为是有向图，所以只有等到之前的入度节点们的计算状态完成后，当前节点才能执行操作。

图计算在内存使用和运算时间两个方面比较高效。

25. RELU比Sigmoid, Tanh好的原因？

- 采用sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多。
- 对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失，参见 @Haofeng Li 答案的第三点），从而无法完成深层网络的训练。
- Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生

26. softmax函数计算时候为什么要减去一个最大值？

参考文章[王货|softmax函数计算时候为什么要减去一个最大值？](#)

$$S_i = \frac{e^i}{\sum_j e^j}$$

Softmax的计算公式如下：

当我们运算比较小的值的时候是不会有问题的，但是如果运算的值比较大的时候，比如很大或很小的时候，朴素的直接计算会上溢出或下溢出，从而导致严重问题。

举个例子，对于[3,1,-3]，直接计算是可行的，我们可以得到(0.88,0.12,0)。

但对于[1000,1000,1000]，却并不可行，我们会得到inf(这也是深度学习训练过程常见的一个错误，看了本文之后，以后出现inf的时候，至少可以考虑softmax运算的上溢和下溢)；对于[-1000,-999,-1000]，还是不行，我们会得到-inf。

这是因为你的浮点数只有64位，在计算指数函数的环节， $\exp\{1000\} = \text{inf}$ ，会发生上溢出； $\exp\{-1000\} = 0$ ，会发生下溢出。

解决的办法如下：

$$\log \sum_{n=1}^N \exp\{x_n\} = a + \log \sum_{n=1}^N \exp\{x_n - a\}$$

对任意a都成立，这意味着我们可以自由地调节指数函数的指数部分，一个典型的做法是取所有数据中的最大值：

$$a = \max x_1, x_2, \dots, x_n$$

这可以保证指数最大不会超过**0**，于是你就不会上溢出。即便剩余的部分下溢出了，加了**a**之后，你也能得到一个合理的值。

证明：

证明**softmax**不受输入的常数偏移影响，即

$$\text{softmax}(x) = \text{softmax}(x+c)$$

也就是证明加了偏移**c**之后，对整个**softmax**层的作用不起影响。如下：

$$\begin{aligned} (\text{softmax}(x+c))_i &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} = \frac{e^{x_i} \times e^c}{e^c \times \sum_j e^{x_j}} \\ &= \frac{e^{x_i} \times \cancel{e^c}}{\cancel{e^c} \times \sum_j e^{x_j}} = (\text{softmax}(x))_i \end{aligned}$$

27. 过拟合和欠拟合的定义。避免过拟合的方法。

过拟合：模型复杂度过高，或者训练过度，模型拟合数据分布的同时也拟合了噪声，导致模型在训练集上拟合效果很好，训练误差很低，但在测试集上误差很大，不能有效预测新样本；

欠拟合：模型复杂度过低，或者训练数据少，不能很好拟合训练数据，训练误差大；

避免过拟合的途径：

1. 根据大数定理，当数据量足够大时，模型将会无限逼近实际，即数据增强；
2. 减少参数数量，降低模型的复杂度，剔除和模型不太相关的参数；
3. 对学习参数进行正则化，约束参数的收敛空间，提高模型的泛化能力；
4. 在训练时，避免因训练过度导致的模型过拟合；

28. 卷积层的参数量计算

计算公式是 $parameters = inputSize * kernelWidth * kernelHeight * outputSize$ 。也就是卷积核大小*卷积核个数（也就是输出特征图个数）*卷积层的深度（也就是输入通道）。

机器学习算法

1. 随机森林的原理？
2. 随机森林如何避免过拟合，而决策树会过拟合的原因
3. 决策树特征选择的方法？
4. 决策树，如果特征值是连续的如何选择？

C4.5算法。首先对特征值进行升序排序，然后使用二分法，即寻找相邻两个特征值的中点作为分裂点，如1,2,3，可以选择1.5和2.5作为分裂点，选择标准是信息增益，选择信息增益最大的分裂点，然后再计算最佳分裂点的信息增益率作为该特征的信息增益率。

5. 决策树，做回归时使用的准则是？

平方误差最小化

6. KNN的三个注意点，K过大和过小时的偏差和方差？

(1) K值选择，距离度量和分类决策规则；

(2) k过小的时候，偏差小而方差大，容易过拟合；k过大时，偏差大，方差小

7. LR使用L1正则化，再使用梯度下降，会出现什么问题？

8. LR的损失函数，为什么使用这个？


这是似然损失函数，使用极大似然估计求解；

参数估计方法除了极大似然估计，还有EM算法、点估计。点估计是计算样本的均值，使用均值作为参数估计。

9 随机森林需要剪枝吗？

不用，一般很多的决策树算法都有一个很重要的步骤-剪枝，这里不需要这样做，因为之前的两个随机采样的过程保证了随机性，即对数据的随机采样和特征的随机采样，就算不减枝，也不会出现过拟合。

10. SVM的kkt条件？用于解决什么问题？

(1) ，即《统计学习方法》P105。它是指函数的最优值必定满足下面条件：

(1) L对各个x求导为零；

(2) $h(x)=0$;

(3) $\sum \alpha_i g_i(x) = 0, \alpha_i \geq 0$

(2) 拉格朗日乘子法(Lagrange Multiplier)和KKT(Karush-Kuhn-Tucker)条件是求解约束优化问题的重要方法，在有等式约束时使用拉格朗日乘子法，在有不等约束时使用KKT条件。前提是：只有当目标函数为[凸函数](#)时，使用这两种方法才保证求得的是最优解。

11. SVM的核函数？如果使用RBF核函数，可以将原本5维的数据映射到多少维？

(1)

(2) 映射到无限维，因为它是使用泰勒展开式，可以展开到无穷维。泰勒展开式其实是0-n维的多项式核函数的和。我们知道多项式核函数将低维数据映射到高维(维度是有限的)，那么 对于无限个 不同维的多项式核函数之和的高斯核，其中也包括 无穷维度 的 多项式核函数。

12. AUC的含义？

AUC是指 从一堆样本中随机抽一个，抽到正样本的概率 比 抽到负样本的概率 大的可能性。

AUC的计算方法: roc曲线下方的面积积分即可, 或者大数定律的投点实验

13. C++实现kmeans, LR, softmax算法

分别参考

- [【机器学习实战之三】: C++实现K-均值\(K-Means\)聚类算法、](#)
- [Softmax回归C++实现](#)
- [c++实现logistic回归代码](#)

14. LR为什么用sigmoid函数。这个函数有什么优点和缺点? 为什么不用其他函数?

使用的原因:

答案1: **logistic**是基于**Bernoulli**分布的假设, 也就是 $y|X \sim \text{Bernoulli}$ 分布, 而**Bernoulli**分布的指数族的形式就是 $\frac{1}{(1+exp^{-z})}$

答案2: 对于logistic多分类而言,

x_1, x_2, \dots, x_n , 属于k类的概率正比于:

$$\exp(w_{k1}x_1 + w_{k2}x_2 + \dots + w_{kn}x_n)$$

我们回到2类:

x_1, x_2, \dots, x_n 属于1的概率是:

$$\frac{\exp(w_{11}x_1 + w_{12}x_2 + \dots + w_{1n}x_n)}{(\exp(w_{11}x_1 + w_{12}x_2 + \dots + w_{1n}x_n) + \exp(w_{01}x_1 + w_{02}x_2 + \dots + w_{0n}x_n))}$$

分子分母同除以分子即为 $\frac{1}{(1+exp^{-z})}$, $z = w_{11} - w_{01}$

优点:

1. 数据压缩能力, 将数据规约在 $[0, 1]$ 之间, 可以很好的模仿概率值。
1. 导数形式优秀, 方便计算。
1. 有阈值, 并且在阈值附近不会陡升陡降克服了阶跃函数的缺点。

缺点:

1. 容易梯度消失, x 稍大的情况下就趋近一条水平线
2. 非0中心化, 在神经网络算法等情况下, 造成反向传播时权重的全正全负的情况。

15. k-means的k怎么取, 初始中心点的选择方法, 改进

如果有真实数据可以进行聚类效果检验, 采用ARI选取当前最大值的 K作为标准。

如果是没有真实数据进行聚类效果检验, 则采用calinski_harabaz_score。

2.3.9.6. Calinski-Harabaz Index

If the ground truth labels are not known, the Calinski-Harabaz index (`sklearn.metrics.calinski_harabaz_score`) can be used to evaluate the model, where a higher Calinski-Harabaz score relates to a model with better defined clusters.

For k clusters, the Calinski-Harabaz score s is given as the ratio of the between-clusters dispersion mean and the within-cluster dispersion:

$$s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N - k}{k - 1}$$

where B_K is the between group dispersion matrix and W_K is the within-cluster dispersion matrix defined by:

$$W_k = \sum_{q=1}^k \sum_{x \in C_q} (x - c_q)(x - c_q)^T$$
$$B_k = \sum_q n_q (c_q - c)(c_q - c)^T$$

with N be the number of points in our data, C_q be the set of points in cluster q , c_q be the center of cluster q , c be the center of E , n_q be the number of points in cluster q .

1. 最简单的方法: $K \approx \sqrt{N/2}$
2. 数据的先验知识, 或者数据进行简单分析能得到
3. 基于结构的算法: 即比较类内距离、类间距离以确定 K 。这个也是最常用的办法, 如使用平均轮廓系数, 越趋近1聚类效果越好; 如计算类内距离/类间距离, 值越小越好; 等。
4. 基于一致性矩阵的算法: 即认为在正确的 K 时, 不同次聚类的结果会更加相似, 以此确定 K 。
5. 基于层次聚类: 即基于合并或分裂的思想, 在一定情况下停止从而获得 K 。
6. 基于采样的算法: 即对样本采样, 分别做聚类; 根据这些结果的相似性确定 K 。如, 将样本分为训练与测试样本; 对训练样本训练分类器, 用于预测测试样本类别, 并与聚类的类别比较。
7. 拐点法: 把聚类结果的F-test值 (类间Variance和全局Variance的比值) 对聚类个数的曲线画出来, 选择图中拐点
8. 基于Information Critieron的方法: 如果模型有似然函数 (如GMM), 用BIC、DIC等决策; 即使没有似然函数, 如KMean, 也可以搞一个假似然出来, 例如用GMM等来代替
9. 基于信息论的方法 (Jump法), 计算一个distortion函数对 K 值的曲线, 选择其中的jump点
10. 交叉验证
11. 特别地, 在文本中, 如果词频矩阵为 mn 维度, 其中 t 个不为0, 则 $K \approx mn/t$
12. 核方法: 构造Kernal矩阵, 对其做eigenvalue decomposition, 通过结果统计Compactness, 获得Compactness—K曲线, 选择拐点

初始类中心的选择:

- 1) 选择彼此距离尽可能远的 K 个点

首先随机选择一个点作为第一个初始类簇中心点, 然后选择距离该点最远的那个点作为第二个初始类簇中心点, 然后再选择距离前两个点的最近距离最大的点作为第三个初始类簇的中心点, 以此类推, 直至选出 K 个初始类簇中心点。

- 2) 先对数据用层次聚类算法或者Canopy算法进行聚类, 得到 K 个簇之后, 从每个类簇中选择一个点, 该点可以是该类簇的中心点, 或者是距离类簇中心点最近的那个点。

常用的层次聚类算法有BIRCH和ROCK, 在此不作介绍, 下面简单介绍一下Canopy算法, 主要摘自Mahout的Wiki:

首先定义两个距离 $T1$ 和 $T2$, $T1 > T2$ 。从初始的点的集合 S 中随机移除一个点 P , 然后对于还在 S 中的每个点 I , 计算该点 I 与点 P 的距离, 如果距离小于 $T1$, 则将点 I 加入到点 P 所代表的**Canopy**中, 如果距离小于 $T2$, 则将点 I 从集合 S 中移除, 并将点 I 加入到点 P 所代表的**Canopy**中。迭代完一次之后, 重新从集合 S 中随机选择一个点作为新的点 P , 然后重复执行以上步骤。

Canopy算法执行完毕后会得到很多**Canopy**, 可以认为每个**Canopy**都是一个**Cluster**, 与**KMeans**等硬划分算法不同, **Canopy**的聚类结果中每个点有可能属于多个**Canopy**。我们可以选择距离每个**Canopy**的中心点最近的那个数据点, 或者直接选择每个**Canopy**的中心点作为**KMeans**的初始**K**个类簇中心点。

不足和改进方法:

- 计算量大
 - KD树加速K-means
- 聚类数量**K**需要提前设定, 并影响聚类效果
 - 各种估计**K**的方法
- 聚类中心需要人为初始化, 并影响聚类效果
 - K-means++方法
 - 其他初始化聚类中心方法
- 异常点的存在, 会影响聚类效果
 - 数据预处理
- 只能收敛到局部最优
 - 未知

Kmeans++算法

k-means++算法选择初始seeds的基本思想就是: 初始的聚类中心之间的相互距离要尽可能的远。[wiki](#)上对该算法的描述是如下:

1. 从输入的数据点集合中随机选择一个点作为第一个聚类中心
2. 对于数据集中的每一个点 x , 计算它与最近聚类中心(指已选择的聚类中心)的距离 $D(x)$
3. 选择一个新的数据点作为新的聚类中心, 选择的原理是: $D(x)$ 较大的点, 被选取作为聚类中心的概率较大
4. 重复2和3直到**k**个聚类中心被选出来
5. 利用这**k**个初始的聚类中心来运行标准的k-means算法

这里为了避免噪声, 不能直接选取值最大的元素, 应该选择值较大的元素, 然后将其对应的数据点作为种子点。

从上面的算法描述上可以看到, 算法的关键是第3步, 如何将 $D(x)$ 反映到点被选择的概率上, 一种算法如下([详见此地](#)):

1. 先从我们的数据库随机挑个随机点当“种子点”
2. 对于每个点, 我们都计算其与最近的一个“种子点”的距离 $D(x)$ 并保存在一个数组里, 然后把这些距离加起来得到 $Sum(D(x))$ 。
3. 然后, 再取一个随机值, 用权重的方式来取计算下一个“种子点”。这个算法的实现是, 先取一个能落在 $Sum(D(x))$ 中的随机值 $Random$, 然后用 $Random \div D(x)$, 直到其 ≤ 0 , 此时的点就是下一个“种子点”。
4. 重复2和3直到**k**个聚类中心被选出来
5. 利用这**k**个初始的聚类中心来运行标准的k-means算法

实现代码可以参考文章[简单易学的机器学习算法——K-Means++算法](#), 其中关键的选择距离比较远的类中心代码如下:

```

def nearest(point, cluster_centers):
    min_dist = FLOAT_MAX
    m = np.shape(cluster_centers)[0] # 当前已经初始化的聚类中心的个数
    for i in xrange(m):
        # 计算point与每个聚类中心之间的距离
        d = distance(point, cluster_centers[i, ])
        # 选择最短距离
        if min_dist > d:
            min_dist = d
    return min_dist

def get_centroids(points, k):
    m, n = np.shape(points)
    cluster_centers = np.mat(np.zeros((k, n)))
    # 1、随机选择一个样本点为第一个聚类中心
    index = np.random.randint(0, m)
    cluster_centers[0, ] = np.copy(points[index, ])
    # 2、初始化一个距离的序列
    d = [0.0 for _ in xrange(m)]

    for i in xrange(1, k):
        sum_all = 0
        for j in xrange(m):
            # 3、对每一个样本找到最近的聚类中心点
            d[j] = nearest(points[j, ], cluster_centers[0:i, ])
            # 4、将所有的最短距离相加
            sum_all += d[j]
        # 5、取得sum_all之间的随机值
        sum_all *= random()
        # 6、获得距离最远的样本点作为聚类中心点
        for j, di in enumerate(d):
            sum_all -= di
            if sum_all > 0:
                continue
            cluster_centers[i] = np.copy(points[j, ])
            break
    return cluster_centers

```

参考文章

- [K-means中的K值选择与初始点的选择](#)
- [机器学习-KMeans聚类 K值以及初始类簇中心点的选取](#)
- [当我们在谈论K-means：总结](#)
- [K-Means++算法](#)
- [浅入浅出：从Kmeans到Kmeans++](#)
- [K-MEANS 算法](#)

16. Logistics推导

具体参见博客http://blog.csdn.net/ice_martin/article/details/61966790

对于Logistic regression, $h_{\theta}(x)$ 函数代表的是等于1的概率, 所以有如下的条件概率分布:

$$P(Y=1|x) = \frac{1}{1+e^{-\theta^T x}} = h_{\theta}(x)$$

$$P(Y=0|x) = 1 - \frac{1}{1+e^{-\theta^T x}} = 1 - h_{\theta}(x)$$

那么将两个式子合并起来写在一起就是:

$$P(Y|x) = h_{\theta}(x)^y (1 - h_{\theta}(x))^{1-y}$$

对上面这个式子求似然函数:

$$L(\theta) = \prod_{i=1}^m P(Y_i|x_i) = \prod_{i=1}^m h_{\theta}(x_i)^{y_i} (1 - h_{\theta}(x_i))^{1-y_i}$$

在对上面的似然函数求对数为:

$$l(\theta) = \log L(\theta) = \sum_{i=1}^m [y_i \log h_{\theta}(x_i) + (1 - y_i) \log(1 - h_{\theta}(x_i))]$$

如果求最优解则对上式求极大值时的 θ , 则此时运用的是梯度上升法, 但是在Andrew NG的课程中使用的是梯度下降算法, 故有:

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

17. 常见的正则化有什么, 有什么作用, 为什么L1是会把feature压缩到0而L2做不到?

(1) L1, L2正则化

L1对应python里面 `numpy.linalg.norm(ord=1)`, 形如 $|w_1| + |w_2| + |w_3| + \dots$

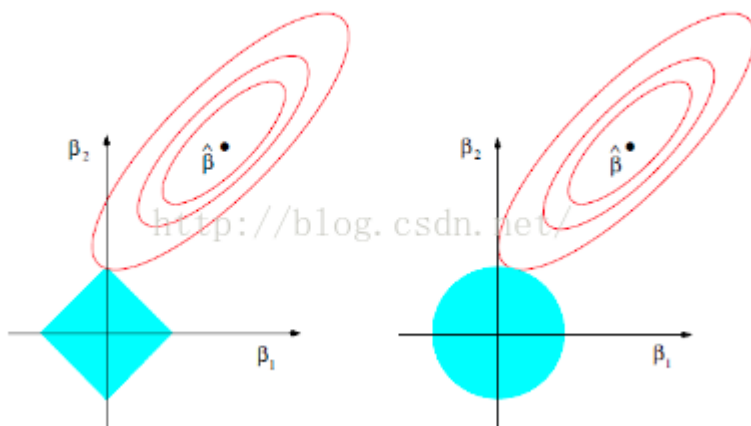
L2对应python里面 `numpy.linalg.norm(ord=2)`, 形如 $w_1^2 + w_2^2 + w_3^2 + \dots$

(2) 防止过拟合

其它防止过拟合的方法还有:

1. 增加数据量
2. 采取bagging算法, 抽样训练数据
3. 深度学习里面的Dropout, 早停机制
4. 交叉验证。

(3) 画图解决



左边的L1, 右边的L2.

L1的时候，只要不是特殊情况下与正方形的边相切，一定是与某个顶点优先相交，那必然存在横纵坐标轴中的一个系数为0，起到对变量的筛选的作用。

L2的时候，其实就可以看作是上面这个蓝色的圆，在这个圆的限制下，点可以是圆上的任意一点，所以 $q=2$ 的时候也叫做岭回归，岭回归是起不到压缩变量的作用的，在这个图里也是可以看出来。

18. 分类模型如何选择？如何判断效果？

(1) 整体上讲：数据量越大，神经网络越好；维度越多，**bagging**算法越优秀；数据量上不多不少的情况下，**SVM**效果最好；

(2) 常用判断：**roc**、**auc**、**ks**、**f1**值、**recall**等；

19. LR与随机森林的比较

相同点：

1. 都是监督学习算法；
2. 都是判别模型
3. 都是分类算法

不同点：

1. 随机森林的输出没有概率意义，但是LR有；
2. 随机森林有输入数据扰动和样本属性扰动；
3. 随机森林可以进行特征选择和处理有缺失值的特征，但LR不行；
4. 随机森林可以处理高维度特征，但LR不行；
5. 随机森林可以处理非线性特征，LR不能直接处理。

20. 过拟合的定义，原因和解决方法

参考文章 [过拟合（原因、解决方案、原理）](#)

定义：给定一个假设空间 H ，一个假设 h 属于 H ，如果存在其他的假设 h' 属于 H ，使得在训练样例上 h 的错误率比 h' 小，但在整个实例分布上 h' 比 h 的错误率小，那么就说假设 h 过度拟合训练数据。

一个假设在训练数据上能够获得比其他假设更好的拟合，但是在训练数据外的数据集上却不能很好的拟合数据。此时我们就叫这个假设出现了overfitting的现象。

原因：

- 建模样本抽取错误，包括（但不限于）样本数量太少，抽样方法错误，抽样时没有足够正确考虑业务场景或业务特点，等等导致抽出的样本数据不能有效足够代表业务逻辑或业务场景；
- 样本里的噪音数据干扰过大，大到模型过分记住了噪音特征，反而忽略了真实的输入输出间的关系；
- 建模时的“逻辑假设”到了模型应用时已经不能成立了。任何预测模型都是在假设的基础上才可以搭建和应用的，常用的假设包括：假设历史数据可以推测未来，假设业务环节没有发生显著变化，假设建模数据与后来的应用数据是相似的，等等。如果上述假设违反了业务场景的话，根据这些假设搭建的模型当然是无法有效应用的。
- 参数太多、模型复杂度高
- 决策树模型。如果我们对于决策树的生长没有合理的限制和修剪的话，决策树的自由生长有可能每片叶子里只包含单纯的事件数据(event)或非事件数据(no event)，可以想象，这种决策树当然可以完美匹配（拟合）训练数据，但是一旦应用到新的业务真实数据时，效果是一塌糊涂。
- 神经网络模型。
 - a. 由于对样本数据,可能存在隐单元的表示不唯一,即产生的分类的决策面不唯一.随着学习的进行, BP算法使权

值可能收敛过于复杂的决策面,并至极致.

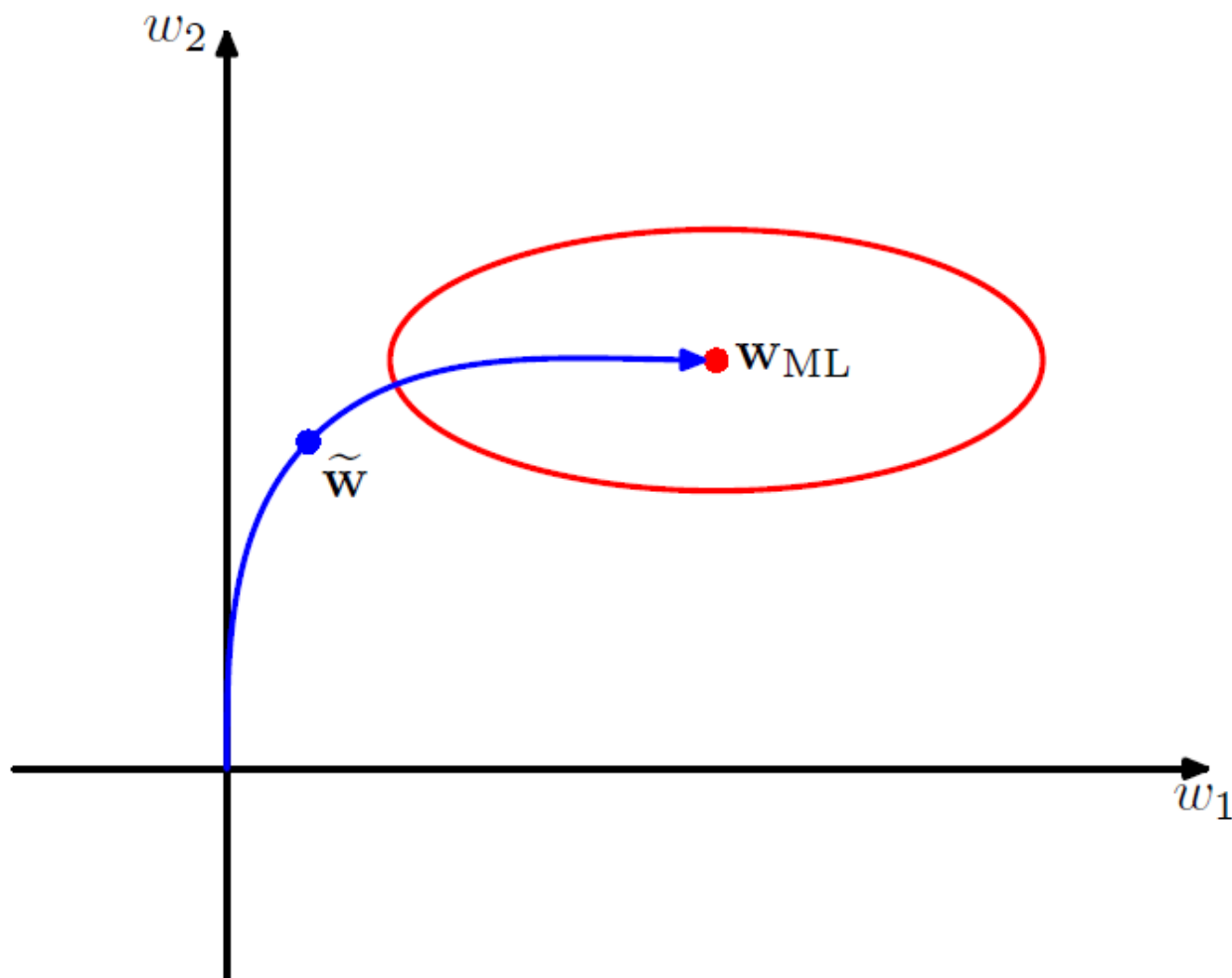
b.权值学习迭代次数足够多(Overtraining),拟合了训练数据中的噪声和训练样例中没有代表性的特征.

解决方法

(1) 权值衰减. 主要应用在神经网络模型中

它在每次迭代过程中以某个小因子降低每个权值,这等效于修改E的定义,加入一个与网络权值的总量相应的惩罚项,此方法的动机是保持权值较小,避免weight decay,从而使学习过程向着复杂决策面的反方向偏。

(2) 适当的stopping criterion



在二次误差函数的情况下,关于早停止和权值衰减类似结果的原因说明。椭圆给出了常数误差函数的轮廓线, \mathbf{W}_{ml} 表示误差函数的最小值。如果权向量的起始点为原点,按照局部负梯度的方向移动,那么它会沿着曲线给出的路径移动。通过对训练过程早停止,我们找到了一个权值向量 \mathbf{w} 。定性地说,它类似于使用检点的权值衰减正则化项,然后最小化正则化误差函数的方法得到的权值。

(3) 验证数据

一个最成功的方法是在训练数据外再为算法提供一套验证数据,应该使用在验证集合上产生最小误差的迭代次数,不是总能明显地确定验证集合何时达到最小误差。

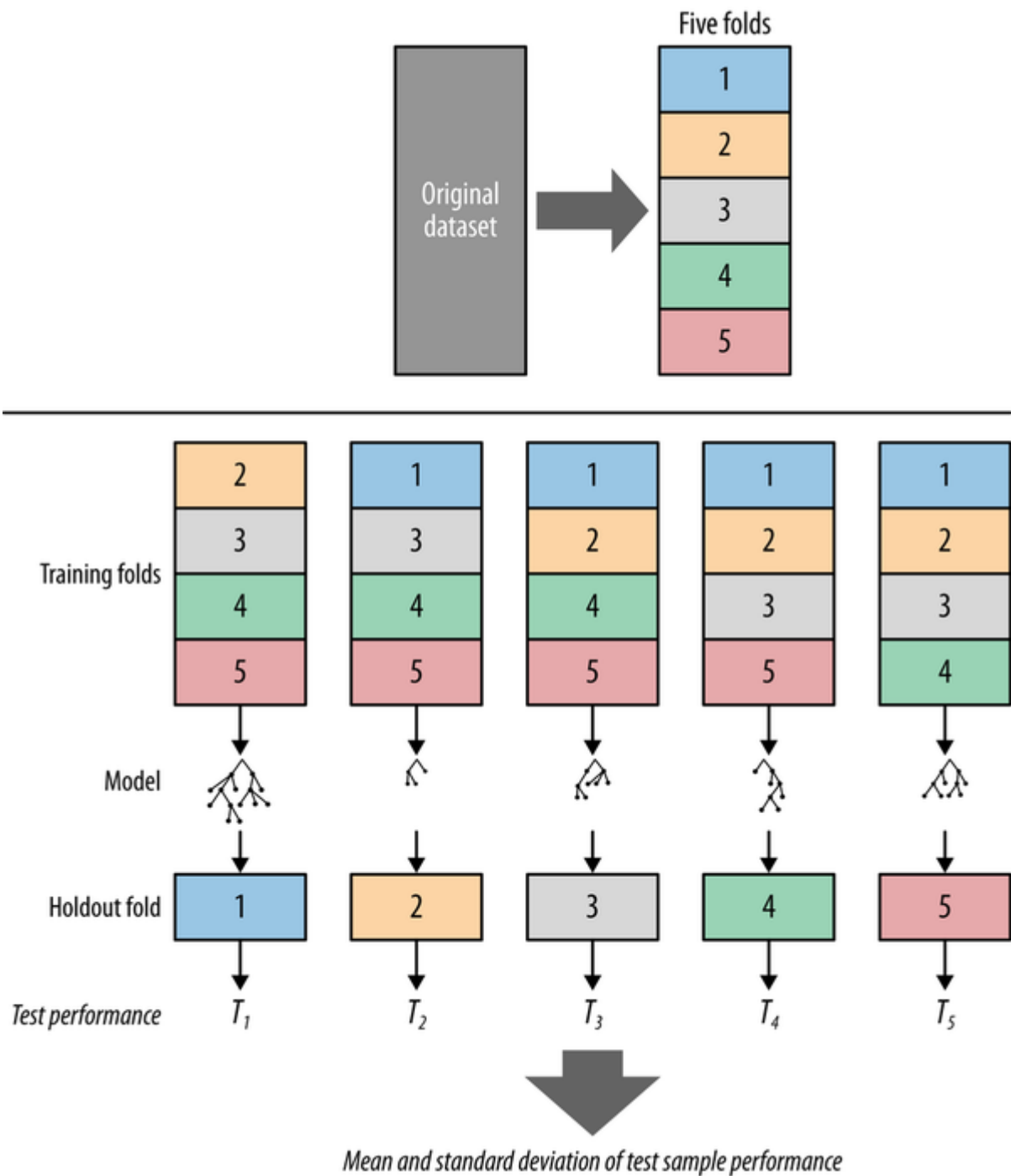
Typically 30% of training patterns; Validation set error is checked each epoch;

Stop training if validation error goes up

(4) 交叉验证

交叉验证方法在可获得额外的数据提供验证集合时工作得很好,但是小训练集合的过度拟合问题更为严重。

原理图



(5) 添加正则项。L1正则更加容易产生稀疏解、L2正则倾向于让参数 w 趋向于0.

(6) 针对树模型

a. 在树过于大之前便停止生长

每个叶中至少需要多少个数据 (threshold)

如何判断这个阈值 (threshold) 是重点【可以考虑用假设检验/P-值】

b. 等树生长到足够大之后进行修剪

修剪枝叶，直到任何改动都会降低正确率

Tip

- (1) 增加样本的全面性和数量;
- (2) 控制模型的复杂度;
- (3) 不要过度训练
- (4) 模型融合本质上也是一种提高泛化能力的方法

21. LR与贝叶斯的比较

相同:

1. 两者都是对特征的线性表达
2. 两者建模的都是条件概率, 对最终求得的分类结果有很好的解释性。

不同:

1. **Naive Bayes**是一个生成模型, 在计算 $P(y|x)$ 之前, 先要从训练数据中计算 $P(x|y)$ 和 $P(y)$ 的概率, 从而利用贝叶斯公式计算 $P(y|x)$ 。

Logistic Regression是一个判别模型, 它通过在训练数据集上最大化判别函数 $P(y|x)$ 学习得到, 不需要知道 $P(x|y)$ 和 $P(y)$ 。

2. **Naive Bayes**是建立在条件独立假设基础之上的, 设特征 X 含有 n 个特征属性 (X_1, X_2, \dots, X_n), 那么在给定 Y 的情况下, X_1, X_2, \dots, X_n 是条件独立的。

Logistic Regression的限制则要宽松很多, 如果数据满足条件独立假设, **Logistic Regression**能够取得非常好的效果; 当数据不满足条件独立假设时, **Logistic Regression**仍然能够通过调整参数让模型最大化的符合数据的分布, 从而训练得到在现有数据集下的一个最优模型。

3. 当数据集比较小的时候, 应该选用**Naive Bayes**, 为了能够取得很好的效果, 数据的需求量为 $O(\log n)$

22. 判别模型和生成模型的定义, 典型算法, 区别

生成方法是由数据学习联合概率分布 $P(X, Y)$, 然后求出条件概率分布 $P(Y|X)$ 作为预测的模型, 即生成模型, 之所以称为生成方法是因为模型表示了给定输入 X 产生输出 Y 的生成关系。典型的生成模型有: 朴素贝叶斯和隐马尔可夫模型。

判别方法由数据直接学习决策函数 $f(X)$ 或者条件概率分布 $P(Y|X)$ 作为预测的模型, 即判别模型。判别方法关心的是对给定输入的 X , 应该预测什么样的输出 Y 。典型的判别模型有: k 近邻法、感知机、决策树、逻辑回归、最大熵模型、支持向量机、提升方法和条件随机场等。

区别:

- 生成方法可以还原出联合概率分布 $P(X, Y)$, 但判别方法不能;
- 生成方法的学习收敛速度更快, 而判别方法直接学习的是决策函数 $f(X)$ 或者条件概率分布 $P(Y|X)$, 直接面对预测, 往往学习的准确率更高;
- 存在隐变量的时候, 可以用生成方法学习, 但不能用判别方法;
- 判别方法由于直接学习决策函数 $f(X)$ 或者条件概率分布 $P(Y|X)$, 可以对数据进行各种程度上的抽象、定义特征并使用特征, 因此可以简化学习问题

23. 如何进行特征选择?

特征选择是一个重要的数据预处理过程, 主要有两个原因, 首先在现实任务中我们会遇到维数灾难的问题(样本密度非常稀疏), 若能从中选择一部分特征, 那么这个问题能大大缓解, 另外就是去除不相关特征会降低学习任务的难度, 增加模型的泛化能力。*冗余特征指该特征包含的信息可以从其他特征中推演出来, 但是这并不代表该冗余特征一定没有作用, 例如在欠拟合的情况下也可以用过加入冗余特征*, 增加简单模型的复杂度。

在理论上如果没有任何领域知识作为先验假设那么只能遍历所有可能的子集。但是这显然是不可能的，因为需要遍历的数量是组合爆炸的。一般我们分为子集搜索和子集评价两个过程，子集搜索一般采用贪心算法，每一轮从候选特征中添加或者删除，分别成为前向和后先搜索。或者两者结合的双向搜索。子集评价一般采用信息增益，对于连续数据往往排序之后选择中点作为分割点。

常见的特征选择方式有过滤式，包裹式和嵌入式，`filter`,`wrapper`和`embedding`。

- **Filter**类型先对数据集进行特征选择，再训练学习器。
- **Wrapper**直接把最终学习器的性能作为特征子集的评价准则，一般通过不断候选子集，然后利用`cross-validation`过程更新候选特征，通常计算量比较大。
- 嵌入式特征选择将特征选择过程和训练过程融为了一体，在训练过程中自动进行了特征选择，例如L1正则化更易于获得稀疏解，而L2正则化更不容易过拟合。L1正则化可以通过PGD, 近端梯度下降进行求解。

24. 归一化和标准化的区别？

归一化：

- 1) 把数据变成(0 , 1)之间的小数
- 2) 把有量纲表达式变成无量纲表达

常见的有线性转换、对数函数转换、反余切函数转换等

标准化：

数据的标准化（**normalization**）是将数据按比例缩放，使之落入一个小的特定区间。在某些比较和评价的指标处理中经常会用到，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。

- 1) 最小—最大规范化(线性变换)

$$y = ((x - \text{MinValue}) / (\text{MaxValue} - \text{MinValue}))(\text{new_MaxValue} - \text{new_MinValue}) + \text{new_minValue}$$

- 2) z-score规范化(或零—均值规范化)

$$y = (x - X \text{ 的平均值}) / X \text{ 的标准差}$$

- 3) 小数定标规范化：通过移动X的小数位置来进行规范化

$$y = x / 10^j \quad (\text{其中 } j \text{ 使得 } \text{Max}(|y|) < 1 \text{ 的最小整数})$$

- 4) 对数Logistic模式：

$$\text{新数据} = 1 / (1 + e^{-(\text{原数据})})$$

- 5) 模糊量化模式

$$\text{新数据} = 1/2 + 1/2 \sin[3.1415 / (\text{极大值} - \text{极小值})]$$

25. 特征向量的缺失值处理

1. 缺失值较多.直接将该特征舍弃掉，否则可能反倒会带入较大的noise，对结果造成不良影响。
2. 缺失值较少,其余的特征缺失值都在10%以内，我们可以采取很多的方式来处理：
 - 把NaN直接作为一个特征，假设用0表示；
 - 用均值填充；

- 用随机森林等算法预测填充

26. 为什么一些机器学习模型需要对数据进行归一化？

参考文章[为什么一些机器学习模型需要对数据进行归一化？](#)

归一化就是要把你需要处理的数据经过处理后（通过某种算法）限制在你需要的一定范围内。

- 1) 归一化后加快了梯度下降求最优解的速度。等高线变得显得圆滑，在梯度下降进行求解时能较快的收敛。如果不做归一化，梯度下降过程容易走之字，很难收敛甚至不能收敛
- 2) 把有量纲表达式变为无量纲表达式，有可能提高精度。一些分类器需要计算样本之间的距离（如欧氏距离），例如KNN。如果一个特征值域范围非常大，那么距离计算就主要取决于这个特征，从而与实际情况相悖（比如这时实际情况是值域范围小的特征更重要）
- 3) 逻辑回归等模型先验假设数据服从正态分布。

归一化的类型有线性归一化、标准差归一化、非线性归一化

数据结构算法

1. 关于最短路径问题，给出 $n \times n$ 格子，格子上有不同数值，求格子上一**A**到点**B**的路径上，数值之和最短的路径？

2. 优先队列可以使用什么实现？

使用堆排序实现

3. 堆排序的建堆时间复杂度是？

$O(n)$ 。在构建堆的过程中，因为是从完全二叉树的最下层最右边的非叶结点开始构建，将它与其孩子进行比较和若有必要的交换，对每个非叶结点，最多进行两次比较和互换操作，这里需要进行这种操作的非叶结点数目是 $n/2$ 个。

4. 图的深度遍历和广度遍历的时间复杂度？

时间复杂度都是 $O(n^2)$ 或者 $O(n+e)$ ，后者是使用邻接表的复杂度， n 是顶点个数， e 个无向图边数，有向图中的弧数。

深度遍历一般使用栈，而广度遍历使用队列。

5. 爬虫如何实现？一天可以爬取数量？

(1)

(2) 一天可以爬取大约16-17万张图片，每分钟大约10张左右图片，10分钟大约115张图片，一个小时大约7000张图片。

6. 给出一个元素无序的数组，求出一个数，使得其左边的数都小于它，右边的数都大于等于它。

举例：1,2,3,1,2,0,5,6，返回下标6（数字为5）。

思路（1）：

朴素算法，对于每一个数，都检测它的左边和右边是否满足题意。

时间复杂度为 $O(n^2)$

思路（2）

使用变量求解：

（1）目前找到符合题意的候选值，nCandid

（2）目前已遍历数组的最大值，nMax：为了选下一次的候选值

（3）目前的候选值是否有效，blsExist：检测是否需要重新选择候选值

思路：如果候选值有效，可以继续遍历下面的数据

如果候选值小于目前的值，则该候选失效。之后遍历元素时，就要重新选择候选值

选择候选值时，对于某一个元素，如果该元素比之前遍历过元素的最大值还要大nMax，则该元素就为候选。

复杂度：遍历一遍数组即可，时间： $O(n)$ ，空间 $O(1)$

```

1.#include <iostream>
2.#include <assert.h>
3.#include <list>
4.using namespace std;
5.
6.int FindNum(int nArr[],int nLen)
7.{
8.    assert(nArr && nLen > 0);
9.    int nPos = 0;
10.    int nCandid = nArr[0];
11.    int nMax = nArr[0];
12.    bool bIsExist = true;
13.    for (int i = 1;i < nLen;i++)
14.    {
15.        if (bIsExist)//候选有效
16.        {
17.            if (nCandid > nArr[i])//候选失效
18.            {
19.                bIsExist = false;
20.            }
21.            else
22.            {
23.                nMax = nArr[i];
24.            }
25.        }
26.        else //候选失效
27.        {
28.            if (nArr[i] >= nMax)//重新找到候选
29.            {
30.                bIsExist = true;
31.
32.                nCandid = nArr[i];
33.                nMax = nArr[i];
34.                nPos = i;
35.            }
36.        }
37.    }
38.    return bIsExist ? nPos : -1;
39.}

```

7. 给定N张扑克牌和一个随机函数，设计一个洗牌算法

```
void shuffle(int cards[],int n)
{
    if(cards==NULL)
        return ;

    srand(time(0));

    for(int i=0;i<n-1;++i)
    {
        //保证每次第i位的值不会涉及到第i位以前
        int index=i+rand()%(n-i);
        int temp=cards[i];
        cards[i]=cards[index];
        cards[index]=temp;
    }
}
```

8. 寻找第k大的数

参考文章[寻找第k大的数](#)

这里列出使用二分法的代码。

```

#include <iostream>
using namespace std ;
const int N = 8 ;
const int K = 4 ;
int partition(int a[] ,int low , int high)
{
    int i = low - 1 ;
    int j = low;

    while(j < high)
    {
        if(a[j] >= a[high])
        {
            swap( a[i+1] , a[j]) ;
            i++ ;
        }
        j++ ;
    }
    //最后处理a[high]
    swap(a[i+1] , a[high]) ;
    return i + 1;
}

int findk(int a[] , int low , int high , int k)
{
    if(low < high)
    {
        int q = partition(a , low , high) ;

        int len = q - low + 1 ; //表示第几个位置
        if(len == k)
            return q ; //返回第k个位置
        else if(len < k)
            return findk(a , q + 1 , high , k - len) ;
        else
            return findk(a , low , q - 1, k ) ;
    }
}

int main()
{
    int a[N] = {5 , 2 , 66 , 23, 11 , 1 , 4 , 55} ;
    findk(a , 0 , N - 1 , K) ;

    for(int i = 0 ; i < K ; i++)
        cout<<a[i]<<endl ;

    system("pause") ;
    return 0 ;
}

```

9. 链表反转。递归和非递归方式

递归方式：

```
ListNode * ReverseList2(ListNode * head)
{
    //如果链表为空或者链表中只有一个元素
    if(head==NULL || head->m_pNext==NULL)
        return head;
    else
    {
        ListNode *newhead=ReverseList2(head->m_pNext); //先反转后面的链表
        head->m_pNext->m_pNext=head; //再将当前节点设置为其原来后面节点的后续节点，即实现了反转操作
        head->m_pNext=NULL;
        return newhead;
    }
}
```

非递归方式：

```
ListNode* ReverseList(ListNode* pHead) {
    ListNode* pNode = pHead;
    ListNode* pPrev = NULL;
    ListNode* pRes = NULL;
    while(pNode != NULL){
        ListNode* pNext = pNode->next;
        if(pNext == NULL)
            pRes = pNode;
        pNode->next = pPrev;
        pPrev = pNode;
        pNode = pNext;
    }
    return pRes;
}
```

C++/python 编程语言

1. C++和python的类的区别？

C++默认private，Python默认public；

C++可以实现多态，而python没有。

2. 堆区和栈区的区别

一个由C/C++编译的程序占用的内存分为以下几个部分

1. 栈区（**stack**）—— 由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于[数据结构](#)中的栈。
2. 堆区（**heap**）—— 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。
3. 全局区（静态区）（**static**）——全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。程序结束后由系统释

放。

4. 文字常量区 —常量字符串就是放在这里的。 程序结束后由系统释放
5. 程序代码区—存放函数体的二进制代码。

3. 多态的实现

- 多态是通过虚函数来实现，结合动态绑定。
- C++中虚函数使用虚函数表和 虚函数表指针实现，虚函数表是一个类的虚函数的地址表，用于索引类本身以及父类的虚函数的地址，假如子类的虚函数重写了父类的虚函数，则对应虚函数表中会把对应的虚函数替换为子类的虚函数的地址；虚函数表指针存在于每个对象中（通常出于效率考虑，会放在对象的开始地址处），它指向对象所在类的虚函数表的地址；在多继承环境下，会存在多个虚函数表指针，分别指向对应不同基类的虚函数表。

最后一个问题之你还有什么想问我的？

1. 我进去之后会做什么？
2. 团队是做什么东西的（业务是什么）？
3. 内部项目还是外部项目？
4. 就我之前的表现来看，你觉得我的优缺点在哪里？（这个问题可以侧面打探出他对你的评价，而且可以帮助你给自己查漏补缺）
5. 偏基础还是偏业务（简单粗暴地说，做基础就是写给程序员用的东西，做业务就是写给用户用的东西）？
6. 技术氛围怎么样？主要用到什么技术？有什么开源产出吗？你们做 **code review** 吗