**有没有一个相对清晰的分界线，哪些情况下就可以使用机器学习，哪些不可以呢？（4年java开发提问）答：在神经网络出现之前，对于特征抽取比较准确的领域都可以使用机器学习。在神经网络出现之后，尤其是卷积神经网络出现之后，对于模式识别方面的提高非常强大。很多原来使用随机森林或者SVM都无法处理好的模式识别问题得到了较好的解决。如果一定要划界限的话，可以从特征提取的难易程度上来划分。如果特征很难提取，则很难在机器学习过程中得到令人满意的结果。**

　　scikit-learn对于线性回归提供了比较多的类库，这些类库都可以用来做线性回归分析，本文就对这些类库的使用做一个总结，重点讲述这些线性回归算法库的不同和各自的使用场景。

　　　　线性回归的目的是要得到输出向量YY和输入特征XX之间的线性关系，求出线性回归系数θθ,也就是 Y=XθY=Xθ。其中YY的维度为mx1，XX的维度为mxn，而θθ的维度为nx1。m代表样本个数，n代表样本特征的维度。

　　　　为了得到线性回归系数θθ，我们需要定义一个损失函数，一个极小化损失函数的优化方法，以及一个验证算法的方法。损失函数的不同，损失函数的优化方法的不同，验证方法的不同，就形成了不同的线性回归算法。scikit-learn中的线性回归算法库可以从这这三点找出各自的不同点。理解了这些不同点，对不同的算法使用场景也就好理解了。

# **1. LinearRegression**

**损失函数：**

　　　　LinearRegression类就是我们平时说的最常见普通的线性回归，它的损失函数也是最简单的，如下：

　　　　J(θ)=12(Xθ−Y)T(Xθ−Y)J(θ)=12(Xθ−Y)T(Xθ−Y)

**损失函数的优化方法：**

　　　　对于这个损失函数，一般有梯度下降法和最小二乘法两种极小化损失函数的优化方法，而scikit中的LinearRegression类用的是最小二乘法。通过最小二乘法，可以解出线性回归系数θθ为：

　　　　θ=(XTX)−1XTYθ=(XTX)−1XTY

**验证方法：**

　　　　LinearRegression类并没有用到交叉验证之类的验证方法，需要我们自己把数据集分成训练集和测试集，然后训练优化。

**使用场景：**

　　　　一般来说，只要我们觉得数据有线性关系，LinearRegression类是我们的首先。如果发现拟合或者预测的不好，再考虑用其他的线性回归库。如果是学习线性回归，推荐先从这个类开始第一步的研究。

# **2. Ridge**

**损失函数：**

　　　　由于第一节的LinearRegression没有考虑过拟合的问题，有可能泛化能力较差，这时损失函数可以加入正则化项，如果加入的是L2范数的正则化项，这就是Ridge回归。损失函数如下：

　　　　J(θ)=12(Xθ−Y)T(Xθ−Y)+12α||θ||22J(θ)=12(Xθ−Y)T(Xθ−Y)+12α||θ||22

　　　　其中αα为常数系数，需要进行调优。||θ||2||θ||2为L2范数。

　　　　Ridge回归在不抛弃任何一个特征的情况下，缩小了回归系数，使得模型相对而言比较的稳定，不至于过拟合。

**损失函数的优化方法：**

　　　　对于这个损失函数，一般有梯度下降法和最小二乘法两种极小化损失函数的优化方法，而scikit中的Ridge类用的是最小二乘法。通过最小二乘法，可以解出线性回归系数θθ为：

　　　　θ=(XTX+αE)−1XTYθ=(XTX+αE)−1XTY

　　　　其中E为单位矩阵。

**验证方法：**

　　　　Ridge类并没有用到交叉验证之类的验证方法，需要我们自己把数据集分成训练集和测试集，需要自己设置好超参数αα。然后训练优化。

**使用场景：**

　　　　一般来说，只要我们觉得数据有线性关系，用LinearRegression类拟合的不是特别好，需要正则化，可以考虑用Ridge类。但是这个类最大的缺点是每次我们要自己指定一个超参数αα，然后自己评估αα的好坏，比较麻烦，一般我都用下一节讲到的RidgeCV类来跑Ridge回归，不推荐直接用这个Ridge类，除非你只是为了学习Ridge回归。

# **3. RidgeCV**

　　　　RidgeCV类的损失函数和损失函数的优化方法完全与Ridge类相同，区别在于验证方法。

**验证方法：**

　　　　RidgeCV类对超参数αα使用了交叉验证，来帮忙我们选择一个合适的αα。在初始化RidgeCV类时候，我们可以传一组备选的αα值，10个，100个都可以。RidgeCV类会帮我们选择一个合适的αα。免去了我们自己去一轮轮筛选αα的苦恼。

**使用场景：**

　　　　一般来说，只要我们觉得数据有线性关系，用LinearRegression类拟合的不是特别好，需要正则化，可以考虑用RidgeCV类。不是为了学习的话就不用Ridge类。为什么这里只是考虑用RidgeCV类呢？因为线性回归正则化有很多的变种，Ridge只是其中的一种。所以可能需要比选。如果输入特征的维度很高，而且是稀疏线性关系的话，RidgeCV类就不合适了。这时应该主要考虑下面几节要讲到的Lasso回归类家族。

# **4.  Lasso**

**损失函数：**

　　　　线性回归的L1正则化通常称为Lasso回归，它和Ridge回归的区别是在损失函数上增加了的是L1正则化的项，而不是L2正则化项。L1正则化的项也有一个常数系数αα来调节损失函数的均方差项和正则化项的权重，具体Lasso回归的损失函数表达式如下：

　　　　J(θ)=12m(Xθ−Y)T(Xθ−Y)+α||θ||1J(θ)=12m(Xθ−Y)T(Xθ−Y)+α||θ||1

　　　　其中n为样本个数，αα为常数系数，需要进行调优。||θ||1||θ||1为L1范数。

　　　　Lasso回归可以使得一些特征的系数变小，甚至还是一些绝对值较小的系数直接变为0。增强模型的泛化能力。

**损失函数的优化方法：**

　　　　Lasso回归的损失函数优化方法常用的有两种，坐标轴下降法和最小角回归法。Lasso类采用的是坐标轴下降法，后面讲到的LassoLars类采用的是最小角回归法

**验证方法：**

　　　　Lasso类并没有用到交叉验证之类的验证方法，和Ridge类类似。需要我们自己把数据集分成训练集和测试集，需要自己设置好超参数αα。然后训练优化。

**使用场景：**

一般来说，对于高维的特征数据，尤其线性关系是稀疏的，我们会采用Lasso回归。或者是要在一堆特征里面找出主要的特征，那么Lasso回归更是首选了。但是Lasso类需要自己对αα调优，所以不是Lasso回归的首选，一般用到的是下一节要讲的LassoCV类。

# **5. LassoCV**

　　　　LassoCV类的损失函数和损失函数的优化方法完全与Lasso类相同，区别在于验证方法。

**验证方法：**

　　　　LassoCV类对超参数αα使用了交叉验证，来帮忙我们选择一个合适的αα。在初始化LassoCV类时候，我们可以传一组备选的αα值，10个，100个都可以。LassoCV类会帮我们选择一个合适的αα。免去了我们自己去一轮轮筛选αα的苦恼。

**使用场景：**

　　　　LassoCV类是进行Lasso回归的首选。当我们面临在一堆高位特征中找出主要特征时，LassoCV类更是必选。当面对稀疏线性关系时，LassoCV也很好用。

# **6. LassoLars**

　　　　LassoLars类的损失函数和验证方法与Lasso类相同，区别在于损失函数的优化方法。

**损失函数的优化方法：**

　　　　Lasso回归的损失函数优化方法常用的有两种，坐标轴下降法和最小角回归法。LassoLars类采用的是最小角回归法，前面讲到的Lasso类采用的是坐标轴下降法。

**使用场景：**

LassoLars类需要自己对αα调优，所以不是Lasso回归的首选，一般用到的是下一节要讲的LassoLarsCV类。

# **7. LassoLarsCV**

　　　　LassoLarsCV类的损失函数和损失函数的优化方法完全与LassoLars类相同，区别在于验证方法。

**验证方法：**

　　　　LassoLarsCV类对超参数αα使用了交叉验证，来帮忙我们选择一个合适的αα。在初始化LassoLarsCV类时候，我们可以传一组备选的αα值，10个，100个都可以。LassoLarsCV类会帮我们选择一个合适的αα。免去了我们自己去一轮轮筛选αα的苦恼。

**使用场景：**

　　　　LassoLarsCV类是进行Lasso回归的第二选择。第一选择是前面讲到LassoCV类。那么LassoLarsCV类有没有适用的场景呢？换句话说，用最小角回归法什么时候比坐标轴下降法好呢？场景一：如果我们想探索超参数αα更多的相关值的话，由于最小角回归可以看到回归路径，此时用LassoLarsCV比较好。场景二： 如果我们的样本数远小于样本特征数的话，用LassoLarsCV也比LassoCV好。其余场景最好用LassoCV。

# **8. LassoLarsIC**

　　　　LassoLarsIC类的损失函数和损失函数的优化方法完全与LassoLarsCV类相同，区别在于验证方法。

**验证方法：**

　　　　LassoLarsIC类对超参数αα没有使用交叉验证，而是用 Akaike信息准则(AIC)和贝叶斯信息准则(BIC)。此时我们并不需要指定备选的αα值，而是由LassoLarsIC类基于AIC和BIC自己选择。用LassoLarsIC类我们可以一轮找到超参数αα，而用K折交叉验证的话，我们需要K+1轮才能找到。相比之下LassoLarsIC类寻找αα更快。

**使用场景：**

　　　　从验证方法可以看出，验证ααLassoLarsIC比LassoLarsCV快很多。那么是不是LassoLarsIC类一定比LassoLarsCV类好呢？ 不一定！由于使用了AIC和BIC准则，我们的数据必须满足一定的条件才能用LassoLarsIC类。这样的准则需要对解的自由度做一个适当的估计。该估计是来自大样本（渐近结果），并假设该模型是正确的（即这些数据确实是由假设的模型产生的）。当待求解的问题的条件数很差的时候（比如特征个数大于样本数量的时候），这些准则就会有崩溃的风险。所以除非我们知道数据是来自一个模型确定的大样本，并且样本数量够大，我们才能用LassoLarsIC。而实际上我们得到的数据大部分都不能满足这个要求，实际应用中我没有用到过这个看上去很美的类。

# **9.  ElasticNet**

**损失函数：**

　　　　ElasticNet可以看做Lasso和Ridge的中庸化的产物。它也是对普通的线性回归做了正则化，但是它的损失函数既不全是L1的正则化，也不全是L2的正则化，而是用一个权重参数ρρ来平衡L1和L2正则化的比重，形成了一个全新的损失函数如下：

　　　　J(θ)=12m(Xθ−Y)T(Xθ−Y)+αρ||θ||1+α(1−ρ)2||θ||22J(θ)=12m(Xθ−Y)T(Xθ−Y)+αρ||θ||1+α(1−ρ)2||θ||22

　　　　其中αα为正则化超参数，ρρ为范数权重超参数。

**损失函数的优化方法：**

　　　　ElasticNet回归的损失函数优化方法常用的有两种，坐标轴下降法和最小角回归法。ElasticNet类采用的是坐标轴下降法。

**验证方法：**

　　　　ElasticNet类并没有用到交叉验证之类的验证方法，和Lasso类类似。需要我们自己把数据集分成训练集和测试集，需要自己设置好超参数αα和ρρ。然后训练优化。

**使用场景：**

ElasticNet类需要自己对αα和ρρ调优，所以不是ElasticNet回归的首选，一般用到的是下一节要讲的ElasticNetCV类。

# **10. ElasticNetCV**

　　　　ElasticNetCV类的损失函数和损失函数的优化方法完全与ElasticNet类相同，区别在于验证方法。

**验证方法：**

　　　　ElasticNetCV类对超参数αα和 ρρ使用了交叉验证，来帮忙我们选择合适的αα和ρρ。在初始化ElasticNetCV类时候，我们可以传一组备选的αα值和ρρ，10个，100个都可以。ElasticNetCV类会帮我们选择一个合适的αα和ρρ。免去了我们自己去一轮轮筛选αα和ρρ的苦恼。

**使用场景：**

　　　　ElasticNetCV类用在我们发现用Lasso回归太过（太多特征被稀疏为0），而用Ridge回归又正则化的不够（回归系数衰减的太慢）的时候。一般不推荐拿到数据就直接就上ElasticNetCV。

# **11. OrthogonalMatchingPursuit**

**损失函数：**

OrthogonalMatchingPursuit（OMP正交匹配追踪）算法和普通的线性回归损失函数的区别是增加了一个限制项，来限制回归系数中非0元素的最大个数。形成了一个全新的损失函数如下：

　　　　J(θ)=12(Xθ−Y)T(Xθ−Y)J(θ)=12(Xθ−Y)T(Xθ−Y)

　　　　subject to ||θ||0≤nnon−zero−coefs||θ||0≤nnon−zero−coefs ,其中(||θ||0(||θ||0代表θθ的L0范数，即非0回归系数的个数。

**损失函数的优化方法：**

　　　　OrthogonalMatchingPursuit类使用前向选择算法来优化损失函数。它是最小角回归算法的缩水版。虽然精度不如最小角回归算法，但是运算速度很快。

**验证方法：**

　　　　OrthogonalMatchingPursuit类并没有用到交叉验证之类的验证方法，和Lasso类类似。需要我们自己把数据集分成训练集和测试集，需要自己选择限制参数nnon−zero−coefsnnon−zero−coefs。然后训练优化。

**使用场景：**

OrthogonalMatchingPursuit类需要自己选择nnon−zero−coefsnnon−zero−coefs，所以不是OrthogonalMatchingPursuit回归的首选，一般用到的是下一节要讲的OrthogonalMatchingPursuitCV类，不过如果你已经定好了nnon−zero−coefsnnon−zero−coefs的值，那用OrthogonalMatchingPursuit比较方便。

# **12. OrthogonalMatchingPursuitCV**

　　　　OrthogonalMatchingPursuitCV类的损失函数和损失函数的优化方法完全与OrthogonalMatchingPursuit类相同，区别在于验证方法。

**验证方法：**

　　　　OrthogonalMatchingPursuitCV类使用交叉验证，在S折交叉验证中以MSE最小为标准来选择最好的nnon−zero−coefsnnon−zero−coefs。

**使用场景：**

　　　　OrthogonalMatchingPursuitCV类通常用在稀疏回归系数的特征选择上，这点和LassoCV有类似的地方。不过由于它的损失函数优化方法是前向选择算法，精确度较低，一般情况不是特别推荐用，用LassoCV就够，除非你对稀疏回归系数的精确个数很在意，那可以考虑用OrthogonalMatchingPursuitCV。

# **13.  MultiTaskLasso**

　　　　从这节到第16节，类里面都带有一个“MultiTask”的前缀。不过他不是编程里面的多线程，而是指多个线性回归模型共享样本特征，但是有不同的回归系数和特征输出。具体的线性回归模型是Y=XWY=XW。其中X是mxn维度的矩阵。W为nxk维度的矩阵，Y为mxk维度的矩阵。m为样本个数，n为样本特征，而k就代表多个回归模型的个数。所谓的“MultiTask”这里其实就是指k个线性回归的模型一起去拟合。

**损失函数：**

　　　　由于这里是多个线性回归一起拟合，所以损失函数和前面的都很不一样：

　　　　J(W)=12m(||XW−Y||)2Fro+α||W||21J(W)=12m(||XW−Y||)Fro2+α||W||21

　　　　其中， (||XW−Y||)Fro(||XW−Y||)Fro是Y=XWY=XW的Frobenius范数。而||W||21||W||21代表W的各列的根平方和之和。

**损失函数的优化方法：**

　　　　MultiTaskLasso类使用坐标轴下降法来优化损失函数。

**验证方法：**

　　　　MultiTaskLasso类并没有用到交叉验证之类的验证方法，和Lasso类类似。需要我们自己把数据集分成训练集和测试集，需要自己设置好超参数αα。然后训练优化。

**使用场景：**

MultiTaskLasso类需要自己对αα调优，所以不是共享特征协同回归的首选，一般用到的是下一节要讲的MultiTaskLassoCV类。

# **14.  MultiTaskLassoCV**

　　　　MultiTaskLassoCV类的损失函数和损失函数的优化方法完全与MultiTaskLasso类相同，区别在于验证方法。

**验证方法：**

　　　　MultiTaskLassoCV类对超参数αα使用了交叉验证，来帮忙我们选择一个合适的αα。在初始化LassoLarsCV类时候，我们可以传一组备选的αα值，10个，100个都可以。MultiTaskLassoCV类会帮我们选择一个合适的αα。

**使用场景：**

　　　　MultiTaskLassoCV是多个回归模型需要一起共享样本特征一起拟合时候的首选。它可以保证选到的特征每个模型都用到。不会出现某个模型选到了某特征而另一个模型没选到这个特征的情况。

# **15.  MultiTaskElasticNet**

**损失函数：**

　　　　MultiTaskElasticNet类和MultiTaskLasso类的模型是相同的。不过损失函数不同。损失函数表达式如下：

　　　　J(W)=12m(||XW−Y||)2Fro+αρ||W||21+α(1−ρ)2(||W||)2FroJ(W)=12m(||XW−Y||)Fro2+αρ||W||21+α(1−ρ)2(||W||)Fro2

　　　　其中， (||XW−Y||)Fro(||XW−Y||)Fro是Y=XWY=XW的Frobenius范数。而||W||21||W||21代表W的各列的根平方和之和。

**损失函数的优化方法：**

　　　　MultiTaskElasticNet类使用坐标轴下降法来优化损失函数。

**验证方法：**

　　　　MultiTaskElasticNet类并没有用到交叉验证之类的验证方法，和Lasso类类似。需要我们自己把数据集分成训练集和测试集，需要自己设置好超参数αα和ρρ。然后训练优化。

**使用场景：**

MultiTaskElasticNet类需要自己对αα调优，所以不是共享特征协同回归的首选，如果需要用MultiTaskElasticNet，一般用到的是下一节要讲的MultiTaskElasticNetCV类。

# **16.  MultiTaskElasticNetCV**

　　　　MultiTaskElasticNetCV类的损失函数和损失函数的优化方法完全与MultiTaskElasticNet类相同，区别在于验证方法。

**验证方法：**

　　　　MultiTaskElasticNetCV类对超参数α和 ρ使用了交叉验证，来帮忙我们选择合适的α和ρ。在初始化MultiTaskElasticNetCV类时候，我们可以传一组备选的α值和ρ，10个，100个都可以。ElasticNetCV类会帮我们选择一个合适的αα和ρρ。免去了我们自己去一轮轮筛选αα和ρρ的苦恼。

**使用场景：**

　　　　MultiTaskElasticNetCV是多个回归模型需要一起共享样本特征一起拟合时候的两个备选之一，首选是MultiTaskLassoCV。如果我们发现用MultiTaskLassoCV时回归系数衰减的太快，那么可以考虑用MultiTaskElasticNetCV。

# **17. BayesianRidge**

　　　　第17和18节讲的都是贝叶斯回归模型。贝叶斯回归模型假设先验概率，似然函数和后验概率都是正态分布。先验概率是假设模型输出Y是符合均值为Xθ的正态分布，正则化参数α被看作是一个需要从数据中估计得到的随机变量。回归系数θ的先验分布规律为球形正态分布，超参数为λ。我们需要通过最大化边际似然函数来估计超参数α和λ，以及回归系数θ。

　　　　此处对损失函数即负的最大化边际似然函数不多讨论，不过其形式和Ridge回归的损失函数很像，所以也取名BayesianRidge。

**使用场景：**

如果我们的数据有很多缺失或者矛盾的病态数据，可以考虑BayesianRidge类，它对病态数据鲁棒性很高，也不用交叉验证选择超参数。但是极大化似然函数的推断过程比较耗时，一般情况不推荐使用。

# **18. ARDRegression**

　　　　ARDRegression和BayesianRidge很像，唯一的区别在于对回归系数θθ的先验分布假设。BayesianRidge假设θθ的先验分布规律为球形正态分布，而ARDRegression丢掉了BayesianRidge中的球形高斯的假设，采用与坐标轴平行的椭圆形高斯分布。这样对应的超参数λλ有n个维度，各不相同。而上面的BayesianRidge中球形分布的θθ对应的λλ只有一个。

　　　　ARDRegression也是通过最大化边际似然函数来估计超参数αα和λλ向量，以及回归系数θθ。

**使用场景：**

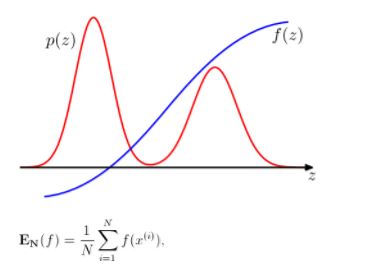
如果我们的数据有很多缺失或者矛盾的病态数据，可以考虑BayesianRidge类，如果发现拟合不好，可以换ARDRegression试一试。因为ARDRegression对回归系数先验分布的假设没有BayesianRidge严格，某些时候会比BayesianRidge产生更好的后验结果。

# **贝叶斯线性回归的引入主要是在最大似然估计中很难决定模型的复杂程度，ridge回归加入的惩罚参数其实也是解决这个问题的，同时可以采用的方法还有对数据进行正规化处理，另一个可以解决此问题的方法就是采用贝叶斯方法。**

# **贝叶斯计算广泛采用：[MCMC(Markov Chain Monte](http://www.cnblogs.com/ywl925/archive/2013/06/05/3118875.html)** **[Carlo) and Gibbs Sampling](http://www.cnblogs.com/ywl925/archive/2013/06/05/3118875.html) 随机模拟与吉布斯采样**

**马氏链收敛，与初始概率分布无关，只与转换矩阵有关**

**蒙特卡洛原理：随机抽样取平均值。Monte Carlo 抽样计算随即变量的期望值是接下来内容的重点：X 表示随即变量，服从概率分布 p(x), 那么要计算 f(x) 的期望，只需要我们不停从 p(x) 中抽样xi，然后对这些f（xi）取平均即可近似f(x)的期望。**



**没有最好的分类器，只有最合适的分类器。**

随机森林平均来说最强，但也只在9.9%的数据集上拿到了第一，优点是鲜有短板。

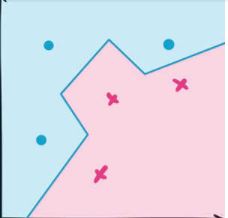
SVM的平均水平紧随其后，在10.7%的数据集上拿到第一。

神经网络（13.2%）和boosting（~9%）表现不错。

**数据维度越高**，随机森林就比AdaBoost强越多，但是整体不及SVM[2]。

**数据量越大**，神经网络就越强。

**1,近邻 (Nearest Neighbor)**



典型的例子是KNN，它的思路就是——对于待判断的点，找到离它最近的几个数据点，根据它们的类型决定待判断点的类型。

它的特点是完全跟着数据走，没有数学模型可言。

适用情景：

需要一个特别容易解释的模型的时候。

比如需要向用户解释原因的推荐算法。

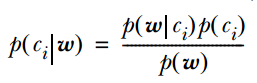
## **贝叶斯 (Bayesian)**

典型的例子是Naive Bayes，核心思路是根据条件概率计算待判断点的类型。

是相对容易理解的一个模型，至今依然被垃圾邮件过滤器使用。

　1. 如果给出的特征向量长度可能不同，这是需要归一化为通长度的向量（这里以文本分类为例），比如说是句子单词的话，则长度为整个词汇量的长度，对应位置是该单词出现的次数。

　　2. 计算公式如下：



　　其中一项条件概率可以通过朴素贝叶斯条件独立展开。要注意一点就是 IMG_257的计算方法，而由朴素贝叶斯的前提假设可知，IMG_258 =IMG_259 ，因此一般有两种，一种是在类别为ci的那些样本集中，找到wj出现次数的总和，然后除以该样本的总和；第二种方法是类别为ci的那些样本集中，找到wj出现次数的总和，然后除以该样本中所有特征出现次数的总和。

　　3. 如果 IMG_260中的某一项为0，则其联合概率的乘积也可能为0，即2中公式的分子为0，为了避免这种现象出现，一般情况下会将这一项初始化为1，当然为了保证概率相等，分母应对应初始化为2（这里因为是2类，所以加2，如果是k类就需要加k，术语上叫做laplace光滑, 分母加k的原因是使之满足全概率公式）。

**朴素贝叶斯的优点：**

　　对小规模的数据表现很好，适合多分类任务，适合增量式训练。

　　缺点**：**

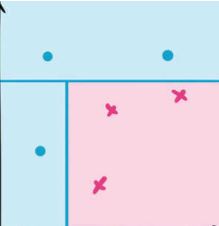
　　对输入数据的表达形式很敏感。

适用情景：

需要一个比较容易解释，而且不同维度之间相关性较小的模型的时候。

可以高效处理高维数据，虽然结果可能不尽如人意。

## **决策树 (Decision tree)**



决策树的特点是它总是在沿着特征做切分。随着层层递进，这个划分会越来越细。

虽然生成的树不容易给用户看，但是数据分析的时候，通过观察树的上层结构，能够对分类器的核心思路有一个直观的感受。

举个简单的例子，当我们预测一个孩子的身高的时候，决策树的第一层可能是这个孩子的性别。男生走左边的树进行进一步预测，女生则走右边的树。这就说明性别对身高有很强的影响。

决策树可以认为是将空间进行划分，ID3和C4.5算是比较经典的决策树算法，可以用来分类，也可以用来回归，但业界很少直接使用一棵树，一般使用多棵树，组成committee，较为经典有GBDT 和RF，两者都是ensemble learning的典范，只不过前者使用boosting降低bias，后者使用bagging降低variance从而提升模型的performance。在ESL中有个对比，使树形模型几乎完爆其他算法，泛化能力和学习能力都很牛逼。业界的话一般用来做搜索排序和相关性。

**决策树：**

　　决策树中很重要的一点就是选择一个属性进行分枝，因此要注意一下信息增益的计算公式，并深入理解它。

　　信息熵的计算公式如下:

IMG_256

　　其中的n代表有n个分类类别（比如假设是2类问题，那么n=2）。分别计算这2类样本在总样本中出现的概率p1和p2，这样就可以计算出未选中属性分枝前的信息熵。

　　现在选中一个属性xi用来进行分枝，此时分枝规则是：如果xi=vx的话，将样本分到树的一个分支；如果不相等则进入另一个分支。很显然，分支中的样本很有可能包括2个类别，分别计算这2个分支的熵H1和H2,计算出分枝后的总信息熵H’=p1\*H1+p2\*H2.，则此时的信息增益ΔH=H-H’。以信息增益为原则，把所有的属性都测试一边，选择一个使增益最大的属性作为本次分枝属性。

**决策树的优点：**

　　计算量简单，可解释性强，比较适合处理有缺失属性值的样本，能够处理不相关的特征；

**缺点：**

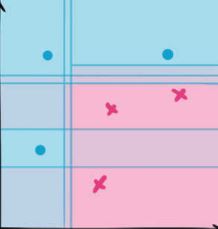
容易过拟合（后续出现了随机森林，减小了过拟合现象）；  
适用情景：

因为它能够生成清晰的基于特征(feature)选择不同预测结果的树状结构，数据分析师希望更好的理解手上的数据的时候往往可以使用决策树。

同时它也是相对容易被攻击的分类器[3]。这里的攻击是指人为的改变一些特征，使得分类器判断错误。常见于垃圾邮件躲避检测中。因为决策树最终在底层判断是基于单个条件的，攻击者往往只需要改变很少的特征就可以逃过监测。

受限于它的简单性，决策树更大的用处是作为一些更有用的算法的基石。

## **随机森林 (Random forest)**



提到决策树就不得不提随机森林。顾名思义，森林就是很多树。

严格来说，随机森林其实算是一种集成算法。它首先随机选取不同的特征(feature)和训练样本(training sample)，生成大量的决策树，然后综合这些决策树的结果来进行最终的分类。

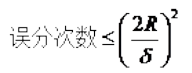
随机森林在现实分析中被大量使用，它相对于决策树，在准确性上有了很大的提升，同时一定程度上改善了决策树容易被攻击的特点。

适用情景：

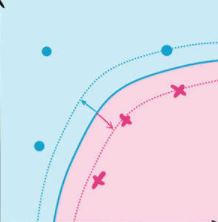
数据维度相对低（几十维），同时对准确性有较高要求时。

因为不需要很多参数调整就可以达到不错的效果，基本上不知道用什么方法的时候都可以先试一下随机森林。

## 捕获**SVM (Support vector machine)**

1. svm中的最优分类面是对所有样本的几何裕量最大（为什么要选择最大间隔分类器，请从数学角度上说明？网易深度学习岗位面试过程中有被问到。答案就是几何间隔与样本的误分次数间存在关系： ，其中的分母就是样本到分类间隔距离，分子中的R是所有样本中的最长向量值），即：

SVM的核心思想就是找到不同类别之间的分界面，使得两类样本尽量落在面的两边，而且离分界面尽量远。

最早的SVM是平面的，局限很大。但是利用核函数(kernel function)，我们可以把平面投射(mapping)成曲面，进而大大提高SVM的适用范围。

提高之后的SVM同样被大量使用，在实际分类中展现了很优秀的正确率。

适用情景：

SVM在很多数据集上都有优秀的表现。

相对来说，SVM尽量保持与样本间距离的性质导致它抗攻击的能力更强。

和随机森林一样，这也是一个拿到数据就可以先尝试一下的算法。

**SVM算法优点：**

　　可用于线性/非线性分类，也可以用于回归；

　　低泛化误差；

　　容易解释；

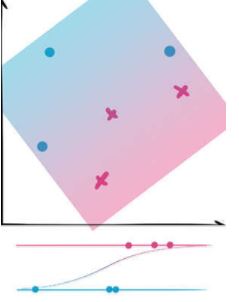
　　计算复杂度较低；

**缺点：**

　　对参数和核函数的选择比较敏感；

原始的SVM只比较擅长处理二分类问题；

## **逻辑斯蒂回归 (Logistic regression)**



逻辑斯蒂回归这个名字太诡异了，我就叫它LR吧，反正讨论的是分类器，也没有别的方法叫LR。顾名思义，它其实是回归类方法的一个变体。

回归方法的核心就是为函数找到最合适的参数，使得函数的值和样本的值最接近。例如线性回归(Linear regression)就是对于函数f(x)=ax+b，找到最合适的a,b。

LR拟合的就不是线性函数了，它拟合的是一个概率学中的函数，f(x)的值这时候就反映了样本属于这个类的概率。

 logsitc回归方法主要是用最大似然估计来学习的，所以单个样本的后验概率为：

IMG_256

**适用情景：**

LR同样是很多分类算法的基础组件，它的好处是输出值自然地落在0到1之间，并且有概率意义。

因为它本质上是一个线性的分类器，所以处理不好特征之间相关的情况。

虽然效果一般，却胜在模型清晰，背后的概率学经得住推敲。它拟合出来的参数就代表了每一个特征(feature)对结果的影响。也是一个理解数据的好工具。

**Logistic回归优点：**

　　1、实现简单；

　　2、分类时计算量非常小，速度很快，存储资源低；

**缺点：**

　　1、容易欠拟合，一般准确度不太高

　　2、只能处理两分类问题（在此基础上衍生出来的softmax可以用于多分类），且必须线性可分；

**线性回归：**

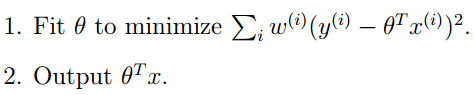
　　线性回归才是真正用于回归的，而不像logistic回归是用于分类，其基本思想是用梯度下降法对最小二乘法形式的误差函数进行优化，当然也可以用normal equation直接求得参数的解，结果为：

IMG_256

　　而在LWLR（局部加权线性回归）中，参数的计算表达式为:

IMG_257

　　因为此时优化的是：



　　由此可见LWLR与LR不同，LWLR是一个非参数模型，因为每次进行回归计算都要遍历训练样本至少一次。

**线性回归优点：**

　　实现简单，计算简单；

　　缺点：

　　不能拟合非线性数据；

**KNN算法：**

　　KNN即最近邻算法，其主要过程为：

　　1. 计算训练样本和测试样本中每个样本点的距离（常见的距离度量有欧式距离，马氏距离等）；

　　2. 对上面所有的距离值进行排序；

　　3. 选前k个最小距离的样本；

　　4. 根据这k个样本的标签进行投票，得到最后的分类类别；

　　如何选择一个最佳的K值，这取决于数据。一般情况下，在分类时较大的K值能够减小噪声的影响。但会使类别之间的界限变得模糊。一个较好的K值可通过各种启发式技术来获取，比如，交叉验证。另外噪声和非相关性特征向量的存在会使K近邻算法的准确性减小。

　　近邻算法具有较强的一致性结果。随着数据趋于无限，算法保证错误率不会超过贝叶斯算法错误率的两倍。对于一些好的K值，K近邻保证错误率不会超过贝叶斯理论误差率。

　　注：马氏距离一定要先给出样本集的统计性质，比如均值向量，协方差矩阵等。关于马氏距离的介绍如下：

**KNN算法的优点：**

　　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. 可用于非线性分类；

　　3. 训练时间复杂度为O(n)；

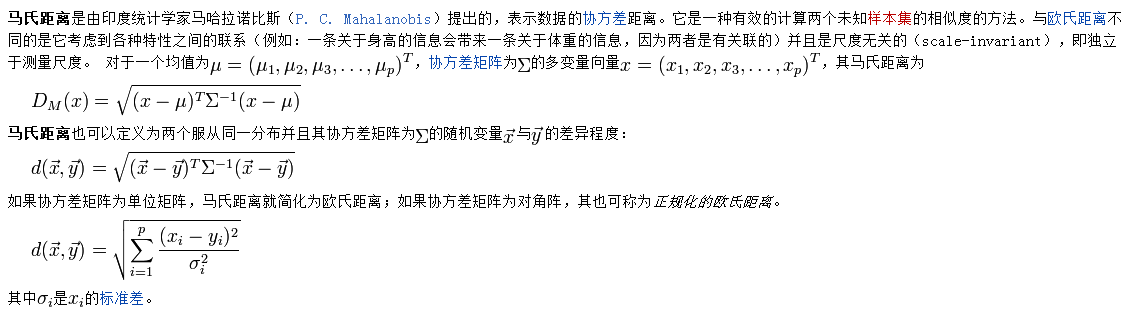
　　4. 准确度高，对数据没有假设，对outlier不敏感；

**缺点：**

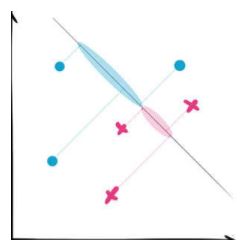
　　1. 计算量大；

　　2. 样本不平衡问题（即有些类别的样本数量很多，而其它样本的数量很少）；

　　3. 需要大量的内存；



## **判别分析 (Discriminant analysis)**



判别分析主要是统计那边在用.

判别分析的典型例子是线性判别分析(Linear discriminant analysis)，简称LDA。

（这里注意不要和隐含狄利克雷分布(Latent Dirichlet allocation)弄混，虽然都叫LDA但说的不是一件事。）

LDA的核心思想是把高维的样本投射(project)到低维上，如果要分成两类，就投射到一维。要分三类就投射到二维平面上。这样的投射当然有很多种不同的方式，LDA投射的标准就是让同类的样本尽量靠近，而不同类的尽量分开。对于未来要预测的样本，用同样的方式投射之后就可以轻易地分辨类别了。

使用情景：

判别分析适用于高维数据需要降维的情况，自带降维功能使得我们能方便地观察样本分布。它的正确性有数学公式可以证明，所以同样是很经得住推敲的方式。

但是它的分类准确率往往不是很高，所以不是统计系的人就把它作为降维工具用吧。

同时注意它是假定样本成正态分布的，所以那种同心圆形的数据就不要尝试了。

## 神经网络 (Neural network)

神经网络现在是火得不行啊。它的核心思路是利用训练样本(training sample)来逐渐地完善参数。还是举个例子预测身高的例子，如果输入的特征中有一个是性别（1:男；0:女），而输出的特征是身高（1:高；0:矮）。那么当训练样本是一个个子高的男生的时候，在神经网络中，从“男”到“高”的路线就会被强化。同理，如果来了一个个子高的女生，那从“女”到“高”的路线就会被强化。

最终神经网络的哪些路线比较强，就由我们的样本所决定。

神经网络的优势在于，它可以有很多很多层。如果输入输出是直接连接的，那它和LR就没有什么区别。但是通过大量中间层的引入，它就能够捕捉很多输入特征之间的关系。卷积神经网络有很经典的不同层的可视化展示(visulization)，我这里就不赘述了。

神经网络的提出其实很早了，但是它的准确率依赖于庞大的训练集，原本受限于计算机的速度，分类效果一直不如随机森林和SVM这种经典算法。

使用情景：

数据量庞大，参数之间存在内在联系的时候。

当然现在神经网络不只是一个分类器，它还可以用来生成数据，用来做降维，这些就不在这里讨论了。

## 提升算法（Boosting）

接下来讲的一系列模型，都属于集成学习算法(Ensemble Learning)，基于一个核心理念：三个臭皮匠，顶个诸葛亮。

翻译过来就是：当我们把多个较弱的分类器结合起来的时候，它的结果会比一个强的分类器更

典型的是AdaBoost，它的实现是一个渐进的过程，从一个最基础的分类器开始，每次寻找一个最能解决当前错误样本的分类器。用加权取和(weighted sum)的方式把这个新分类器结合进已有的分类器中。

它的好处是自带了特征选择（feature selection），只使用在训练集中发现有效的特征(feature)。这样就降低了分类时需要计算的特征数量，也在一定程度上解决了高维数据难以理解的问题。

最经典的AdaBoost实现中，它的每一个弱分类器其实就是一个决策树。这就是之前为什么说决策树是各种算法的基石。

**使用情景：**

好的Boosting算法，它的准确性不逊于随机森林。虽然在[1]的实验中只有一个挤进前十，但是实际使用中它还是很强的。因为自带特征选择（feature selection）所以对新手很友好，是一个“不知道用什么就试一下它吧”的算法。

**Boosting算法的优点：**

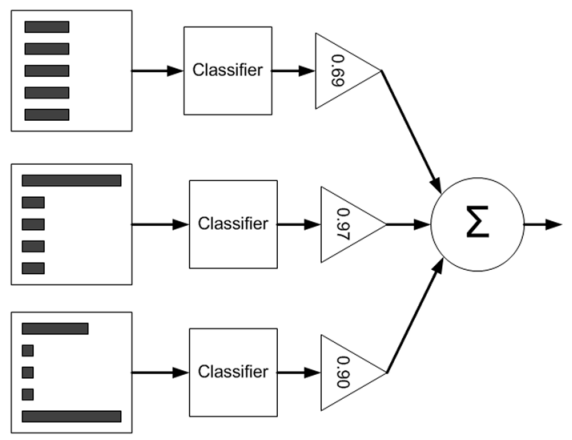
低泛化误差；

容易实现，分类准确率较高，没有太多参数可以调；

**缺点：**

对outlier比较敏感；

从图中可以看到，在训练过程中我们需要训练出多个弱分类器（图中为3个），每个弱分类器是由不同权重的样本（图中为5个训练样本）训练得到（其中第一个弱分类器对应输入样本的权值是一样的），而每个弱分类器对最终分类结果的作用也不同，是通过加权平均输出的，权值见上图中三角形里面的数值。那么这些弱分类器和其对应的权值是怎样训练出来的呢？

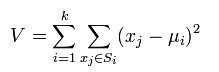


根据聚类思想划分：

　　1. 基于划分的聚类:

　　K-means, k-medoids(每一个类别中找一个样本点来代表),CLARANS.

　　k-means是使下面的表达式值最小：



**k-means算法的优点：**

　　（1）k-means算法是解决聚类问题的一种经典算法，算法简单、快速。

　　（2）对处理大数据集，该算法是相对可伸缩的和高效率的，因为它的复杂度大约是O(nkt)，其中n是所有对象的数目，k是簇的数目,t是迭代的次数。通常k<<n。这个算法通常局部收敛。

　　（3）算法尝试找出使平方误差函数值最小的k个划分。当簇是密集的、球状或团状的，且簇与簇之间区别明显时，聚类效果较好。

**缺点：**

　　（1）k-平均方法只有在簇的平均值被定义的情况下才能使用，且对有些分类属性的数据不适合。

　　（2）要求用户必须事先给出要生成的簇的数目k。

　　（3）对初值敏感，对于不同的初始值，可能会导致不同的聚类结果。

　　（4）不适合于发现非凸面形状的簇，或者大小差别很大的簇。

　　（5）对于"噪声"和孤立点数据敏感，少量的该类数据能够对平均值产生极大影响。

　2. 基于层次的聚类：

　　自底向上的凝聚方法，比如AGNES。

　　自上向下的分裂方法，比如DIANA。

3. 基于密度的聚类：

　　DBSACN,OPTICS,BIRCH(CF-Tree),CURE.

4. 基于网格的方法：

　　STING, WaveCluster.

5. 基于模型的聚类：

　　EM,SOM,COBWEB.

以上这些算法的简介可参考[聚类（百度百科）。](http://baike.baidu.com/view/31801.htm)

**推荐系统：**

推荐系统的实现主要分为两个方面：基于内容的实现和协同滤波的实现。

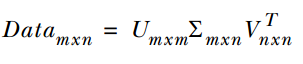
基于内容的实现：

不同人对不同电影的评分这个例子，可以看做是一个普通的回归问题，因此每部电影都需要提前提取出一个特征向量(即x值)，然后针对每个用户建模，即每个用户打的分值作为y值，利用这些已有的分值y和电影特征值x就可以训练回归模型了(最常见的就是线性回归)。这样就可以预测那些用户没有评分的电影的分数。（值得注意的是需对每个用户都建立他自己的回归模型）

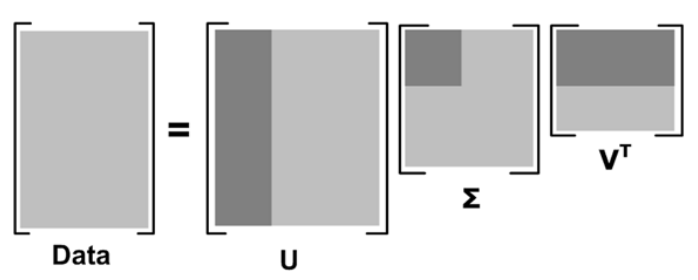
基于协同滤波的实现：

协同滤波（CF）可以看做是一个分类问题，也可以看做是矩阵分解问题。协同滤波主要是基于每个人自己的喜好都类似这一特征，它不依赖于个人的基本信息。比如刚刚那个电影评分的例子中，预测那些没有被评分的电影的分数只依赖于已经打分的那些分数，并不需要去学习那些电影的特征。

SVD将矩阵分解为三个矩阵的乘积，公式如下所示：



中间的矩阵sigma为对角矩阵，对角元素的值为Data矩阵的奇异值(注意奇异值和特征值是不同的)，且已经从大到小排列好了。即使去掉特征值小的那些特征，依然可以很好的重构出原始矩阵。如下图所示：



　　其中更深的颜色代表去掉小特征值重构时的三个矩阵。

　　果m代表商品的个数，n代表用户的个数，则U矩阵的每一行代表商品的属性，现在通过降维U矩阵（取深色部分）后，每一个商品的属性可以用更低的维度表示（假设为k维）。这样当新来一个用户的商品推荐向量X，则可以根据公式X'\*U1\*inv(S1)得到一个k维的向量，然后在V’中寻找最相似的那一个用户（相似度测量可用余弦公式等），根据这个用户的评分来推荐（主要是推荐新用户未打分的那些商品）。具体例子可以参考网页：[SVD在推荐系统中的应用](http://blog.csdn.net/wuyanyi/article/details/7964883)。

**Regularization:**

　　作用是（网易电话面试时有问到）：

　　1. 数值上更容易求解；

　　2. 特征数目太大时更稳定；

　　3. 控制模型的复杂度，光滑性。复杂性越小且越光滑的目标函数泛化能力越强。而加入规则项能使目标函数复杂度减小，且更光滑。

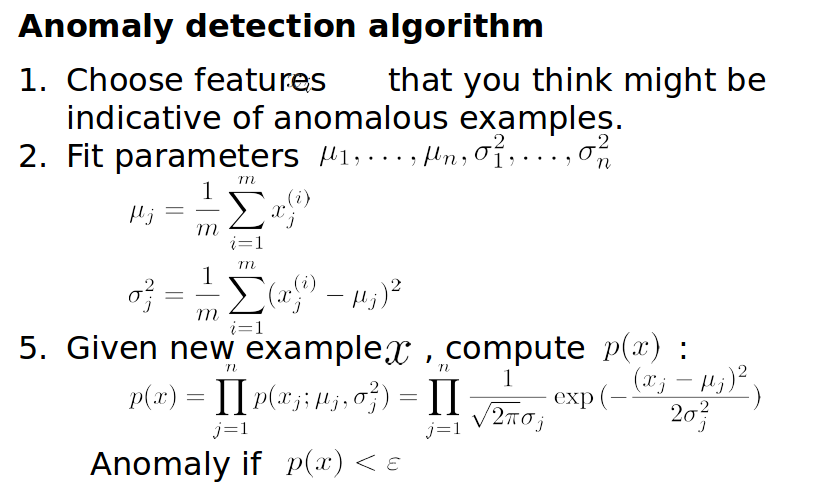
　　4. 减小参数空间；参数空间越小，复杂度越低。

　　5. 系数越小，模型越简单，而模型越简单则泛化能力越强（Ng宏观上给出的解释）。

　　6. 可以看成是权值的高斯先验。

**异常检测：**

　　可以估计样本的密度函数，对于新样本直接计算其密度，如果密度值小于某一阈值，则表示该样本异常。而密度函数一般采用多维的高斯分布。如果样本有n维，则每一维的特征都可以看作是符合高斯分布的，即使这些特征可视化出来不太符合高斯分布，也可以对该特征进行数学转换让其看起来像高斯分布，比如说x=log(x+c), x=x^(1/c)等。异常检测的算法流程如下：



 　　其中的ε也是通过交叉验证得到的，也就是说在进行异常检测时，前面的p(x)的学习是用的无监督，后面的参数ε学习是用的有监督。那么为什么不全部使用普通有监督的方法来学习呢（即把它看做是一个普通的二分类问题）？主要是因为在异常检测中，异常的样本数量非常少而正常样本数量非常多，因此不足以学习到好的异常行为模型的参数，因为后面新来的异常样本可能完全是与训练样本中的模式不同。

　　另外，上面是将特征的每一维看成是相互独立的高斯分布，其实这样的近似并不是最好的，但是它的计算量较小，因此也常被使用。更好的方法应该是将特征拟合成多维高斯分布，这时有特征之间的相关性，但随之计算量会变复杂，且样本的协方差矩阵还可能出现不可逆的情况（主要在样本数比特征数小，或者样本特征维数之间有线性关系时）。

## 装袋算法（Bagging）

同样是弱分类器组合的思路，相对于Boosting，其实Bagging更好理解。它首先随机地抽取训练集（training set），以之为基础训练多个弱分类器。然后通过取平均，或者投票(voting)的方式决定最终的分类结果。

因为它随机选取训练集的特点，Bagging可以一定程度上避免过渡拟合(overfit)。

在[1]中，最强的Bagging算法是基于SVM的。如果用定义不那么严格的话，随机森林也算是Bagging的一种。

使用情景：

相较于经典的必使算法，Bagging使用的人更少一些。一部分的原因是Bagging的效果和参数的选择关系比较大，用默认参数往往没有很好的效果。

虽然调对参数结果会比决策树和LR好，但是模型也变得复杂了，没事有特别的原因就别用它了。

## Stacking

它所做的是在多个分类器的结果上，再套一个新的分类器。

这个新的分类器就基于弱分类器的分析结果，加上训练标签(training label)进行训练。一般这最后一层用的是LR。

Stacking在[1]里面的表现不好，可能是因为增加的一层分类器引入了更多的参数，也可能是因为有过渡拟合(overfit)的现象。

使用情景：

没事就别用了。

## 多专家模型（Mixture of Experts）

最近这个模型还挺流行的，主要是用来合并神经网络的分类结果。我也不是很熟，对神经网络感兴趣，而且训练集异质性（heterogeneity）比较强的话可以研究一下这个。

**讲到这里分类器其实基本说完了。讲一下问题里面其他一些名词吧。**

## 

## 最大熵模型 (Maximum entropy model)

最大熵模型本身不是分类器，它一般是用来判断模型预测结果的好坏的。

对于它来说，分类器预测是相当于是：针对样本，给每个类一个出现概率。比如说样本的特征是：性别男。我的分类器可能就给出了下面这样一个概率：高（60%），矮（40%）。

而如果这个样本真的是高的，那我们就得了一个分数60%。最大熵模型的目标就是让这些分数的乘积尽量大。

最大熵模型是一个概率模型，决策树的数学基础就是它。

LR其实就是使用最大熵模型作为优化目标的一个算法。

## EM(Expectation Maximization)

在统计计算中，最大期望（EM）算法是在概率（probabilistic）模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐藏变量（Latent Variable）。

最大期望经常用在机器学习和计算机视觉的数据聚类（Data Clustering）领域。

比如说食堂的大师傅炒了一份菜，要等分成两份给两个人吃，显然没有必要拿来天平一点一点的精确的去称分量，最简单的办法是先随意的把菜分到两个碗中，然后观察是否一样多，把比较多的那一份取出一点放到另一个碗中，这个过程一直迭代地执行下去，直到大家看不出两个碗所容纳的菜有什么分量上的不同为止。（来自百度百科）

EM算法就是这样，假设我们估计知道A和B两个参数，在开始状态下二者都是未知的，并且知道了A的信息就可以得到B的信息，反过来知道了B也就得到了A。可以考虑首先赋予A某种初值，以此得到B的估计值，然后从B的当前值出发，重新估计A的取值，这个过程一直持续到收敛为止

有时候因为样本的产生和隐含变量有关（隐含变量是不能观察的），而求模型的参数时一般采用最大似然估计，由于含有了隐含变量，所以对似然函数参数求导是求不出来的，这时可以采用EM算法来求模型的参数的（对应模型参数个数可能有多个），EM算法一般分为2步：

　　E步：选取一组参数，求出在该参数下隐含变量的条件概率值；

　　M步：结合E步求出的隐含变量条件概率，求出似然函数下界函数（本质上是某个期望函数）的最大值。

　　重复上面2步直至收敛。

EM算法用于寻找隐藏参数的最大似然估计。该算法首先在E step中计算隐藏参数的似然估计，然后再M step中进行最大化，然后进行EM step的迭代直至收敛。应用场景之一是聚类问题，但EM算法本身并不是一个聚类算法。举个例子，GMM(高斯混合模型)和Kmeans在聚类时都使用了EM算法。

## 隐马尔科夫 (Hidden Markov model)

这是一个基于序列的预测方法，核心思想就是通过上一个（或几个）状态预测下一个状态。

之所以叫“隐”马尔科夫是因为它的设定是状态本身我们是看不到的，我们只能根据状态生成的结果序列来学习可能的状态。

HMM也是一个统计模型，我们不能用它来做什么，但是可以利用这个模型对现实生活中的问题建模

适用场景：

可以用于序列的预测，可以用来生成序列。

## **条件随机场 (Conditional random field)**

典型的例子是linear-chain CRF。

**K近邻**：算法采用测量不同特征值之间的距离的方法进行分类。

*优点：*

1.简单好用，容易理解，精度高，理论成熟，既可以用来做分类也可以用来做回归；

2.可用于数值型数据和离散型数据；

3.训练时间复杂度为O(n)；无数据输入假定；

4.对异常值不敏感

*缺点：*

1.计算复杂性高；空间复杂性高；

2.样本不平衡问题（即有些类别的样本数量很多，而其它样本的数量很少）；

3.一般数值很大的时候不用这个，计算量太大。但是单个样本又不能太少 否则容易发生误分。

4.最大的缺点是无法给出数据的内在含义。

**朴素贝叶斯**

*优点：*

1.生成式模型，通过计算概率来进行分类，可以用来处理多分类问题，

2.对小规模的数据表现很好，适合多分类任务，适合增量式训练，算法也比较简单。

*缺点：*

1.对输入数据的表达形式很敏感，

2.由于朴素贝叶斯的“朴素”特点，所以会带来一些准确率上的损失。

3.需要计算先验概率，分类决策存在错误率。

**决策树**

*优点：*

1.概念简单，计算复杂度不高，可解释性强，输出结果易于理解；

2.数据的准备工作简单， 能够同时处理数据型和常规型属性，其他的技术往往要求数据属性的单一。

3.对中间值得确实不敏感，比较适合处理有缺失属性值的样本，能够处理不相关的特征；

4.应用范围广，可以对很多属性的数据集构造决策树，可扩展性强。决策树可以用于不熟悉的数据集合，并从中提取出一些列规则 这一点强于KNN。

*缺点：*

1.容易出现过拟合；

2.对于那些各类别样本数量不一致的数据，在决策树当中,信息增益的结果偏向于那些具有更多数值的特征。

3. 信息缺失时处理起来比较困难。 忽略数据集中属性之间的相关性。

**Svm**

*优点：*

1.可用于线性/非线性分类，也可以用于回归，泛化错误率低，计算开销不大，结果容易解释；

2.可以解决小样本情况下的机器学习问题，可以解决高维问题 可以避免神经网络结构选择和局部极小点问题。

3.SVM是最好的现成的分类器，现成是指不加修改可直接使用。并且能够得到较低的错误率，SVM可以对训练集之外的数据点做很好的分类决策。

*缺点：*对参数调节和和函数的选择敏感，原始分类器不加修改仅适用于处理二分类问题。

**Logistic回归**：根据现有数据对分类边界线建立回归公式，依次进行分类。

*优点：*实现简单，易于理解和实现；计算代价不高，速度很快，存储资源低；

*缺点：*容易欠拟合，分类精度可能不高

**EM 期望最大化算法-上帝算法**

只要有一些训练数据，再定义一个最大化函数，采用EM算法，利用计算机经过若干次迭代，就可以得到所需的模型。EM算法是自收敛的分类算法，既不需要事先设定类别也不需要数据见的两两比较合并等操作。缺点是当所要优化的函数不是凸函数时，EM算法容易给出局部最佳解，而不是最优解。

