**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）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MultiTaskElasticNetCV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：**

　　　　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XθXθ的正态分布，正则化参数αα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θθ的先验分布规律为球形正态分布，超参数为λλ。我们需要通过最大化边际似然函数来估计超参数αα和λλ，以及回归系数θθ。

　　　　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。**

