广义线性模型

广义线性模型

翻译自scikit-learn 官方文档“广义线性模型”一章。

以下这些方法是针对那些目标函数的值是输入向量的线性组合的回归问题，用数学方法表示如下：

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p$$

1.1.1 最小二乘法

线性回归用参数W $w = (w_1, ..., w_p)$ 来最小化训练集的残差的平方和，从而训练出一个线性模型从数学上讲它解决以下问题：

$$\min_w ||Xw - y||_2^2$$

LinearRegression模型有fit(x, y)函数，并保存训练出的参数W 在coef_变量中：

```
>>> from sklearn import linear_model
>>> clf = linear_model.LinearRegression()
>>> clf.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> clf.coef_array([ 0.5,  0.5])
```

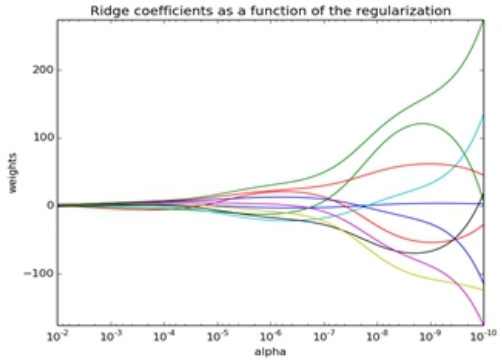
但是，普通最小二乘法的回归系数的估计需要模型之间各个属性相互独立。当属性之间是相关的，输入矩阵X之间的列是线性相关的，这样X就会近似于奇异的从而造成最小二乘估计对随机错误十分敏感，使得误差很大，这种多重共线性会在数据没经过专业处理的时候出现。

1.1.2 岭回归

岭回归通过对参数施加一些惩罚项来解决普通线性回归中的一些问题，岭回归最小化残差平方和及其惩罚项：

$$\min_w ||Xw - y||_2^2 + \alpha ||w||_2^2$$

α>0， 是一个控制参数伸缩系数的参数，α越大，伸缩系数也越大，模型也更倾向于非线性，如下图所示。



和其他线性模型一样，Ridge模型也有fit函数，也将训练的参数保存在cof_变量中。

```
>>> from sklearn import linear_model
>>> clf = linear_model.Ridge (alpha = .5)
>>> clf.fit ([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
Ridge(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=None,      normalize=False, solver='auto', tol=0.001)
>>> clf.coef_array([ 0.34545455,  0.34545455])
```

```
>>> clf.intercept_ 0.13636...
```

岭回归和普通线性回归也同样的时间复杂度。

广义交叉验证：

RidgeCV在岭回归的基础上加上了对α参数的交叉验证，它使用留一交叉验证。

```
>>> from sklearn import linear_model
>>> clf = linear_model.RidgeCV(alphas=[0.1, 1.0, 10.0])
>>> clf.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
    RidgeCV(alphas=[0.1, 1.0, 10.0], cv=None, fit_intercept=True, scoring=None,    normalize=False)
>>> clf.alpha_
0.1
```

1.1.3 Lasso回归

Lasso回归是一个能得到稀疏系数的线性模型。由于它倾向于取得更少参数值的解，有效减少了参数的个数，因此在很多情况下是一种有效的方法。因为这个原因，Lasso及其变体是压缩传感领域的基础，在特定条件下，它可以还原非0的权重。

从数学上讲，Lasso回归是一个包含L1正则的线性模型，目标优化函数如下：

$$\min_{w} \frac{1}{2n_{samples}} | Xw - y| \frac{2}{2} + \alpha||w||_1$$

Lasso回归通过在最小二乘的基础上施加惩罚项$\alpha||w| 1$，α是个常数，$||w||$1是参数向量的L1正则的形式。

Lasso模型使用坐标下降的方法来训练参数：

```
>>> clf = linear_model.Lasso(alpha = 0.1)
>>> clf.fit([[0, 0], [1, 1]], [0, 1])
    Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,    normalize=False, positive=False, precompute=False, random_state=None,
    selection='cyclic', tol=0.0001, warm_start=False)
>>> clf.predict([[1, 1]])array([ 0.8])
```

设定正则化参数：α 控制参数的稀疏度

1. 使用交叉验证

scikit-learn的 LassoCV和LassoLarsCV使用交叉验证来设定参数。

对于高维有很多共线性的回归量的数据集，LassoCV通常很受欢迎，但是，LassoLarsCV能发掘更多α变量的相关值，并且当样本的数量相对观察值的数量更多的时候，

它通常比LassoCV更快。

2. 基于信息标准的模型选择

LassoLarsIC则使用Akaike信息标准和Bayes信息标准

1.1.4 Elastic Net

ElasticNet是一个使用L1和L2正则的线性模型。这种结合在确保了L1正则学习得到稀疏模型的特性下，同时得到L2正则的特性。我们使用l1_ratio 来控制L1和L2
的结合。

当很多特征是相互联系相互影响的时候，ElasticNet是一种很有效的方法。Lasso倾向于随机挑选一个，而ElasticNet则同时选择。

Lasso 和Ridge之间一个优点是它允许ElasticNet在旋转的情况下保留Ridge的一些稳定性。

目标函数为最小化以下函数：

ElasticNetCV 可以通过交叉验证来设定参数alpha 和 l1_ratio

1.1.5. Multi-task Lasso

1.1.6. Least Angle Regression

最小角回归是一种对于高维数据的回归算法，它的优点有：

1）当数据的维度远大于样本数的时候很有效；

2）它和前向选择计算一样快，并且和最小二乘法有同样的时间复杂度；

1.1.10. Logistic regression

逻辑回归是一种线性分类模型，也被叫做最大熵分类器，log线性分类器。它把结果输入到logistic 函数来得到最后的概率描述。

LogisticRegression这一类中实现了逻辑回归，并补充了多分类和L1，L2正则。

带L2正则的逻辑回归极小化以下损失函数：

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

带L1正则的逻辑回归极小化以下损失函数：

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

它的优化方法有 “liblinear”, “newton-cg” 和 “lbfgs”。后两者只支持L2正则，并且被证明在高维数据上收敛比较快，L1正则得到的是稀疏的解。“liblinear”优化方法

使用基于"Liblinear" 的坐标下降方法，它不能学习到一个多分类的模型。对于多分类模型可以通过分解成“one-vs-rest” 模式来得到。

设置multi_class 成“multinomial” 并使用“newton-cg” 或 “lbfgs” 优化算法可以学习到一个多项的逻辑回归模型，它也比通过分解成“one-vs-rest” 模式得到的模型更

有效。 "L-BFGS" 和 "newton-cg" 不能优化得到L1模型，因此“multinomial” 设置不能得到稀疏的模型。

LogisticRegressionCV 类实现了逻辑回归的交叉验证，一般来说“newton-cg” 和 “lbfgs”由于热启动的原因收敛速度更快。对于多分类问题，"multi_class" 如果被设

置成 “ovr”，得到的参数是每个类别的优化参数，如果被设置成 “multinomial”， 得到的参数是最小化交叉熵损失函数得到的。

1.1.11. Stochastic Gradient Descent - SGD

随机梯度下降是拟合线性模型的非常简单但有效的方法，尤其是当样本数和特征维数非常大的时候。SGDClassifier和 SGDRegressor类分别实现了使用线性模型的

分类和回归，并提供多种损失函数和正则化项。loss="log"，SGDClassifier实现的是一个逻辑回归；loss="hinge"， 它实现的是SVM。

1.1.12. Perceptron

感知机也是一种简单的适合大规模学习的算法。默认情况下，它不需要学习率，不能正则化，只在错误数据上更新模型。最后这点性质使得感知机比随机梯度下降

相对较快，模型也相对更稀疏。

1.1.13. Passive Aggressive Algorithms