线性回归总结

> Author:Zreal
>
> 参考：
>
> - https://blog.yaodataking.com/2017/02/28/machine-learning-2/
> - https://zhuanlan.zhihu.com/p/30535220
> - https://zhuanlan.zhihu.com/p/28408516

线性回归总结

理论总结：

Linear Regression

Ridge Regression

Lasso Regression

Elastic Net Regression

Logistc Regression

应用实例

california_housing regression

20newsgroups classify

理论总结：

Linear Regression

线性回归用最适直线(回归线)去建立因变量Y和一个或多个自变量X之间的关系。可以用公式来表示：

$$Y = w^T * X + b$$

如何确定w和b？显然，关键在于如何衡量f(x)于y之间的差别。均方误差是回归任务中最常用好的性能度量，因此我们可以试图让均方误差最小化，（之后很多类型的回归都是在损失函数上进行修改）

$$argmin \sum_{i=1}^{m}(f(x_i) - y_i)^2$$

$$argmin \sum_{i=1}^{m}(f(x_i) - w * x_i - b)^2$$

之后根据最小二乘法得到最优解的闭式解，或者利用梯度下降法无限逼近最优闭式解。

（训练过程跳过）

Ridge Regression

岭回归是一种专用于共线性数据分析的有偏估计回归方法，实质上是一种改良的最小二乘估计发，通过放弃最小二乘法的无偏性，以损失部分信息，降低精度为代价获得得回归系数更加符合实际，更可靠的回归方法，**对病态数据的拟合要强于最小二乘法**

损失函数公式如下：

$$argmin(\sum_{i=1}^{m}(f(x_i)-y_i)^2+alpha*\sum_{i=1}^{m}w_i^2\ )$$

损失函数中添加的一个惩罚项 $alpha*\sum_{i=1}^{m}w_i^2$

称为L2正则化。加入此惩罚项进行优化后，限制了回归系数 w_i的绝对值，数学上可以证明等价形式如下：

$$Loss(w)=\sum_{i=1}^{m}(f(x_i)-y_i)^2$$

$$s.t.\sum_{i=1}^{m}w_i^2<=t$$

其中t为某个阈值

岭回归的特点

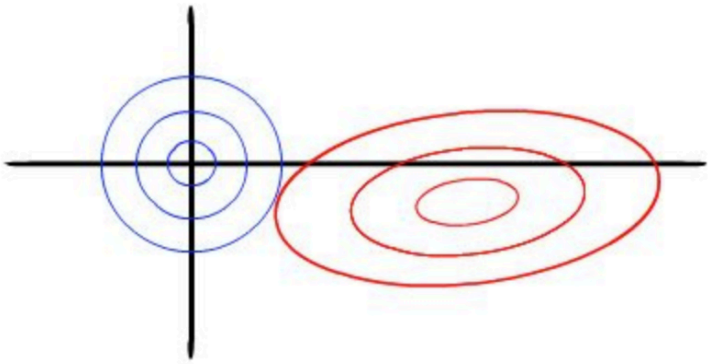
当岭参数 alpha=0时，得到的解时最小二乘解

当岭参数alpha 趋向更大时，岭回归系数w趋向于0，约束项t很小

岭回归限定了所有回归系数的平方和不大于t，在使用普通最小二乘法回归的时候两个变量具有相关性，可能会使得一个系数时很大的正数，另一个系数是一个很大的负数。通过岭回归 的约束条件的现实，可以避免这个问题

岭回归的几何意义

以两个变量为例，残差平方和可以表示 w1,w2的一个二次函数，是一个在三维空间中的抛物面,用等值线来表示。而限制条件$w_1^2+w_2^2<t$，相当于二维平面的一个圆。这个时候等值线与圆相切的点便是在约束条件下的最优点，如下同所示



Lasso Regression

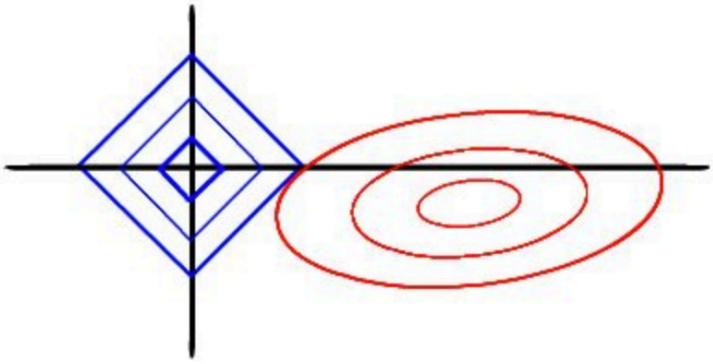
LASSO(The Least Absolute Shrinkage and Selection Operator)是另一种缩减方法，将回归系数收缩在一定的区域内。LASSO的主要思想是构造一个一阶惩罚函数获得一个精炼的模型, **通过最终确定一些变量的系数为0进行特征筛选。**

LASSO的惩罚项为:

$$s.t. \sum_{i=1}^{m} |w_i| <= t$$

与岭回归的不同在于，此约束条件使用了绝对值的一阶惩罚函数代替了平方和的二阶函数。虽然只是形式稍有不同，但是得到的结果却又很大差别。在LASSO中，当alpha很小的时候，一些系数会随着变为0而岭回归却很难使得某个系数**恰好**缩减为0. 我们可以通过几何解释看到LASSO与岭回归之间的不同。

LASSO回归的几何解释特点



筛选变量

因为约束是一个正方形，所以除非相切，正方形与同心椭圆的接触点往往在正方形顶点上。而顶点又落在坐标轴上，这就意味着符合约束的自变量系数有一个值是 0。

相比圆，方形的顶点更容易与抛物面相交，顶点就意味着对应的很多系数为0，而岭回归中的圆上的任意一点都很容易与抛物面相交很难得到正好等于0的系数。这也就意味着，**lasso起到了很好的筛选变量的作用。**

复杂度调整

正放型的大小决定复杂度调整的程度，及惩罚系数t决定复杂度调整的程度，假设t趋近于0，那么这个正方形的大小趋近于一个常数,而所有自变量w的大小趋近于0，这是模型极简情况下的极端情况。及t越小，对参数较多的模型的惩罚程度越大，越容易得到一个简单模型。

Elastic Net Regression

其实Lasso回归和Ridge回归都是属于弹性网回归家族，首先先看Elastic Net 回归的损失函数：

$$argmin(\sum_{i=1}^{m}(f(x_i)-y_i)^2+0.5*alpha*（1-ratio）*\sum_{i=1}^{m}w_i^2+alpha*ratio*|w|)$$

可以看到弹性网回归是将Lasso回归和Ridge回归综合起来，在前边乘了系数，使其对于不同的实际情况有了更广泛的应用性

Logistc Regression

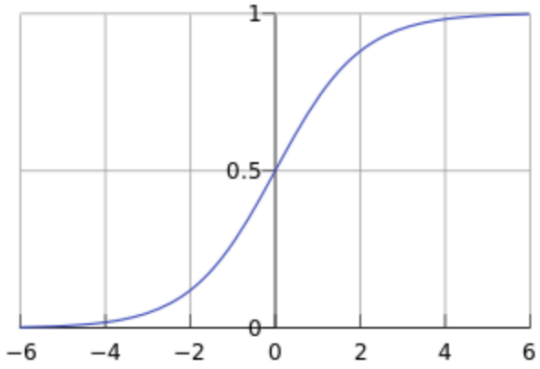
逻辑回归在是一种解决2分类问题的机器学习方法（0/1问题，但不限于此问题softmax回归是在logistc回归的基础上解决多分类问题），在实际案例中运用较广。逻辑回归是一种广义的线性模型，去除sigmoid函数映射，就是一个线性回归模型。逻辑回归通过Sigmoid函数引入了非线性因素，因此可以轻松处理0/1分类问题。

$$Y=sigmoid(w^T*X+b)$$

sigmoid 函数

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

如图：



取值在0，1之间，这个特性对于解决2分类问题极为重要。这个函数在给定的w，b的条件下被认为是X的取1的概率，即：

$$P(Y=1|w,b)=\frac{1}{1+e^{-(w^T*X+b)}}$$

如果该值大于0.5，及划分为1，否则为0

（注：选择0.5为阈值是一个一般的做法，实际应用时特定的情况可以选择不同阈值，如果对正例的判别准确性要求高，可以选择阈值大一些，对正例的召回要求高，则可以选择阈值小一些。）

> 在softmax中，如果划分n类，所得结果为一个1维向量，其中概率最大的为该数据项所划分的类别。

应用实例

> 因为实验过程简单，下面重点介绍API的相关参数作用，注意事项，以及结果分析
>
> 实验源码：https://github.com/Zrealshadow/MachineLearning/tree/master/LAB1%20LinearRegression
>
> 实验步骤解释在注释中有明显说明

california_housing regression

> 分别使用Linear Regression， Ridge Regression， Lasso Regression，ElasticNet Regression对房价进行预测

读取数据：

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
data=fetch_california_housing()
data_x=data.data
data_y=data.target
feature=data.feature_names
x_train,x_test,y_train,y_test=train_test_split(\
                        data_x,data_y,test_size=0.2,random_state=1)
```

其中fetch_california_housing()返回一个class，成员主要成员data，target，feature_names，分别代表，自变量，因变量和特征描述

train_test_split 是一个常用的划分训练测试集的常用函数，其中主要参数如下：

> Arrays: x,y 输入的数据集
>
> test_size: float 0.0-1.0之间，表示测试集占总训练样本的比例 default:0.25

> train_size: float 0.0-1.0之间，表示训练集占总训练样本的比例
>
> random_state: int 随机种子， default：None
>
> shuffle: bool 分裂之前是否打乱数据集

Linear Regression

该类在：

```
from sklearn.linear_model import LinearRegression
```

定义时主要参数：

> fit_intercept: bool 是否对加载入模型的数据进行处理
>
> normalize：bool 如果fit_intercept 为False该参数忽略，否则若normalize 为ture， 对输入数据进行L2正则化
>
> copy_X：bool，运算过程中是否重写x

方法：

> fit(), fit(X,y) 对模型进行训练
>
> get_param() 返回模型参数
>
> predict() predict(test_x)利用模型进行预测，返回Y
>
> score(X,y) 利用模型均行预测，并对模型正确率进行打分，打分公式为R2_score

在后面API中不会重复上述相同功能的参数或方法，只对特别API指出

Ridge Regression / Lasso Regression

```
from sklearn.linear_model import Ridge,Lasso
```

参数：

> 同Linear Regression
>
> max_iter: 最大迭代次数
>
> alpha： 惩罚系数，为正则项前的系数

ElasticNet regression

```
from sklearn.linear_model import ElasticNet
```

参数：

> 同Linear Regression
>
> max_iter: 最大迭代次数
>
> alpha： 惩罚系数，为正则项前的系数
>
> L1_ratio:float 调整L1正则项和L2正则项之间比例

网格参数调优 GridSearch

```
from sklearn.model_selection import GridSearchCV
```

主要参数：

> estimator:评估标准，及定义一个损失函数，模型
>
> param_grid： dict 候选参数
>
> cv：int 几则交叉验证

返回对象成员：

> **cv_results_** :data frame / np.array/list 返回所有参数组合后的结果
>
> best_score_:float 交叉运算中的最佳得分
>
> best_param_:dict 最好的参数组合

20newsgroups classify

> 利用logistic对数据进行分类

获得数据集：

```
from sklearn.datasets import fetch_20newsgroups_vectorized

#
================================================================================
===
#   导入新闻数据
#
================================================================================
===
data_logistic=fetch_20newsgroups_vectorized()
data_log_x=data_logistic.data
data_log_y=data_logistic.target

#
================================================================================
===
# 分割训练数据集
#
================================================================================
===
x_train,x_test,y_train,y_test=train_test_split(data_log_x,data_log_y,test_
size=0.2,random_state=1)
```

logistic regression

```
from sklearn.linear_model import LogisticRegression

```

参数：

> **penalty**：惩罚项，str类型，可选参数为l1和l2，默认为l2。用于指定惩罚项中使用的规范。newton-cg、sag和lbfgs求解算法只支持L2规范。L1G规范假设的是模型的参数满足拉普拉斯分布，L2假设的模型参数满足高斯分布，所谓的范式就是加上对参数的约束，使得模型更不会过拟合(overfit)，但是如果要说是不是加了约束就会好，这个没有人能回答，只能说，加约束的情况下，理论上应该可以获得泛化能力更强的结果。 **dual**：对偶或原始方法，bool类型，默认为False。对偶方法只用在求解线性多核(liblinear)的L2惩罚项上。当样本数量>样本特征的时候，dual通常设置为False。 **tol**：停止求解的标准，float类型，默认为1e-4。就是求解到多少的时候，停止，认为已经求出最优解。 **c**：正则化系数λ的倒数，float类型，默认为1.0。必须是正浮点型数。像SVM一样，越小的数值表示越强的正则化。 **fit_intercept**：是否存在截距或偏差，bool类型，默认为True。 **intercept_scaling**：仅在正则化项为”liblinear”，且fit_intercept设置为True时有用。float类型，默认为1。 **class_weight**：用于标示分类模型中各种类型的权重，可以是一个字典或者’balanced’字符串，默认为不输入，也就是不考虑权重，即为None。如果选择输入的话，可以选择balanced让类库自己计算类型权重，或者自己输入各个类型的权重。举个例子，比如对于0,1的二元模型，我们可以定义class_weight={0:0.9,1:0.1}，这样类型0的权重为90%，而类型1的权重为10%。如果class_weight选择balanced，那么类库会根据训练样本量来计算权重。某种类型样本量越多，则权重越低，样本量越少，则权重越高。当class_weight为balanced时，类权重计算方法如下：n_samples / (n_classes * np.bincount(y))。n_samples为样本数，n_classes为类别数量，np.bincount(y)会输出每个类的样本数，例如y=[1,0,0,1,1],则

np.bincount(y)=[2,3]。 那么class_weight有什么作用呢？ 在分类模型中，我们经常会遇到两类问题： 第一种是误分类的代价很高。比如对合法用户和非法用户进行分类，将非法用户分类为合法用户的代价很高，我们宁愿将合法用户分类为非法用户，这时可以人工再甄别，但是却不愿将非法用户分类为合法用户。这时，我们可以适当提高非法用户的权重。 第二种是样本是高度失衡的，比如我们有合法用户和非法用户的二元样本数据10000条，里面合法用户有9995条，非法用户只有5条，如果我们不考虑权重，则我们可以将所有的测试集都预测为合法用户，这样预测准确率理论上有99.95%，但是却没有任何意义。这时，我们可以选择balanced，让类库自动提高非法用户样本的权重。提高了某种分类的权重，相比不考虑权重，会有更多的样本分类划分到高权重的类别，从而可以解决上面两类问题。 **random_state**：随机数种子，int类型，可选参数，默认为无，仅在正则化优化算法为sag,liblinear时有用。 **solver**：优化算法选择参数，只有五个可选参数，即newton-cg,lbfgs,liblinear,sag,saga。默认为liblinear。solver参数决定了我们对逻辑回归损失函数的优化方法，有四种算法可以选择，分别是： 1.liblinear：使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。 2.lbfgs：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。 3.newton-cg：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。 4.sag：即随机平均梯度下降，是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，适合于样本数据多的时候。 5.saga：线性收敛的随机优化算法的的变重。

总结： liblinear适用于小数据集，而sag和saga适用于大数据集因为速度更快。 对于多分类问题，只有newton-cg,sag,saga和lbfgs能够处理多项损失，而liblinear受限于一对剩余(OvR)。啥意思，就是用liblinear的时候，如果是多分类问题，得先把一种类别作为一个类别，剩余的所有类别作为另外一个类别。一次类推，遍历所有类别，进行分类。 newton-cg,sag和lbfgs这三种优化算法时都需要损失函数的一阶或者二阶连续导数，因此不能用于没有连续导数的L1正则化，只能用于L2正则化。而liblinear和saga通吃L1正则化和L2正则化。 同时，sag每次仅仅使用了部分样本进行梯度迭代，所以当样本量少的时候不要选择它，而如果样本量非常大，比如大于10万，sag是第一选择。但是sag不能用于L1正则化，所以当你有大量的样本，又需要L1正则化的话就要自己做取舍了。要么通过对样本采样来降低样本量，要么回到L2正则化。 从上面的描述，大家可能觉得，既然newton-cg, lbfgs和sag这么多限制，如果不是大样本，我们选择liblinear不就行了嘛！错，因为liblinear也有自己的弱点！我们知道，逻辑回归有二元逻辑回归和多元逻辑回归。对于多元逻辑回归常见的有one-vs-rest(OvR)和many-vs-many(MvM)两种。而MvM一般比OvR分类相对准确一些。郁闷的是liblinear只支持OvR，不支持MvM，这样如果我们需要相对精确的多元逻辑回归时，就不能选择liblinear了。也意味着如果我们需要相对精确的多元逻辑回归不能使用L1正则化了。

max_iter：算法收敛最大迭代次数，int类型，默认为10。仅在正则化优化算法为newton-cg, sag和lbfgs才有用，算法收敛的最大迭代次数。 **multi_class**：分类方式选择参数，str类型，可选参数为ovr和multinomial，默认为ovr。ovr即前面提到的one-vs-rest(OvR)，而multinomial即前面提到的many-vs-many(MvM)。如果是二元逻辑回归，ovr和multinomial并没有任何区别，区别主要在多元逻辑回归上。 OvR和MvM有什么不同？ OvR的思想很简单，无论你是多少元逻辑回归，我们都可以看做二元逻辑回归。具体做法是，对于第K类的分类决策，我们把所有第K类的样本作为正例，除了第K类样本以外的所有样本都作为负例，然后在上面做二元逻辑回归，得到第K类的分类模型。其他类的分类模型获得以此类推。 而MvM则相对复杂，这里举MvM的特例one-vs-one(OvO)作讲解。如果模型有T类，我们每次在所有的T类样本里面选择两类样本出来，不妨记为T1类和T2类，把所有的输出为T1和T2的样本放在一起，把T1作为正例，T2作为负例，进行二元逻辑回归，得到模型参数。我们一共需要T(T-1)/2次分类。 可以看出OvR相对简单，但分类效果相对略差（这里指大多数样本分布情况，某些样本分布下OvR可能更好）。而MvM分类相对精

> 确，但是分类速度没有OvR快。如果选择了ovr，则4种损失函数的优化方法liblinear，newton-cg,lbfgs和sag都可以选择。但是如果选择了multinomial,则只能选择newton-cg, lbfgs和sag了。
>
> **verbose**：日志冗长度，int类型。默认为0。就是不输出训练过程，1的时候偶尔输出结果，大于1，对于每个子模型都输出。 **warm_start**：热启动参数，bool类型。默认为False。如果为True，则下一次训练是以追加树的形式进行（重新使用上一次的调用作为初始化）。
>
> **n_jobs**：并行数。int类型，默认为1。1的时候，用CPU的一个内核运行程序，2的时候，用CPU的2个内核运行程序。为-1的时候，用所有CPU的内核运行程序。
>
> 参考：https://blog.csdn.net/jark_/article/details/78342644