# 不太想说话

## Regularization 和 L1、L2 罚项

2018-12-04 | 更新于 2018-12-04 | 机器学习

疑惑了好久的术语，借机研究一下。

本文大部分翻译自“Regularization: Simple Definition, L1 & L2 Penalties”。

另外参考：

- 如何理解Normalization，Regularization 和 standardization?
- L1 Penalty and Sparsity in Logistic Regression

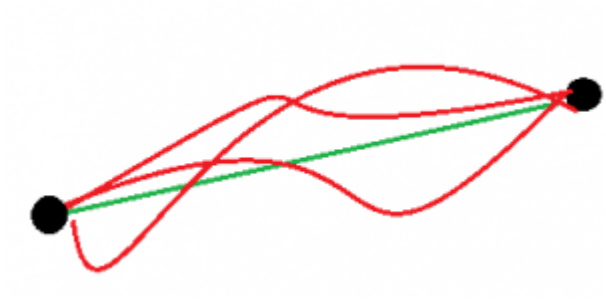
## 什么是正则化

正则化是为了避免 过拟合（overfitting） 采取的一种手段。它通过对回归系数中的较大值做罚项来实现。简单来说，它能够**减少参数、缩小（简化）模型**。这样更加流形化、更加 简约（parsimonious） 的模型往往会在实际预测的时候表现更加良好。正则化给更加复杂的模型加上罚项后，再将潜在模型从过拟合最轻到最严重排序，有最小“过拟合”得分的模型一般认为在预测能力上是最佳的。

## 正则化为什么是必要的

因为 最小二乘法（least-squares-regression-line） 中，平方残差和（residual-sum-squares） 最小化的过程中会导致不稳定。这点在模型中存在 多重共线性（multicollinearity） 时表现得尤为突出。但是仅仅在模型拟合的时候，就出现了明显的缺陷：任何数据集都可以在一个模型上拟合，即使它极其复杂。

例如，拿一个只具有两个点的数据集来说，最简单的模型是在两点间连线，或者一个一阶多项式。但是无数其他模型都可以在这个数据集上拟合：二阶多项式、三阶多项式等等。



在小数据集上拟合往往会得到很复杂、过拟合的模型，而简单的模型则表现不佳。例子中的两个点在同一条直线上不代表新增的点也在这条直线上，而且大概率不在。所以简单来说，正则化对复杂模型进行罚项，而在不牺牲模型预测能力的情况下更偏好简单（回归系数更小）的模型。

## 惩罚方法

正则化的作用是将数据逐渐 逼近（biasing） 某些特定值（例如接近于零的极小值）。这样的逼近是通过增加一个 _调整参数_ 来改变下式中的 **R** 实现的：

$$min \sum_{i=1}^{n} J(x_i, y_i) + \lambda R(f)$$

- **L1 正则化** 增加了一项大小等于系数**离散程度绝对值**的 L1 罚项。可以通过 L1 产生稀疏的模型（如系数很少的模型）；一些系数可以归零并去除。**Lasso 回归** 用的就是这个方法。
- **L2 正则化** 增加了一项大小等于系数离散程度平方的 L2 罚项。L2 **不会**产生稀疏模型，所有稀疏会被相同的因子缩小（但并不会剔除其中任意一个）。 **岭回归** 和 **支持向量机（SVM）** 使用这种方法。
- **Elastic nets** 综合了 L1 & L2 方法，但是增加了一个超参数。（参见 Zou and Hastie 的这篇文章）

也即分别的，
带 **L1** 罚项的 logistic 回归主要解决以下优化问题：

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

而带 **L2** 罚项的 logistic 回归主要解决以下优化问题：

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

## 和 Normalization、Standardization 的区别

1. normalization和standardization差不多，都是把数据进行前处理，从而**使数值都落入到统一的数值范围**，从而在建模过程中，各个特征量没差别对待。**normalization** 一般是把数据限定在需要的范围，比如一般都是 **[0, 1]**，从而**消除了数据量纲对建模的影响**。**standardization** 一般是指将数据**正态化**，使平均值0方差为1。因此normalization和standardization 是**针对数据而言的**，消除一些数值差异带来的特种重要性偏见。经过归一化的数据，能**加快训练速度，促进算法的收敛**。

2. 而regularization是**在cost function里面加罚项项，增加建模的模糊性**，从而把捕捉到的趋势从局部细微趋势，调整到整体大概趋势。虽然一定程度上的放宽了建模要求，但是能**有效防止over-fitting的问题**，增加模型准确性。因此，regularization是**针对模型而言**。

**normalization 的方法主要有：**

1. 最大最小值
2. 对数
3. 反正切

## 为什么 Regularization 有效

引用知乎某位答友的说法：

> 不用深究其为何work，研究起来就是泛函分析。

大佬请便。

## 参考代码

```
# Authors: Alexandre Gramfort <alexandre.gramfort@inria.fr>
#          Mathieu Blondel <mathieu@mblondel.org>
#          Andreas Mueller <amueller@ais.uni-bonn.de>
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn import datasets
# Scaler 是 preprocessing 的步骤
from sklearn.preprocessing import StandardScaler

# 这句话使输出中不会有省略号
```

```
np.set_printoptions(threshold=np.nan)

# 加载了手写数字图片数据集（详见最下方链接），
# data    位图每个像素对应的向量
# target 图片对应的实际数字
digits = datasets.load_digits()
X, y = digits.data, digits.target

# 将所有值正态分布到 μ=0，σ²=1 的空间
X = StandardScaler().fit_transform(X)

# 按标签是否大于 4 来做二分类（大于4的值为1否则为0）
y = (y > 4).astype(np.int)

# L1、L2 算法的参数 C，分别取 (100, 1, 0.01) 试验
for i, C in enumerate((100, 1, 0.01)):
    #  （各参数解释见下）
    clf_l1_LR = LogisticRegression(C=C, penalty='l1', tol=0.01)
    clf_l2_LR = LogisticRegression(C=C, penalty='l2', tol=0.01)
    clf_l1_LR.fit(X, y)
    clf_l2_LR.fit(X, y)

    # numpy.ravel() 见最下链接
    # 取得系数并打平
    coef_l1_LR = clf_l1_LR.coef_.ravel()
    coef_l2_LR = clf_l2_LR.coef_.ravel()

    # 因为 L1 正则化产生稀疏矩阵，coef_l1_LR 包含 0

    # 计算两系数矩阵的稀疏度
    sparsity_l1_LR = np.mean(coef_l1_LR == 0) * 100
    sparsity_l2_LR = np.mean(coef_l2_LR == 0) * 100

    # LogisticRegression.score() 返回经训练后的回归器在定特征和标签上的平均正确率
    print("C=%.2f" % C)
    print("Sparsity with L1 penalty: %.2f%%" % float(sparsity_l1_LR))
    print("score with L1 penalty: %.4f" % clf_l1_LR.score(X, y))
    print("Sparsity with L2 penalty: %.2f%%" % float(sparsity_l2_LR))
    print("score with L2 penalty: %.4f" % clf_l2_LR.score(X, y))

    # 作图
    l1_plot = plt.subplot(3, 2, 2 * i + 1)
    l2_plot = plt.subplot(3, 2, 2 * (i + 1))
    if i == 0:
```

```
        l1_plot.set_title("L1 penalty")
        l2_plot.set_title("L2 penalty")

    # 把系数矩阵还原后显示为灰度图
    l1_plot.imshow(np.abs(coef_l1_LR.reshape(8, 8)), interpolation='nearest',
                   cmap='binary', vmax=1, vmin=0)
    l2_plot.imshow(np.abs(coef_l2_LR.reshape(8, 8)), interpolation='nearest',
                   cmap='binary', vmax=1, vmin=0)
    plt.text(-8, 3, "C = %.2f" % C)

    # 不显示坐标轴
    l1_plot.set_xticks(())
    l1_plot.set_yticks(())
    l2_plot.set_xticks(())
    l2_plot.set_yticks(())

plt.show()
```

- sklearn.datasets.load_digits()
- numpy.ravel()
- sklearn.datasets.load_digits()
- numpy.ravel()
- sklearn.linear_model.LogisticRegression
- LogisticRegression 主要参数：
  - penalty : str, 'l1' or 'l2', default: 'l2'

    指定正则化罚项的方式。 'newton-cg', 'lbfgs' 和 'sag' 算法只能使用 L2 罚项。
  - tol : float, default: 1e-4

    算法停止的边界，越大停止越快。
  - C : float, default: 1.0

    正则化强度的逆（inverse），必须是正的浮点数。

    正如支持向量机，越小的值代表越大的正则化强度。
  - solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default: 'liblinear'

    用于问题优化的算法。

**运行结果：**

```
C=100.00
Sparsity with L1 penalty: 6.25%
score with L1 penalty: 0.9110
Sparsity with L2 penalty: 4.69%
score with L2 penalty: 0.9098

```

```
C=1.00
Sparsity with L1 penalty: 9.38%
score with L1 penalty: 0.9104
Sparsity with L2 penalty: 4.69%
score with L2 penalty: 0.9093

C=0.01
Sparsity with L1 penalty: 85.94%
score with L1 penalty: 0.8631
Sparsity with L2 penalty: 4.69%
score with L2 penalty: 0.8915
```

**结果分析：**

- C 值代表正则化强度，必须为正值，且越接近零，正则化强度越大，指数级缩放。
- 无论正则化强度怎么变，L2 正则化的系数矩阵稀疏度不变，而 L1 正则化系数矩阵稀疏度和正则化强度呈正相关。
- L2 正则化相比 L1 正则化能保留更多的细节信息，在正则化强度较大是相对表现更好一些。

## 源码分析

“liblinear” 应用了**坐标下降算法**（Coordinate Descent, **CD**），并基于 scikit-learn 内附的高性能 C++ 库 LIBLINEAR library 实现。不过 CD 算法训练的模型不是真正意义上的多分类模型，而是基于 “**one-vs-rest**” 思想分解了这个优化问题，为每个类别都训练了一个二元分类器。因为实现在底层，所以求解器的 `LogisticRegression` 实例对象表面上看是一个多元分类器。

`sklearn.svm.l1_min_c` 可以计算使用 L1 罚项时 C 的下界，以避免模型为空（即全部特征分量的权重为零）。

- 对于**小数据集**，应该使用 ‘liblinear’，而在**大（高维度）数据集**上使用 ‘sag’ 或 ‘saga’ 会快一些；
- 对于**多分类**问题，**‘liblinear’ 只能处理 one-vs-rest 的情形**，只有 ‘newton-cg’, ‘sag’, ‘saga’ 和 ‘lbfgs’ 可以**计算多项损失**，这些求解器的参数 `multi_class` 设为 “multinomial” 即可训练一个**真正的多项式** logistic 回归 [link] ，其预测的概率比默认的 “one-vs-rest” 设定**更为准确**。
- ‘newton-cg’, ‘lbfgs’ 和 ‘sag’ 算法**只能使用 L2 罚项**，而 ‘liblinear’ 和 ‘saga’ 使用 L1 罚项。
- “sag” 求解器基于**平均随机梯度下降算法（Stochastic Average Gradient descent）** [link]。在大数据集上的表现更快，大数据集指样本量大且特征数多。
- “saga” 求解器 [link] 是 “sag” 的一类变体，它支持**非平滑（non-smooth）**的 L1 正则选项 `penalty="l1"` 。因此对于**稀疏**多项式 logistic 回归 ，往往选用该求解器。
- **“saga” 一般都是最佳的选择**，但出于一些历史遗留原因默认的是 “liblinear” 。
- 请注意 ‘sag’ 和 ‘saga’ 快速收敛的性质仅在具有差不多大小的 feature 数值时得到保证（译者注：即**所有 feature 差不多大**）。你可以通过 sklearn.preprocessing 来对你的数据进行**预处理**。

继续完善……

http://sklearn.apachecn.org/cn/0.19.0/modules/linear_model.html#logistic

———— EOF ————

# 线性回归 # sklearn

Boosting 和 XGBoost 和 LightGBM 和其他

决策树基础