

机器学习实战教程（七）：Logistic回归实战篇之预测病马死亡率

🕒 2017年11月9日 21:07:51

💬 66

👁 10,903

🌡 °C

✎ 编辑



一、前言

本文对梯度上升算法和改进的随机梯度上升算法进行了对比，总结了各自的优缺点，并对sklearn.linear_model.LogisticRegression进行了详细介绍。

二、改进的随机梯度上升算法

梯度上升算法在每次更新回归系数(最优参数)时，都需要遍历整个数据集。可以看一下我们之前写的梯度上升算法：

Pyth

```
1 def gradAscent(dataMatIn, classLabels):
2     dataMatrix = np.mat(dataMatIn)
3     labelMat = np.mat(classLabels).transpose()
4     m, n = np.shape(dataMatrix)
5     alpha = 0.01
6     maxCycles = 500
7     weights = np.ones((n,1))
8     for k in range(maxCycles):
9         h = sigmoid(dataMatrix * weights)
10        error = labelMat - h
11        weights = weights + alpha * dataMatrix.transpose() * error
12    return weights.getA(),weights_array
```

#转换成numpy的mat
#转换成numpy的mat,并进行转置
#返回dataMatrix的大小。m为行数,n为列数。
#移动步长,也就是学习速率,控制更新的幅度。
#最大迭代次数
#梯度上升矢量化公式
#将矩阵转换为数组,返回权重数组

假设，我们使用的数据集一共有100个样本。那么，dataMatrix就是一个100*3的矩阵。每次计算h的时候，都要计算dataMatrix*weights这个矩阵乘法运算，要进行100*3次乘法运算和100*2次加法运算。同理，更新回归系数(最优参数)weights时，也需要用到整个数据集，要进行矩阵乘法运算。总而言之，该方法处理100个左右的数据集时尚可，但如果数十亿样本和成千上万的特征，那么该方法的计算复杂度就太高了。因此，需要对算法进行改进，我们每次更新回归系数(最优参数)的时候，能不能不用所有样本呢？一次只用一个样本点去更新回归系数(最优参数)？这样就可以有效减少计算量了，这种方法就叫做随机梯度上升算法。

1、随机梯度上升算法

让我们直接看代码：

Pyth

```
1 def stocGradAscent1(dataMatrix, classLabels, numIter=150):
2     m,n = np.shape(dataMatrix)
3     weights = np.ones(n)
4     for j in range(numIter):
5         dataIndex = list(range(m))
6         for i in range(m):
7             alpha = 4/(1.0+j+i)+0.01
8             randIndex = int(random.uniform(0,len(dataIndex)))
9             h = sigmoid(sum(dataMatrix[dataIndex[randIndex]]*weights))
10            error = classLabels[dataIndex[randIndex]] - h
11            weights = weights + alpha * error * dataMatrix[dataIndex[randIndex]]
12            del(dataIndex[randIndex])
13    return weights
```

#返回dataMatrix的大小。m为行数,n为列数。
#参数初始化
#降低alpha的大小，每次减小1/(j+i)。
#随机选取样本
#选择随机选取的一个样本，计算h
#计算误差
#更新回归系数
#删除已经使用的样本
#返回

该算法第一个改进之处在于，alpha在每次迭代的时候都会调整，并且，虽然alpha会随着迭代次数不断减小，但永远不会减小到0，因为这里还存在一个常数项。必须这样做的原因是为了保证在多次迭代之后新数据仍然具有一定的影响。如果需要处理的问题是动态变化的，那么可以适当加大上述常数项，来确保新的值获得更大的回归系数。另一点值得注意的是，在降低alpha的函数中，alpha每次减少1/(j+i)，其中j是迭代次数，i是样本点的下标。第二个改进的地方在于更新回归系数(最优参数)时，只使用一个样本点，并且选择的样本点是随机的，每次迭代不使用已经用过的样本点。这样的方法，就有效地减少了计算量，并保证了回归效果。

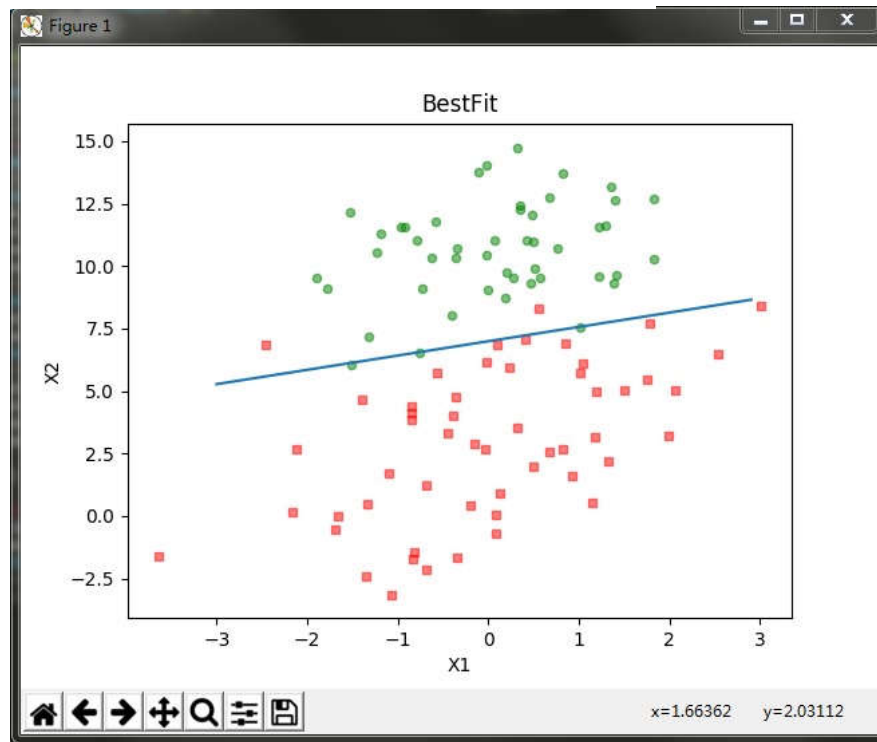
编写代码如下，看下改进的随机梯度上升算法分类效果如何：

```

1  #-*- coding:UTF-8 -*-
2  from matplotlib.font_manager import FontProperties
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import random
6
7  """
8  函数说明:加载数据
9
10 Parameters:
11     无
12 Returns:
13     dataMat - 数据列表
14     labelMat - 标签列表
15 Author:
16     Jack Cui
17 Blog:
18     http://blog.csdn.net/c406495762
19 Zhihu:
20     https://www.zhihu.com/people/Jack--Cui/
21 Modify:
22     2017-08-28
23 """
24 def loadDataSet():
25     dataMat = []
26     labelMat = []
27     fr = open('testSet.txt')
28     for line in fr.readlines():
29         lineArr = line.strip().split()
30         dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
31         labelMat.append(int(lineArr[2]))
32     fr.close()
33     return dataMat, labelMat
34
35 """
36 函数说明:sigmoid函数
37
38 Parameters:
39     inX - 数据
40 Returns:
41     sigmoid函数
42 Author:
43     Jack Cui
44 Blog:
45     http://blog.csdn.net/c406495762
46 Zhihu:
47     https://www.zhihu.com/people/Jack--Cui/
48 Modify:
49     2017-08-28
50 """
51 def sigmoid(inX):
52     return 1.0 / (1 + np.exp(-inX))
53
54 """
55 函数说明:绘制数据集
56
57 Parameters:
58     weights - 权重参数数组
59 Returns:
60     无
61 Author:
62     Jack Cui
63 Blog:
64     http://blog.csdn.net/c406495762
65 Zhihu:
66     https://www.zhihu.com/people/Jack--Cui/
67 Modify:

```

代码运行结果：



2、回归系数与迭代次数的关系

可以看到分类效果也是不错的。不过，从这个分类结果中，我们不好看出迭代次数和回归系数的关系，也就不能直观的看到每个回归方法的收敛情况。因此，我们编写程序，绘制出回归系数和迭代次数的关系曲线：

```

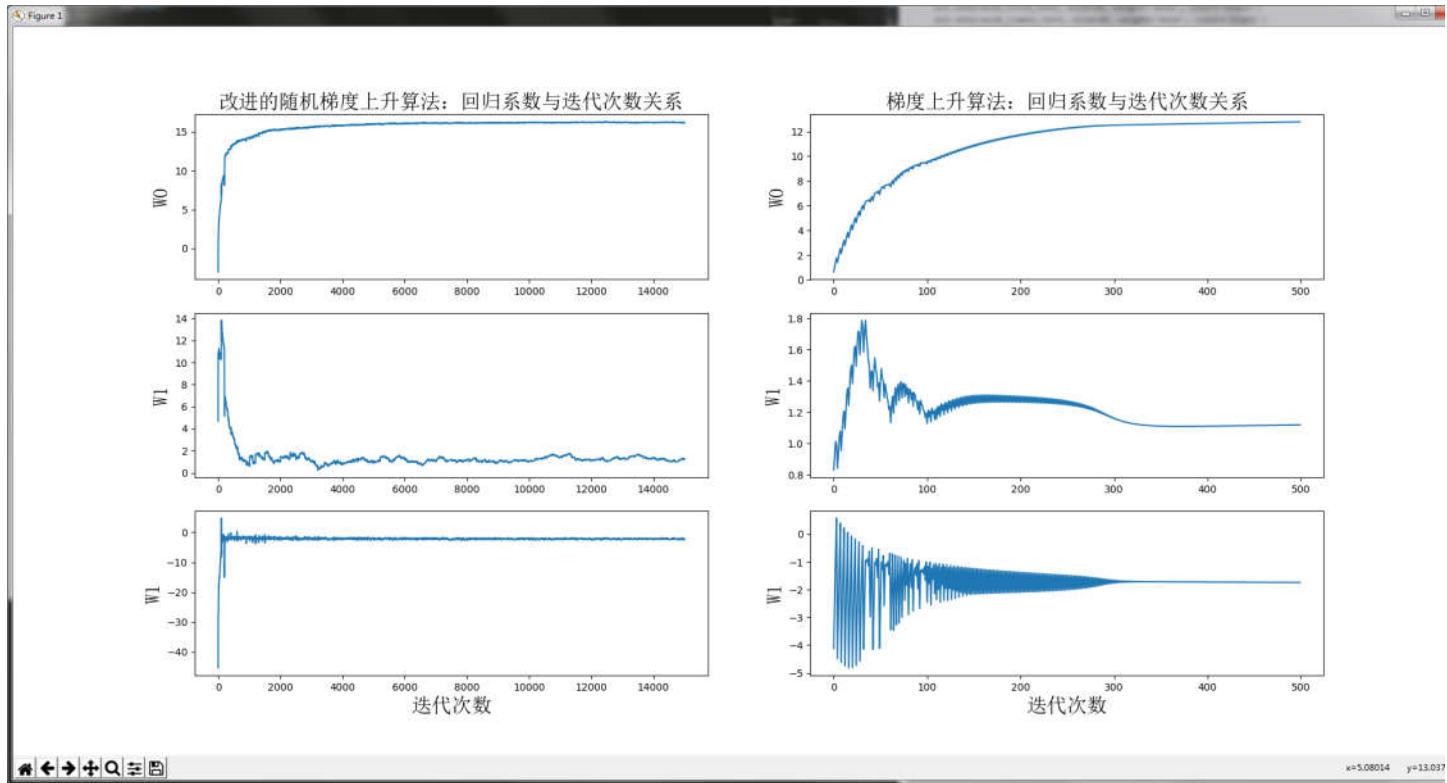
1  # -*- coding:UTF-8 -*-
2  from matplotlib.font_manager import FontProperties
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import random
6
7
8  """
9  函数说明:加载数据
10
11 Parameters:
12  无
13 Returns:
14  dataMat - 数据列表
15  labelMat - 标签列表
16 Author:
17  Jack Cui
18 Blog:
19  http://blog.csdn.net/c406495762
20 Zhihu:
21  https://www.zhihu.com/people/Jack--Cui/
22 Modify:
23  2017-08-28
24 """
25 def loadDataSet():
26     dataMat = []
27     labelMat = []
28     fr = open('testSet.txt')
29     for line in fr.readlines():
30         lineArr = line.strip().split()
31         dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
32         labelMat.append(int(lineArr[2]))
33     fr.close()
34     return dataMat, labelMat
35
36 """
37 函数说明:sigmoid函数
38
39 Parameters:
40  inX - 数据
41 Returns:
42  sigmoid函数
43 Author:
44  Jack Cui
45 Blog:
46  http://blog.csdn.net/c406495762
47 Zhihu:
48  https://www.zhihu.com/people/Jack--Cui/
49 Modify:
50  2017-08-28
51 """
52 def sigmoid(inX):
53     return 1.0 / (1 + np.exp(-inX))
54
55 """
56 函数说明:梯度上升算法
57
58 Parameters:

```

Pyth

```
59 dataMatIn - 数据集
60 classLabels - 数据标签
61 Returns:
62     weights.getA() - 求得的权重数组(最优参数)
63     weights_array - 每次更新的回归系数
64 Author:
65     Jack Cui
66 Blog:
67     http://blog.csdn.net/c496495762
```

运行结果如下：



由于改进的随机梯度上升算法，随机选取样本点，所以每次的运行结果是不同的。但是大体趋势是一样的。我们改进的随机梯度上升算法收敛效果更好。为什么这么说呢？让我们分析一下。我们一共有100个样本点，改进的随机梯度上升算法迭代次数为150。而上图显示15000次迭代次数的原因是，每一次样本就更新一下回归系数。因此，迭代150次，相当于更新回归系数150*100=15000次。简而言之，迭代150次，更新1.5万次回归参数。从上图左侧改进随机梯度上升算法回归效果中可以看出，其实在更新2000次回归系数的时候，已经收敛了。相当于遍历整个数据集20次的时候，回归系数已收敛。训练已完成。

再让我们看看上图右侧的梯度上升算法回归效果，梯度上升算法每次更新回归系数都要遍历整个数据集。从图中可以看出，当迭代次数为300多次的时候，回归系数才收敛。凑个整，就当它在遍历整个数据集300次的时候已经收敛好了。

没有对比就没有伤害，改进的随机梯度上升算法，在遍历数据集的第20次开始收敛。而梯度上升算法，在遍历数据集的第300次才开始收敛。想像一下，大量数据的情况下，谁更牛逼？

三、从疝气病症预测病马的死亡率

1、实战背景

本次实战内容，将使用Logistic回归来预测患疝气病的马的存活问题。原始数据集下载地址：[数据集下载](#)

这里的数据包含了368个样本和28个特征。这种病不一定源自马的肠胃问题，其他问题也可能引发马疝病。该数据集中包含了医院检测马疝病的一些指标，有的指标比较主观，有的指标难以测量，例如马的疼痛级别。另外需要说明的是，除了部分指标主观和难以测量外，该数据还存在一个问题，数据集里有30%的值是缺失的。下面将首先介绍如何处理数据集中的数据缺失问题，然后再利用Logistic回归和随机梯度上升算法来预测病马的生死。

2、准备数据

数据中的缺失值是一个非常棘手的问题，很多文献都致力于解决这个问题。那么，数据缺失究竟带来了什么问题？假设有100个样本和20个特征，这些数据都是机器收集回来的。若机器上的某个传感器损坏导致一个特征无效时该怎么办？它们是否还可用于？答案是肯定的。因为有时候数据相当昂贵，扔掉重新获取都是不可取的，所以必须采用一些方法来解决这个问题。下面给出了一些可选的做法：

- 使用可用特征的均值来填补缺失值；
- 使用特殊值来填补缺失值，如-1；
- 忽略有缺失值的样本；

- 使用相似样本的均值添补缺失值；
- 使用另外的机器学习算法预测缺失值。

预处理数据做两件事：

- 如果测试集中一条数据的特征值已经确实，那么我们选择实数0来替换所有缺失值，因为本文使用Logistic回归。因此这样做不会影响回归系数的值。 $\text{sigmoid}(0)=0.5$ ，即它对结果的预测不具有任何倾向性。
- 如果测试集中一条数据的类别标签已经缺失，那么我们将该类别数据丢弃，因为类别标签与特征不同，很难确定采用某个合适的值来替换。

原始的数据集经过处理，保存为两个文件：horseColicTest.txt和horseColicTraining.txt。已经处理好的“干净”可用的数据集下载地址：

- 数据集1
- 数据集2

有了这些数据集，我们只需要一个Logistic分类器，就可以利用该分类器来预测病马的生死问题了。

3、使用Python构建Logistic回归分类器

在使用Sklearn构建Logistic回归分类器之前，我们先用自己写的改进的随机梯度上升算法进行预测，先热热身。使用Logistic回归方法进行分类并不需要做很多工作，所需做的只是把测试集上每个特征向量乘以最优化方法得来的回归系数，再将乘积结果求和，最后输入到Sigmoid函数中即可。如果对应Sigmoid值大于0.5就预测类别标签为1，否则为0。

Python

```
1  #-*- coding:UTF-8 -*-
2  import numpy as np
3  import random
4
5  """
6  函数说明:sigmoid函数
7
8  Parameters:
9      inX - 数据
10 Returns:
11     sigmoid函数
12 Author:
13     Jack Cui
14 Blog:
15     http://blog.csdn.net/c406495762
16 Zhihu:
17     https://www.zhihu.com/people/Jack--Cui/
18 Modify:
19     2017-09-05
20 """
21 def sigmoid(inX):
22     return 1.0 / (1 + np.exp(-inX))
23
24 """
25 函数说明:改进的随机梯度上升算法
26
27 Parameters:
28     dataMatrix - 数据数组
29     classLabels - 数据标签
30     numIter - 迭代次数
31 Returns:
32     weights - 求得的回归系数数组(最优参数)
33 Author:
34     Jack Cui
35 Blog:
36     http://blog.csdn.net/c406495762
37 Zhihu:
38     https://www.zhihu.com/people/Jack--Cui/
39 Modify:
40     2017-09-05
41 """
42 def stocGradAscent1(dataMatrix, classLabels, numIter=150):
43     m,n = np.shape(dataMatrix)
44     weights = np.ones(n)
45     for j in range(numIter):
46         dataIndex = list(range(m))
47         for i in range(m):
48             alpha = 4/(1.0+j+i)+0.01
49             randIndex = int(random.uniform(0,len(dataIndex)))
50             h = sigmoid(sum(dataMatrix[dataIndex[randIndex]]*weights))
51             error = classLabels[dataIndex[randIndex]] - h
52             weights = weights + alpha * error * dataMatrix[dataIndex[randIndex]]
53             del(dataIndex[randIndex])
54     return weights
55
56 """
57 函数说明:使用Python写的Logistic分类器做预测
58
59 Parameters:
60     无
61 Returns:
62     无
63 Author:
64     Jack Cui
65 Blog:
66     http://blog.csdn.net/c406495762
67 ... ..
```

运行结果如下：

```

112
113 def classifyVector(inX, weights):
114     prob = sigmoid(sum(inX*weights))
115     if prob > 0.5: return 1.0
116     else: return 0.0
117
118 if __name__ == '__main__':
119     colicTest()

```

测试集错误率为：34.33%
[Finished in 1.9s]

错误率还是蛮高的，而且耗时1.9s，并且每次运行的错误率也是不同的，错误率高的时候可能达到40%多。为啥这样？首先，因为数据集本身有30%数据缺失，这个是不能避免的。另一个主要原因是，我们使用的是改进的随机梯度上升算法，因为数据集本身就很小，就几百的数据量。用改进的随机梯度上升算法显然不合适。让我们再试试梯度上升算法，看看它的效果如何？

```

1  #-*- coding:UTF-8 -*-
2  import numpy as np
3  import random
4
5  """
6  函数说明:sigmoid函数
7
8  Parameters:
9      inX - 数据
10 Returns:
11     sigmoid函数
12 Author:
13     Jack Cui
14 Blog:
15     http://blog.csdn.net/c406495762
16 Zhihu:
17     https://www.zhihu.com/people/Jack--Cui/
18 Modify:
19     2017-09-05
20 """
21 def sigmoid(inX):
22     return 1.0 / (1 + np.exp(-inX))
23
24 """
25 函数说明:梯度上升算法
26
27 Parameters:
28     dataMatIn - 数据集
29     classLabels - 数据标签
30 Returns:
31     weights.getA() - 求得的权重数组(最优参数)
32 Author:
33     Jack Cui
34 Blog:
35     http://blog.csdn.net/c406495762
36 Zhihu:
37     https://www.zhihu.com/people/Jack--Cui/
38 Modify:
39     2017-08-28
40 """
41 def gradAscent(dataMatIn, classLabels):
42     dataMatrix = np.mat(dataMatIn)
43     labelMat = np.mat(classLabels).transpose()
44     m, n = np.shape(dataMatrix)
45     alpha = 0.01
46     maxCycles = 500
47     weights = np.ones((n,1))
48     for k in range(maxCycles):
49         h = sigmoid(dataMatrix * weights)
50         error = labelMat - h
51         weights = weights + alpha * dataMatrix.transpose() * error
52     return weights.getA()
53
54
55 """
56 函数说明:使用Python写的Logistic分类器做预测
57
58 Parameters:
59     无
60 Returns:
61     无
62 Author:
63     Jack Cui
64 Blog:
65     http://blog.csdn.net/c406495762
66 Zhihu:

```

运行结果如下：


```
146 def classifyVector(inX, weights):
147     prob = sigmoid(sum(inX*weights))
148     if prob > 0.5: return 1.0
149     else: return 0.0
150
151 if __name__ == '__main__':
152     colicTest()
```

测试集错误率为：28.36%
[Finished in 0.5s]

可以看到算法耗时减少了，错误率稳定且较低。很显然，使用随机梯度上升算法，反而得不偿失了。所以可以得到如下结论：

- 当数据集较小时，我们使用梯度上升算法
- 当数据集较大时，我们使用改进的随机梯度上升算法

对应的，在Sklearn中，我们就可以根据数据情况选择优化算法，比如数据较小时，我们使用liblinear，数据较大时，我们使用sag和saga。

四、使用Sklearn构建Logistic回归分类器

开始新一轮的征程，让我们看下Sklearn的Logistic回归分类器！

官方英文文档地址：[点我查看](#)

sklearn.linear_model模块提供了很多模型供我们使用，比如Logistic回归、Lasso回归、贝叶斯脊回归等，可见需要学习的东西还有很多很多。本章，我们使用LogisticRegression。

sklearn.linear_model: Generalized Linear Models

The `sklearn.linear_model` module implements generalized linear models. It includes Ridge regression, Bayesian Regression, Lasso and Elastic Net estimators computed with Least Angle Regression and coordinate descent. It also implements Stochastic Gradient Descent related algorithms.

User guide: See the Generalized Linear Models section for further details.

<code>linear_model.ARDRegression</code> ([n_iter, tol, ...])	Bayesian ARD regression.
<code>linear_model.BayesianRidge</code> ([n_iter, tol, ...])	Bayesian ridge regression
<code>linear_model.ElasticNet</code> ([alpha, l1_ratio, ...])	Linear regression with combined L1 and L2 priors as regularizer.
<code>linear_model.ElasticNetCV</code> ([l1_ratio, eps, ...])	Elastic Net model with iterative fitting along a regularization path
<code>linear_model.HuberRegressor</code> ([epsilon, ...])	Linear regression model that is robust to outliers.
<code>linear_model.Lars</code> ([fit_intercept, verbose, ...])	Least Angle Regression model a.k.a.
<code>linear_model.LarsCV</code> ([fit_intercept, ...])	Cross-validated Least Angle Regression model
<code>linear_model.Lasso</code> ([alpha, fit_intercept, ...])	Linear Model trained with L1 prior as regularizer (aka the Lasso)
<code>linear_model.LassoCV</code> ([eps, n_alphas, ...])	Lasso linear model with iterative fitting along a regularization path
<code>linear_model.LassoLars</code> ([alpha, ...])	Lasso model fit with Least Angle Regression a.k.a.
<code>linear_model.LassoLarsCV</code> ([fit_intercept, ...])	Cross-validated Lasso, using the LARS algorithm
<code>linear_model.LassoLarsIC</code> ([criterion, ...])	Lasso model fit with Lars using BIC or AIC for model selection
<code>linear_model.LinearRegression</code> ([...])	Ordinary least squares Linear Regression.
<code>linear_model.LogisticRegression</code> ([penalty, ...])	Logistic Regression (aka logit, MaxEnt) classifier.
<code>linear_model.LogisticRegressionCV</code> ([Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.
<code>linear_model.MultiTaskLasso</code> ([alpha, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer
<code>linear_model.MultiTaskElasticNet</code> ([alpha, ...])	Multi-task ElasticNet model trained with L1/L2 mixed-norm as regularizer
<code>linear_model.MultiTaskLassoCV</code> ([eps, ...])	Multi-task L1/L2 Lasso with built-in cross-validation.
<code>linear_model.MultiTaskElasticNetCV</code> ([...])	Multi-task L1/L2 ElasticNet with built-in cross-validation.
<code>linear_model.OrthogonalMatchingPursuit</code> ([...])	Orthogonal Matching Pursuit model (OMP)
<code>linear_model.OrthogonalMatchingPursuitCV</code> ([...])	Cross-validated Orthogonal Matching Pursuit model (OMP)
<code>linear_model.PassiveAggressiveClassifier</code> ([...])	Passive Aggressive Classifier
<code>linear_model.PassiveAggressiveRegressor</code> ([C, ...])	Passive Aggressive Regressor
<code>linear_model.Perceptron</code> ([penalty, alpha, ...])	Read more in the User Guide.
<code>linear_model.RANSACRegressor</code> ([...])	RANSAC (RANDOM SAMple Consensus) algorithm.
<code>linear_model.Ridge</code> ([alpha, fit_intercept, ...])	Linear least squares with l2 regularization.
<code>linear_model.RidgeClassifier</code> ([alpha, ...])	Classifier using Ridge regression.
<code>linear_model.RidgeClassifierCV</code> ([alphas, ...])	Ridge classifier with built-in cross-validation.
<code>linear_model.RidgeCV</code> ([alphas, ...])	Ridge regression with built-in cross-validation.
<code>linear_model.SGDClassifier</code> ([loss, penalty, ...])	Linear classifiers (SVM, logistic regression, a.o.) with SGD training.
<code>linear_model.SGDRegressor</code> ([loss, penalty, ...])	Linear model fitted by minimizing a regularized empirical loss with SGD
<code>linear_model.TheilSenRegressor</code> ([...])	Theil-Sen Estimator: robust multivariate regression model.
<code>linear_model.enet_path</code> (X, y[, l1_ratio, ...])	Compute elastic net path with coordinate descent
<code>linear_model.lars_path</code> (X, y[, Xy, Gram, ...])	Compute Least Angle Regression or Lasso path using LARS algorithm [1]
<code>linear_model.lasso_path</code> (X, y[, eps, ...])	Compute Lasso path with coordinate descent
<code>linear_model.lasso_stability_path</code> (*args, ...)	DEPRECATED: The function <code>lasso_stability_path</code> is deprecated in 0.19 and will be removed in 0.21.
<code>linear_model.logistic_regression_path</code> (X, y)	Compute a Logistic Regression model for a list of regularization parameters.
<code>linear_model.orthogonal_mp</code> (X, y[, ...])	Orthogonal Matching Pursuit (OMP)
<code>linear_model.orthogonal_mp_gram</code> (Gram, Xy[, ...])	Gram Orthogonal Matching Pursuit (OMP)

1、LogisticRegression

让我们先看下LogisticRegression这个函数，一共有14个参数：

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression (penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr',
verbose=0, warm_start=False, n_jobs=1)
```

[source]

参数说明如下：

- **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），但是如果要是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:
 - 1.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。
 - 2.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:
 - **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。
 - **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。
 - **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。
 - **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。
 - **saga**: 线性收敛的随机优化算法的的变种。
 - **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。

累死我了....终于写完所有参数了。

除此之外，LogisticRegression也有一些方法供我们使用：

Methods	
<code>decision_function(X)</code>	Predict confidence scores for samples.
<code>densify()</code>	Convert coefficient matrix to dense array format.
<code>fit(X, y[, sample_weight])</code>	Fit the model according to the given training data.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	Predict class labels for samples in X.
<code>predict_log_proba(X)</code>	Log of probability estimates.
<code>predict_proba(X)</code>	Probability estimates.
<code>score(X, y[, sample_weight])</code>	Returns the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>sparsify()</code>	Convert coefficient matrix to sparse format.

有一些方法和MultinomialNB的方法都是类似的，因此不再累述。

对于每个函数的具体使用，可以看下官方文档：[点我查看](#)

同时，如果对于过拟合、正则化、L1范数、L2范数不了解的，可以看这位大牛的博客：[点我查看](#)

2、编写代码

了解到这些，我们就可以编写Sklearn分类器的代码了。代码非常短：

```

1  # -*- coding:UTF-8 -*-
2  from sklearn.linear_model import LogisticRegression
3
4  """
5  函数说明:使用Sklearn构建Logistic回归分类器
6
7  Parameters:
8      无
9  Returns:
10     无
11  Author:
12     Jack Cui
13  Blog:
14     http://blog.csdn.net/c406495762
15  Zhihu:
16     https://www.zhihu.com/people/Jack--Cui/
17  Modify:
18     2017-09-05
19  """
20  def colicSklearn():
21      frTrain = open('horseColicTraining.txt')
22      frTest = open('horseColicTest.txt')
23      trainingSet = []; trainingLabels = []
24      testSet = []; testLabels = []
25      for line in frTrain.readlines():
26          currLine = line.strip().split('\t')
27          lineArr = []
28          for i in range(len(currLine)-1):
29              lineArr.append(float(currLine[i]))
30          trainingSet.append(lineArr)
31          trainingLabels.append(float(currLine[-1]))
32      for line in frTest.readlines():
33          currLine = line.strip().split('\t')
34          lineArr = []
35          for i in range(len(currLine)-1):
36              lineArr.append(float(currLine[i]))
37          testSet.append(lineArr)
38          testLabels.append(float(currLine[-1]))
39      classifier = LogisticRegression(solver='liblinear',max_iter=10).fit(trainingSet, trainingLabels)
40      test_accuracy = classifier.score(testSet, testLabels) * 100
41      print('正确率:%f%%' % test_accuracy)
42
43  if __name__ == '__main__':
44      colicSklearn()

```

运行结果如下：

```

189     print('正确率:%f%%' % test_accu
190
191 if __name__ == '__main__':
192     colicSklearn()

```

正确率:73.134328%
[Finished in 0.8s]

可以看到，正确率又高一些了。更改solver参数，比如设置为sag，使用随机平均梯度下降算法，看一看效果。你会发现，有警告了。

```

185     testSet.append(lineArr)
186     testLabels.append(float(currLine[-1]))
187     classifier = LogisticRegression(solver='sag', max_iter=10).fit(trainingSet, trainLabels)
188     test_accuracy = classifier.score(testSet, testLabels) * 100
189     print('正确率:%f%%' % test_accuracy)
190
191 if __name__ == '__main__':
192     colicSklearn()

```

D:\Python3.4.4-32bit\lib\site-packages\sklearn\linear_model\sag.py:286: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
"the coef_ did not converge", ConvergenceWarning)

正确率:76.119403%
[Finished in 1.2s]

显而易见，警告是因为算法还没有收敛。更改max_iter=5000，再运行代码：

```

185     testSet.append(lineArr)
186     testLabels.append(float(currLine[-1]))
187     classifier = LogisticRegression(solver='sag', max_iter=5000).fit(trainingSet, trainLabels)
188     test_accuracy = classifier.score(testSet, testLabels) * 100
189     print('正确率:%f%%' % test_accuracy)
190
191 if __name__ == '__main__':
192     colicSklearn()

```

正确率:73.134328%
[Finished in 1.4s]

可以看到，对于我们这样的小数据集，sag算法需要迭代上千次才收敛，而liblinear只需要不到10次。

还是那句话，我们需要根据数据集情况，选择最优优化算法。

五、总结

1、Logistic回归的优缺点

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。

2、其他


- Logistic回归的目的是寻找一个非线性函数Sigmoid的最佳拟合参数，求解过程可以由最优优化算法完成。
- 改进的一些最优优化算法，比如sag。它可以在新数据到来时就完成参数更新，而不需要重新读取整个数据集来进行批量处理。
- 机器学习的一个重要问题就是如何处理缺失数据。这个问题没有标准答案，取决于实际应用中的需求。现有一些解决方案，每种方案都各有优缺点。
- 我们需要根据数据的情况，这是Sklearn的参数，以期达到更好的分类效果。
- 下篇文章将讲解支持向量机SVM。
- 如有问题，请留言。如有错误，还望指正，谢谢！

PS：如果觉得本篇本章对您有所帮助，欢迎关注、评论、赞！

本文出现的所有代码和数据集，均可在我的github上下载，欢迎Follow、Star：[点击查看](#)

参考文献：

- 《机器学习实战》的第五章内容。



微信公众号

分享技术，乐享生活：微信公众号搜索「JackCui-AI」关注一个在互联网摸爬滚打的潜行者。

黑夜无论怎样悠长，白昼总会到来。--- 莎士比亚