[用Python开始机器学习（1：配置windows平台）](http://blog.csdn.net/lsldd/article/details/41084043)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-13 19:07 1408人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41084043#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41084043#report)

使用机器学习的开发工具很多，如Matlab，R语言，Python等等。

本系列文章不会涉及深入的机器学习原理，旨在让你迅速上手，入门Python进行机器学习。

本文提供一系列资源，教你打造一个Python机器学习的平台。

1、下载资源

Python

本文以Python 3.4为例。当然你可以使用老版本。

老版本的一个优势是扩展库比较多。

链接：<https://www.python.org/>

下面是Python的扩展库。扩展库必须与使用的Python版本相对应。

这里给出一个神地址，能快速下载到下面需要的，各种平台、各种python版本的扩展库。

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

**numpy**：用于处理大规模多维数据。

**scipy**：数学库。

**scikit-learn**：机器学习库。

**matplotlib**：可视化数据神器。文档和示例地址：<http://matplotlib.org/gallery.html>

上面的安装文件如果是exe，会自动帮你拷贝到python目录。否则你需要自己拷贝到python3.4.2\Lib\site-packages下。

2、测试库

在python中使用类似“import matplotlib”的方式来查看库的安装是否正确。提示缺少什么库，就去上面的神地址里搜索，下载安装。

这几个库装完后，可能会有部分依赖库需要单独安装：

**six**、**nose**、**pyparsing**、**dateutil**。

如果出现类似six版本低于1.3之类的提示，你可能需要找到site-packages下的six.py拷贝到site-packages目录下替换老的。

3、其他资源推荐

Python教程：<http://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000>

sci-learn教程：<http://scikit-learn.org/stable/>

windows下Python的IDE选择：<http://blog.csdn.net/cserchen/article/details/7036435>

经典Python机器学习入门书籍《机器学习系统设计》的源码和数据：<https://github.com/luispedro/BuildingMachineLearningSystemsWithPython/tree/master/ch02>

# [用Python开始机器学习（2：决策树分类算法）](http://blog.csdn.net/lsldd/article/details/41223147)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-18 01:05 3001人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41223147#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41223147#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41223147)

从这一章开始进入正式的算法学习。

首先我们学习经典而有效的分类算法：**决策树分类算法**。

## 1、决策树算法

决策树用树形结构对样本的属性进行分类，是最直观的分类算法，而且也可以用于回归。不过对于一些特殊的逻辑分类会有困难。典型的如异或（XOR）逻辑，决策树并不擅长解决此类问题。

决策树的构建不是唯一的，遗憾的是最优决策树的构建属于NP问题。因此如何构建一棵好的决策树是研究的重点。

J. Ross Quinlan在1975提出将信息熵的概念引入决策树的构建，这就是鼎鼎大名的ID3算法。后续的C4.5, C5.0, CART等都是该方法的改进。

熵就是“无序，混乱”的程度。刚接触这个概念可能会有些迷惑。想快速了解如何用信息熵增益划分属性，可以参考这位兄弟的文章：<http://blog.csdn.net/alvine008/article/details/37760639>

如果还不理解，请看下面这个例子。

假设要构建这么一个自动选好苹果的决策树，简单起见，我只让他学习下面这4个样本：

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41223147)[在CODE上查看代码片](https://code.csdn.net/snippets/523477)

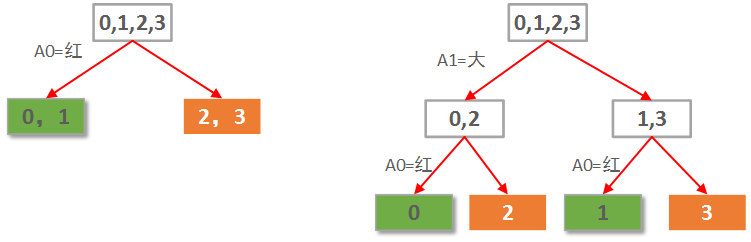
1. 样本    红     大      好苹果
2. 0       1      1         1
3. 1       1      0         1
4. 2       0      1         0
5. 3       0      0         0

样本中有2个属性，A0表示是否红苹果。A1表示是否大苹果。

那么这个样本在分类前的信息熵就是S = -(1/2 \* log(1/2) + 1/2 \* log(1/2)) = 1。

信息熵为1表示当前处于最混乱，最无序的状态。

本例仅2个属性。那么很自然一共就只可能有2棵决策树，如下图所示：



显然左边先使用A0（红色）做划分依据的决策树要优于右边用A1（大小）做划分依据的决策树。

当然这是直觉的认知。定量的考察，则需要计算每种划分情况的信息熵增益。

先选A0作划分，各子节点信息熵计算如下：

0，1叶子节点有2个正例，0个负例。信息熵为：e1 = -(2/2 \* log(2/2) + 0/2 \* log(0/2)) = 0。

2，3叶子节点有0个正例，2个负例。信息熵为：e2 = -(0/2 \* log(0/2) + 2/2 \* log(2/2)) = 0。

因此选择A0划分后的信息熵为每个子节点的信息熵所占比重的加权和：E = e1\*2/4 + e2\*2/4 = 0。

选择A0做划分的信息熵增益G（S, A0）=S - E = 1 - 0 = 1.

事实上，决策树叶子节点表示已经都属于相同类别，因此信息熵一定为0。

同样的，如果先选A1作划分，各子节点信息熵计算如下：

0，2子节点有1个正例，1个负例。信息熵为：e1 = -(1/2 \* log(1/2) + 1/2 \* log(1/2)) = 1。

1，3子节点有1个正例，1个负例。信息熵为：e2 = -(1/2 \* log(1/2) + 1/2 \* log(1/2)) = 1。

因此选择A1划分后的信息熵为每个子节点的信息熵所占比重的加权和：E = e1\*2/4 + e2\*2/4 = 1。也就是说分了跟没分一样！

选择A1做划分的信息熵增益G（S, A1）=S - E = 1 - 1 = 0.

因此，每次划分之前，我们只需要计算出信息熵增益最大的那种划分即可。

## 2、数据集

为方便讲解与理解，我们使用如下一个极其简单的测试数据集：

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41223147)[在CODE上查看代码片](https://code.csdn.net/snippets/523477)

1. 1.5 50 thin
2. 1.5 60 fat
3. 1.6 40 thin
4. 1.6 60 fat
5. 1.7 60 thin
6. 1.7 80 fat
7. 1.8 60 thin
8. 1.8 90 fat
9. 1.9 70 thin
10. 1.9 80 fat

这个数据一共有10个样本，每个样本有2个属性，分别为身高和体重，第三列为类别标签，表示“胖”或“瘦”。该数据保存在1.txt中。

我们的任务就是训练一个决策树分类器，输入身高和体重，分类器能给出这个人是胖子还是瘦子。

（数据是作者主观臆断，具有一定逻辑性，但请无视其合理性）

决策树对于“是非”的二值逻辑的分枝相当自然。而在本数据集中，身高与体重是连续值怎么办呢？

虽然麻烦一点，不过这也不是问题，只需要找到将这些连续值划分为不同区间的中间点，就转换成了二值逻辑问题。

本例决策树的任务是找到身高、体重中的一些临界值，按照大于或者小于这些临界值的逻辑将其样本两两分类，自顶向下构建决策树。

使用python的机器学习库，实现起来相当简单和优雅。

## 3、Python实现

Python代码实现如下：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41223147)[在CODE上查看代码片](https://code.csdn.net/snippets/523477)

1. # -\*- coding: utf-8 -\*-
2. **import** numpy as np
3. **import** scipy as sp
4. **from** sklearn **import** tree
5. **from** sklearn.metrics **import** precision\_recall\_curve
6. **from** sklearn.metrics **import** classification\_report
7. **from** sklearn.cross\_validation **import** train\_test\_split

10. ''''' 数据读入 '''
11. data   = []
12. labels = []
13. with open("data\\1.txt") as ifile:
14. **for** line **in** ifile:
15. tokens = line.strip().split(' ')
16. data.append([float(tk) **for** tk **in** tokens[:-1]])
17. labels.append(tokens[-1])
18. x = np.array(data)
19. labels = np.array(labels)
20. y = np.zeros(labels.shape)

23. ''''' 标签转换为0/1 '''
24. y[labels=='fat']=1
26. ''''' 拆分训练数据与测试数据 '''
27. x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2)
29. ''''' 使用信息熵作为划分标准，对决策树进行训练 '''
30. clf = tree.DecisionTreeClassifier(criterion='entropy')
31. **print**(clf)
32. clf.fit(x\_train, y\_train)
34. ''''' 把决策树结构写入文件 '''
35. with open("tree.dot", 'w') as f:
36. f = tree.export\_graphviz(clf, out\_file=f)
38. ''''' 系数反映每个特征的影响力。越大表示该特征在分类中起到的作用越大 '''
39. **print**(clf.feature\_importances\_)
41. '''''测试结果的打印'''
42. answer = clf.predict(x\_train)
43. **print**(x\_train)
44. **print**(answer)
45. **print**(y\_train)
46. **print**(np.mean( answer == y\_train))
48. '''''准确率与召回率'''
49. precision, recall, thresholds = precision\_recall\_curve(y\_train, clf.predict(x\_train))
50. answer = clf.predict\_proba(x)[:,1]
51. **print**(classification\_report(y, answer, target\_names = ['thin', 'fat']))

输出结果类似如下所示：

[ 0.2488562  0.7511438]  
array([[  1.6,  60. ],  
       [  1.7,  60. ],  
       [  1.9,  80. ],  
       [  1.5,  50. ],  
       [  1.6,  40. ],  
       [  1.7,  80. ],  
       [  1.8,  90. ],  
       [  1.5,  60. ]])  
array([ 1.,  0.,  1.,  0.,  0.,  1.,  1.,  1.])  
array([ 1.,  0.,  1.,  0.,  0.,  1.,  1.,  1.])  
1.0

             precision    recall  f1-score   support  
       thin       0.83      1.00      0.91         5  
        fat        1.00      0.80      0.89         5

avg / total       1.00      1.00      1.00         8

array([ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.])  
array([ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.])

可以看到，对训练过的数据做测试，准确率是100%。但是最后将所有数据进行测试，会出现1个测试样本分类错误。

说明本例的决策树对训练集的规则吸收的很好，但是预测性稍微差点。

这里有3点需要说明，这在以后的机器学习中都会用到。

1、**拆分训练数据与测试数据**。

这样做是为了方便做交叉检验。交叉检验是为了充分测试分类器的稳定性。

代码中的0.2表示随机取20%的数据作为测试用。其余80%用于训练决策树。

也就是说10个样本中随机取8个训练。本文数据集小，这里的目的是可以看到由于取的训练数据随机，每次构建的决策树都不一样。

2、**特征的不同影响因子。**

样本的不同特征对分类的影响权重差异会很大。分类结束后看看每个样本对分类的影响度也是很重要的。

在本例中，身高的权重为0.25，体重为0.75，可以看到重量的重要性远远高于身高。对于胖瘦的判定而言，这也是相当符合逻辑的。

3、**准确率与召回率**。

这2个值是评判分类准确率的一个重要标准。比如代码的最后将所有10个样本输入分类器进行测试的结果：

测试结果：array([ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.])  
真实结果：array([ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.])

分为thin的准确率为0.83。是因为分类器分出了6个thin，其中正确的有5个，因此分为thin的准确率为5/6=0.83。

分为thin的召回率为1.00。是因为数据集中共有5个thin，而分类器把他们都分对了（虽然把一个fat分成了thin！），召回率5/5=1。

分为fat的准确率为1.00。不再赘述。

分为fat的召回率为0.80。是因为数据集中共有5个fat，而分类器只分出了4个（把一个fat分成了thin！），召回率4/5=0.80。

很多时候，尤其是数据分类难度较大的情况，准确率与召回率往往是矛盾的。你可能需要根据你的需要找到最佳的一个平衡点。

比如本例中，你的目标是尽可能保证找出来的胖子是真胖子（准确率），还是保证尽可能找到更多的胖子（召回率）。

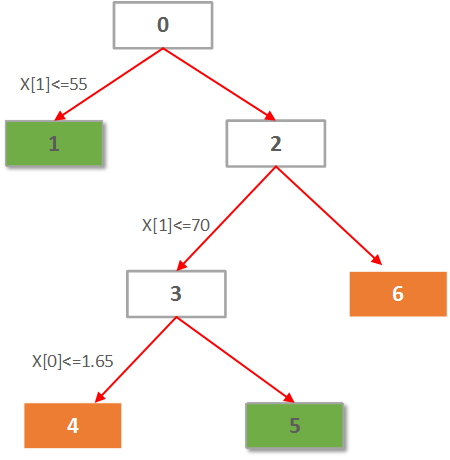
代码还把决策树的结构写入了tree.dot中。打开该文件，很容易画出决策树，还可以看到决策树的更多分类信息。

本文的tree.dot如下所示：

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41223147)[在CODE上查看代码片](https://code.csdn.net/snippets/523477)

1. digraph Tree {
2. 0 [label="X[1] <= 55.0000\nentropy = 0.954434002925\nsamples = 8", shape="box"] ;
3. 1 [label="entropy = 0.0000\nsamples = 2\nvalue = [ 2.  0.]", shape="box"] ;
4. 0 -> 1 ;
5. 2 [label="X[1] <= 70.0000\nentropy = 0.650022421648\nsamples = 6", shape="box"] ;
6. 0 -> 2 ;
7. 3 [label="X[0] <= 1.6500\nentropy = 0.918295834054\nsamples = 3", shape="box"] ;
8. 2 -> 3 ;
9. 4 [label="entropy = 0.0000\nsamples = 2\nvalue = [ 0.  2.]", shape="box"] ;
10. 3 -> 4 ;
11. 5 [label="entropy = 0.0000\nsamples = 1\nvalue = [ 1.  0.]", shape="box"] ;
12. 3 -> 5 ;
13. 6 [label="entropy = 0.0000\nsamples = 3\nvalue = [ 0.  3.]", shape="box"] ;
14. 2 -> 6 ;
15. }

根据这个信息，决策树应该长的如下这个样子：



[用Python开始机器学习（3：数据拟合与广义线性回归）](http://blog.csdn.net/lsldd/article/details/41251583)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-19 01:55 4025人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41251583#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41251583#report)

机器学习中的预测问题通常分为2类：**回归**与**分类**。

简单的说回归就是预测数值，而分类是给数据打上标签归类。

本文讲述如何用Python进行基本的数据拟合，以及如何对拟合结果的误差进行分析。

本例中使用一个2次函数加上随机的扰动来生成500个点，然后尝试用1、2、100次方的多项式对该数据进行拟合。  
拟合的目的是使得根据训练数据能够拟合出一个多项式函数，这个函数能够很好的拟合现有数据，并且能对未知的数据进行预测。

代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41251583)[在CODE上查看代码片](https://code.csdn.net/snippets/524870)

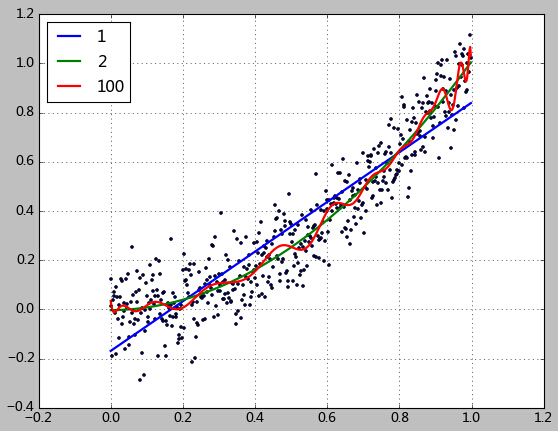
1. **import** matplotlib.pyplot as plt
2. **import** numpy as np
3. **import** scipy as sp
4. **from** scipy.stats **import** norm
5. **from** sklearn.pipeline **import** Pipeline
6. **from** sklearn.linear\_model **import** LinearRegression
7. **from** sklearn.preprocessing **import** PolynomialFeatures
8. **from** sklearn **import** linear\_model
10. ''''' 数据生成 '''
11. x = np.arange(0, 1, 0.002)
12. y = norm.rvs(0, size=500, scale=0.1)
13. y = y + x\*\*2
15. ''''' 均方误差根 '''
16. **def** rmse(y\_test, y):
17. **return** sp.sqrt(sp.mean((y\_test - y) \*\* 2))
19. ''''' 与均值相比的优秀程度，介于[0~1]。0表示不如均值。1表示完美预测.这个版本的实现是参考scikit-learn官网文档  '''
20. **def** R2(y\_test, y\_true):
21. **return** 1 - ((y\_test - y\_true)\*\*2).sum() / ((y\_true - y\_true.mean())\*\*2).sum()

24. ''''' 这是Conway&White《机器学习使用案例解析》里的版本 '''
25. **def** R22(y\_test, y\_true):
26. y\_mean = np.array(y\_true)
27. y\_mean[:] = y\_mean.mean()
28. **return** 1 - rmse(y\_test, y\_true) / rmse(y\_mean, y\_true)

31. plt.scatter(x, y, s=5)
32. degree = [1,2,100]
33. y\_test = []
34. y\_test = np.array(y\_test)

37. **for** d **in** degree:
38. clf = Pipeline([('poly', PolynomialFeatures(degree=d)),
39. ('linear', LinearRegression(fit\_intercept=False))])
40. clf.fit(x[:, np.newaxis], y)
41. y\_test = clf.predict(x[:, np.newaxis])
43. **print**(clf.named\_steps['linear'].coef\_)
44. **print**('rmse=%.2f, R2=%.2f, R22=%.2f, clf.score=%.2f' %
45. (rmse(y\_test, y),
46. R2(y\_test, y),
47. R22(y\_test, y),
48. clf.score(x[:, np.newaxis], y)))
50. plt.plot(x, y\_test, linewidth=2)
52. plt.grid()
53. plt.legend(['1','2','100'], loc='upper left')
54. plt.show()

该程序运行的显示结果如下：



[-0.16140183  0.99268453]  
rmse=0.13, R2=0.82, R22=0.58, clf.score=0.82

[ 0.00934527 -0.03591245  1.03065829]  
rmse=0.11, R2=0.88, R22=0.66, clf.score=0.88  
[  6.07130354e-02  -1.02247150e+00   6.66972089e+01  -1.85696012e+04  
......

-9.43408707e+12  -9.78954604e+12  -9.99872105e+12  -1.00742526e+13  
-1.00303296e+13  -9.88198843e+12  -9.64452002e+12  -9.33298267e+12  
  -1.00580760e+12]

rmse=0.10, R2=0.89, R22=0.67, clf.score=0.89

显示出的coef\_就是多项式参数。如1次拟合的结果为

y = 0.99268453x -0.16140183

这里我们要注意这几点：

1、**误差分析**。

做回归分析，常用的误差主要有均方误差根（RMSE）和R-平方（R2）。

**RMSE**是预测值与真实值的误差平方根的均值。这种度量方法很流行（Netflix机器学习比赛的评价方法），是一种定量的权衡方法。

**R2**方法是将预测值跟只使用均值的情况下相比，看能好多少。其区间通常在（0,1）之间。0表示还不如什么都不预测，直接取均值的情况，而1表示所有预测跟真实结果完美匹配的情况。

R2的计算方法，不同的文献稍微有不同。如本文中函数R2是依据scikit-learn官网文档实现的，跟clf.score函数结果一致。

而R22函数的实现来自Conway的著作《机器学习使用案例解析》，不同在于他用的是2个RMSE的比值来计算R2。

我们看到多项式次数为1的时候，虽然拟合的不太好，R2也能达到0.82。2次多项式提高到了0.88。而次数提高到100次，R2也只提高到了0.89。

2、**过拟合**。

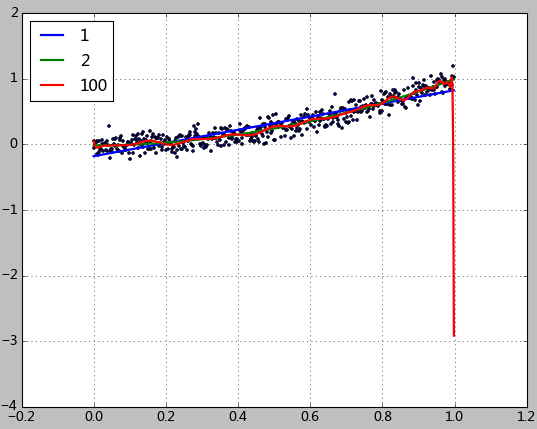
使用100次方多项式做拟合，效果确实是高了一些，然而该模型的据测能力却极其差劲。

而且注意看多项式系数，出现了大量的大数值，甚至达到10的12次方。

这里我们修改代码，将500个样本中的最后2个从训练集中移除。然而在测试中却仍然测试所有500个样本。

clf.fit(x[:**498**, np.newaxis], y[:**498**])

这样修改后的多项式拟合结果如下：



[-0.17933531  1.0052037 ]  
rmse=0.12, R2=0.85, R22=0.61, clf.score=0.85  
[-0.01631935  0.01922011  0.99193521]  
rmse=0.10, R2=0.90, R22=0.69, clf.score=0.90

...

rmse=0.21, R2=0.57, R22=0.34, clf.score=**0.57**

仅仅只是缺少了最后2个训练样本，红线（100次方多项式拟合结果）的预测发生了剧烈的偏差，R2也急剧下降到0.57。

而反观1，2次多项式的拟合结果，R2反而略微上升了。

这说明高次多项式过度拟合了训练数据，包括其中大量的噪音，导致其完全丧失了对数据趋势的预测能力。前面也看到，100次多项式拟合出的系数数值无比巨大。人们自然想到通过在拟合过程中限制这些系数数值的大小来避免生成这种畸形的拟合函数。

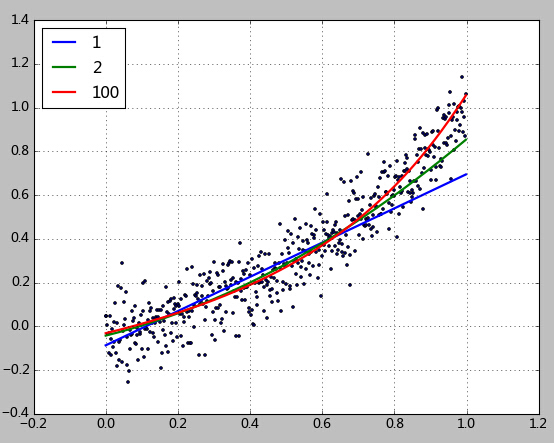
其基本原理是将拟合多项式的所有系数绝对值之和（L1正则化）或者平方和（L2正则化）加入到惩罚模型中，并指定一个惩罚力度因子w，来避免产生这种畸形系数。

这样的思想应用在了岭（Ridge）回归（使用L2正则化）、Lasso法（使用L1正则化）、弹性网（Elastic net，使用L1+L2正则化）等方法中，都能有效避免过拟合。更多原理可以参考相关资料。

下面以岭回归为例看看100次多项式的拟合是否有效。将代码修改如下:

clf = Pipeline([('poly', PolynomialFeatures(degree=d)),  
                    ('linear', linear\_model.**Ridge** ())])  
clf.fit(x[:**400**, np.newaxis], y[:**400**])

结果如下：



[ 0.          0.75873781]  
rmse=0.15, R2=0.78, R22=0.53, clf.score=0.78  
[ 0.          0.35936882  0.52392172]  
rmse=0.11, R2=0.87, R22=0.64, clf.score=0.87  
[  0.00000000e+00   2.63903249e-01   3.14973328e-01   2.43389461e-01  
   1.67075328e-01   1.10674280e-01   7.30672237e-02   4.88605804e-02  
   ......  
   3.70018540e-11   2.93631291e-11   2.32992690e-11   1.84860002e-11  
   1.46657377e-11]  
rmse=0.10, R2=0.90, R22=0.68, clf.score=0.90

可以看到，100次多项式的系数参数变得很小。大部分都接近于0.

另外值得注意的是，使用岭回归之类的惩罚模型后，1次和2次多项式回归的R2值可能会稍微低于基本线性回归。

然而这样的模型，即使使用100次多项式，在训练400个样本，预测500个样本的情况下不仅有更小的R2误差，而且还具备优秀的预测能力。

[用Python开始机器学习（3：数据拟合与广义线性回归）](http://blog.csdn.net/lsldd/article/details/41251583)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-19 01:55 4026人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41251583#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41251583#report)

机器学习中的预测问题通常分为2类：**回归**与**分类**。

简单的说回归就是预测数值，而分类是给数据打上标签归类。

本文讲述如何用Python进行基本的数据拟合，以及如何对拟合结果的误差进行分析。

本例中使用一个2次函数加上随机的扰动来生成500个点，然后尝试用1、2、100次方的多项式对该数据进行拟合。  
拟合的目的是使得根据训练数据能够拟合出一个多项式函数，这个函数能够很好的拟合现有数据，并且能对未知的数据进行预测。

代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41251583)[在CODE上查看代码片](https://code.csdn.net/snippets/524870)

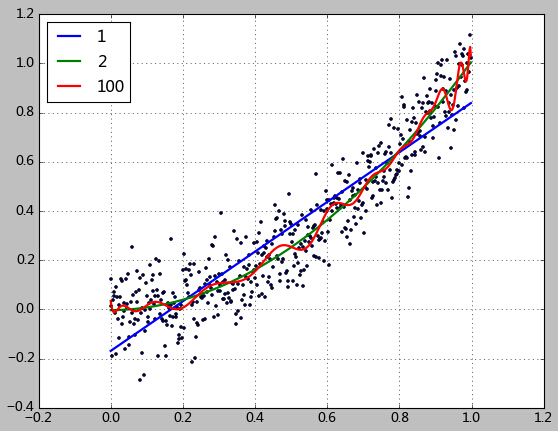
1. **import** matplotlib.pyplot as plt
2. **import** numpy as np
3. **import** scipy as sp
4. **from** scipy.stats **import** norm
5. **from** sklearn.pipeline **import** Pipeline
6. **from** sklearn.linear\_model **import** LinearRegression
7. **from** sklearn.preprocessing **import** PolynomialFeatures
8. **from** sklearn **import** linear\_model
10. ''''' 数据生成 '''
11. x = np.arange(0, 1, 0.002)
12. y = norm.rvs(0, size=500, scale=0.1)
13. y = y + x\*\*2
15. ''''' 均方误差根 '''
16. **def** rmse(y\_test, y):
17. **return** sp.sqrt(sp.mean((y\_test - y) \*\* 2))
19. ''''' 与均值相比的优秀程度，介于[0~1]。0表示不如均值。1表示完美预测.这个版本的实现是参考scikit-learn官网文档  '''
20. **def** R2(y\_test, y\_true):
21. **return** 1 - ((y\_test - y\_true)\*\*2).sum() / ((y\_true - y\_true.mean())\*\*2).sum()

24. ''''' 这是Conway&White《机器学习使用案例解析》里的版本 '''
25. **def** R22(y\_test, y\_true):
26. y\_mean = np.array(y\_true)
27. y\_mean[:] = y\_mean.mean()
28. **return** 1 - rmse(y\_test, y\_true) / rmse(y\_mean, y\_true)

31. plt.scatter(x, y, s=5)
32. degree = [1,2,100]
33. y\_test = []
34. y\_test = np.array(y\_test)

37. **for** d **in** degree:
38. clf = Pipeline([('poly', PolynomialFeatures(degree=d)),
39. ('linear', LinearRegression(fit\_intercept=False))])
40. clf.fit(x[:, np.newaxis], y)
41. y\_test = clf.predict(x[:, np.newaxis])
43. **print**(clf.named\_steps['linear'].coef\_)
44. **print**('rmse=%.2f, R2=%.2f, R22=%.2f, clf.score=%.2f' %
45. (rmse(y\_test, y),
46. R2(y\_test, y),
47. R22(y\_test, y),
48. clf.score(x[:, np.newaxis], y)))
50. plt.plot(x, y\_test, linewidth=2)
52. plt.grid()
53. plt.legend(['1','2','100'], loc='upper left')
54. plt.show()

该程序运行的显示结果如下：



[-0.16140183  0.99268453]  
rmse=0.13, R2=0.82, R22=0.58, clf.score=0.82

[ 0.00934527 -0.03591245  1.03065829]  
rmse=0.11, R2=0.88, R22=0.66, clf.score=0.88  
[  6.07130354e-02  -1.02247150e+00   6.66972089e+01  -1.85696012e+04  
......

-9.43408707e+12  -9.78954604e+12  -9.99872105e+12  -1.00742526e+13  
-1.00303296e+13  -9.88198843e+12  -9.64452002e+12  -9.33298267e+12  
  -1.00580760e+12]

rmse=0.10, R2=0.89, R22=0.67, clf.score=0.89

显示出的coef\_就是多项式参数。如1次拟合的结果为

y = 0.99268453x -0.16140183

这里我们要注意这几点：

1、**误差分析**。

做回归分析，常用的误差主要有均方误差根（RMSE）和R-平方（R2）。

**RMSE**是预测值与真实值的误差平方根的均值。这种度量方法很流行（Netflix机器学习比赛的评价方法），是一种定量的权衡方法。

**R2**方法是将预测值跟只使用均值的情况下相比，看能好多少。其区间通常在（0,1）之间。0表示还不如什么都不预测，直接取均值的情况，而1表示所有预测跟真实结果完美匹配的情况。

R2的计算方法，不同的文献稍微有不同。如本文中函数R2是依据scikit-learn官网文档实现的，跟clf.score函数结果一致。

而R22函数的实现来自Conway的著作《机器学习使用案例解析》，不同在于他用的是2个RMSE的比值来计算R2。

我们看到多项式次数为1的时候，虽然拟合的不太好，R2也能达到0.82。2次多项式提高到了0.88。而次数提高到100次，R2也只提高到了0.89。

2、**过拟合**。

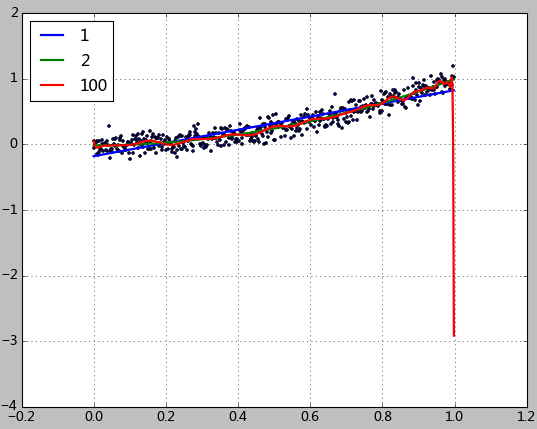
使用100次方多项式做拟合，效果确实是高了一些，然而该模型的据测能力却极其差劲。

而且注意看多项式系数，出现了大量的大数值，甚至达到10的12次方。

这里我们修改代码，将500个样本中的最后2个从训练集中移除。然而在测试中却仍然测试所有500个样本。

clf.fit(x[:**498**, np.newaxis], y[:**498**])

这样修改后的多项式拟合结果如下：



[-0.17933531  1.0052037 ]  
rmse=0.12, R2=0.85, R22=0.61, clf.score=0.85  
[-0.01631935  0.01922011  0.99193521]  
rmse=0.10, R2=0.90, R22=0.69, clf.score=0.90

...

rmse=0.21, R2=0.57, R22=0.34, clf.score=**0.57**

仅仅只是缺少了最后2个训练样本，红线（100次方多项式拟合结果）的预测发生了剧烈的偏差，R2也急剧下降到0.57。

而反观1，2次多项式的拟合结果，R2反而略微上升了。

这说明高次多项式过度拟合了训练数据，包括其中大量的噪音，导致其完全丧失了对数据趋势的预测能力。前面也看到，100次多项式拟合出的系数数值无比巨大。人们自然想到通过在拟合过程中限制这些系数数值的大小来避免生成这种畸形的拟合函数。

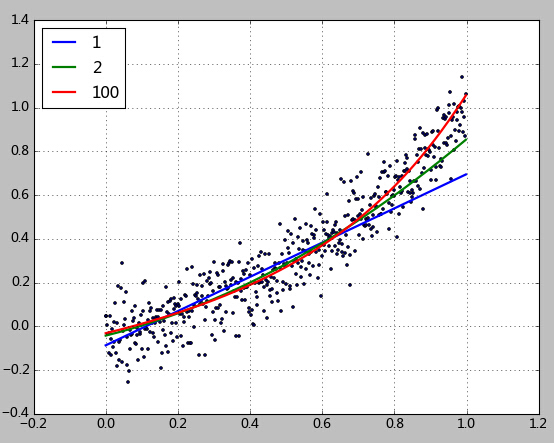
其基本原理是将拟合多项式的所有系数绝对值之和（L1正则化）或者平方和（L2正则化）加入到惩罚模型中，并指定一个惩罚力度因子w，来避免产生这种畸形系数。

这样的思想应用在了岭（Ridge）回归（使用L2正则化）、Lasso法（使用L1正则化）、弹性网（Elastic net，使用L1+L2正则化）等方法中，都能有效避免过拟合。更多原理可以参考相关资料。

下面以岭回归为例看看100次多项式的拟合是否有效。将代码修改如下:

clf = Pipeline([('poly', PolynomialFeatures(degree=d)),  
                    ('linear', linear\_model.**Ridge** ())])  
clf.fit(x[:**400**, np.newaxis], y[:**400**])

结果如下：



[ 0.          0.75873781]  
rmse=0.15, R2=0.78, R22=0.53, clf.score=0.78  
[ 0.          0.35936882  0.52392172]  
rmse=0.11, R2=0.87, R22=0.64, clf.score=0.87  
[  0.00000000e+00   2.63903249e-01   3.14973328e-01   2.43389461e-01  
   1.67075328e-01   1.10674280e-01   7.30672237e-02   4.88605804e-02  
   ......  
   3.70018540e-11   2.93631291e-11   2.32992690e-11   1.84860002e-11  
   1.46657377e-11]  
rmse=0.10, R2=0.90, R22=0.68, clf.score=0.90

可以看到，100次多项式的系数参数变得很小。大部分都接近于0.

另外值得注意的是，使用岭回归之类的惩罚模型后，1次和2次多项式回归的R2值可能会稍微低于基本线性回归。

然而这样的模型，即使使用100次多项式，在训练400个样本，预测500个样本的情况下不仅有更小的R2误差，而且还具备优秀的预测能力。

# [用Python开始机器学习（4：KNN分类算法）](http://blog.csdn.net/lsldd/article/details/41357931)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-23 17:24 2563人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41357931#comments)(5) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41357931#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41357931)

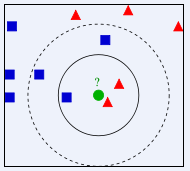
## 1、KNN分类算法

KNN分类算法（K-Nearest-Neighbors Classification），又叫K近邻算法，是一个概念极其简单，而分类效果又很优秀的分类算法。

他的核心思想就是，要确定测试样本属于哪一类，就寻找所有训练样本中与该测试样本“距离”最近的前K个样本，然后看这K个样本大部分属于哪一类，那么就认为这个测试样本也属于哪一类。简单的说就是让最相似的K个样本来投票决定。

这里所说的距离，一般最常用的就是多维空间的**欧式距离**。这里的维度指**特征维度**，即样本有几个特征就属于几维。

KNN示意图如下所示。（图片来源：百度百科<http://baike.baidu.com/view/1485833.htm?from_id=3479559&type=syn&fromtitle=knn&fr=aladdin>）



上图中要确定测试样本绿色属于蓝色还是红色。

显然，当K=3时，将以1：2的投票结果分类于红色；而K=5时，将以3：2的投票结果分类于蓝色。

KNN算法简单有效，但没有优化的暴力法效率容易达到瓶颈。如样本个数为N，特征维度为D的时候，该算法时间复杂度呈O（DN)增长。

所以通常KNN的实现会把训练数据构建成K-D Tree（K-dimensional tree），构建过程很快，甚至不用计算D维欧氏距离，而搜索速度高达O（D\*log（N））。

不过当D维度过高，会产生所谓的”维度灾难“，最终效率会降低到与暴力法一样。

因此通常D>20以后，最好使用更高效率的Ball-Tree，其时间复杂度为O（D\*log（N））。

人们经过长期的实践发现KNN算法虽然简单，但能处理大规模的数据分类，尤其适用于样本分类边界不规则的情况。最重要的是该算法是很多高级机器学习算法的基础。

当然，KNN算法也存在一切问题。比如如果训练数据大部分都属于某一类，投票算法就有很大问题了。这时候就需要考虑设计每个投票者票的权重了。

## 2、测试数据

测试数据的格式仍然和前面使用的身高体重数据一致。不过数据增加了一些：

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41357931)[在CODE上查看代码片](https://code.csdn.net/snippets/529730)

1. 1.5 40 thin
2. 1.5 50 fat
3. 1.5 60 fat
4. 1.6 40 thin
5. 1.6 50 thin
6. 1.6 60 fat
7. 1.6 70 fat
8. 1.7 50 thin
9. 1.7 60 thin
10. 1.7 70 fat
11. 1.7 80 fat
12. 1.8 60 thin
13. 1.8 70 thin
14. 1.8 80 fat
15. 1.8 90 fat
16. 1.9 80 thin
17. 1.9 90 fat

## 3、Python代码

scikit-learn提供了优秀的KNN算法支持。使用Python代码如下：

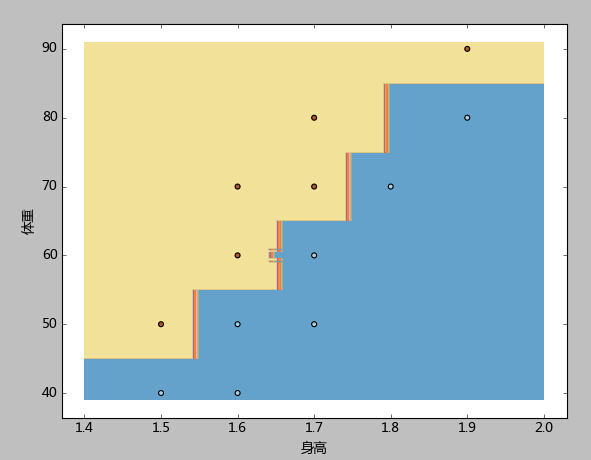
**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41357931)[在CODE上查看代码片](https://code.csdn.net/snippets/529730)

1. # -\*- coding: utf-8 -\*-
2. **import** numpy as np
3. **from** sklearn **import** neighbors
4. **from** sklearn.metrics **import** precision\_recall\_curve
5. **from** sklearn.metrics **import** classification\_report
6. **from** sklearn.cross\_validation **import** train\_test\_split
7. **import** matplotlib.pyplot as plt
9. ''''' 数据读入 '''
10. data   = []
11. labels = []
12. with open("data\\1.txt") as ifile:
13. **for** line **in** ifile:
14. tokens = line.strip().split(' ')
15. data.append([float(tk) **for** tk **in** tokens[:-1]])
16. labels.append(tokens[-1])
17. x = np.array(data)
18. labels = np.array(labels)
19. y = np.zeros(labels.shape)
21. ''''' 标签转换为0/1 '''
22. y[labels=='fat']=1
24. ''''' 拆分训练数据与测试数据 '''
25. x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2)
27. ''''' 创建网格以方便绘制 '''
28. h = .01
29. x\_min, x\_max = x[:, 0].min() - 0.1, x[:, 0].max() + 0.1
30. y\_min, y\_max = x[:, 1].min() - 1, x[:, 1].max() + 1
31. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),
32. np.arange(y\_min, y\_max, h))
34. ''''' 训练KNN分类器 '''
35. clf = neighbors.KNeighborsClassifier(algorithm='kd\_tree')
36. clf.fit(x\_train, y\_train)
38. '''''测试结果的打印'''
39. answer = clf.predict(x)
40. **print**(x)
41. **print**(answer)
42. **print**(y)
43. **print**(np.mean( answer == y))
45. '''''准确率与召回率'''
46. precision, recall, thresholds = precision\_recall\_curve(y\_train, clf.predict(x\_train))
47. answer = clf.predict\_proba(x)[:,1]
48. **print**(classification\_report(y, answer, target\_names = ['thin', 'fat']))
50. ''''' 将整个测试空间的分类结果用不同颜色区分开'''
51. answer = clf.predict\_proba(np.c\_[xx.ravel(), yy.ravel()])[:,1]
52. z = answer.reshape(xx.shape)
53. plt.contourf(xx, yy, z, cmap=plt.cm.Paired, alpha=0.8)
55. ''''' 绘制训练样本 '''
56. plt.scatter(x\_train[:, 0], x\_train[:, 1], c=y\_train, cmap=plt.cm.Paired)
57. plt.xlabel(u'身高')
58. plt.ylabel(u'体重')
59. plt.show()

## 4、结果分析

其输出结果如下：

[ 0.  0.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  1.]  
[ 0.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  1.]  
准确率=0.94, score=0.94  
             precision    recall  f1-score   support  
       thin      0.89      1.00      0.94         8  
        fat       1.00      0.89      0.94         9  
avg / total       0.95      0.94      0.94        17



KNN分类器在众多分类算法中属于最简单的之一，需要注意的地方不多。有这几点要说明：  
1、KNeighborsClassifier可以设置3种算法：‘brute’，‘kd\_tree’，‘ball\_tree’。如果不知道用哪个好，设置‘auto’让KNeighborsClassifier自己根据输入去决定。

2、注意统计准确率时，分类器的score返回的是计算正确的比例，而不是R2。R2一般应用于回归问题。

3、本例先根据样本中身高体重的最大最小值，生成了一个密集网格（步长h=0.01），然后将网格中的每一个点都当成测试样本去测试，最后使用contourf函数，使用不同的颜色标注出了胖、廋两类。

容易看到，本例的分类边界，属于相对复杂，但却又与距离呈现明显规则的锯齿形。

这种边界线性函数是难以处理的。而KNN算法处理此类边界问题具有天生的优势。我们在后续的系列中会看到，这个数据集达到准确率=0.94算是很优秀的结果了。

# [用Python开始机器学习（5：文本特征抽取与向量化）](http://blog.csdn.net/lsldd/article/details/41520953)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-26 22:09 3169人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41520953#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41520953#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41520953)

假设我们刚看完诺兰的大片《星际穿越》，设想如何让机器来自动分析各位观众对电影的评价到底是“赞”（positive）还是“踩”（negative）呢？

这类问题就属于情感分析问题。这类问题处理的第一步，就是将文本转换为特征。

因此，这章我们只学习第一步，如何从文本中抽取特征，并将其向量化。

由于中文的处理涉及到分词问题，本文用一个简单的例子来说明如何使用Python的机器学习库，对英文进行特征提取。

## 1、数据准备

Python的sklearn.datasets支持从目录读取所有分类好的文本。不过目录必须按照一个文件夹一个标签名的规则放好。比如本文使用的数据集共有2个标签，一个为“net”，一个为“pos”，每个目录下面有6个文本文件。目录如下所示：  
neg  
    1.txt  
    2.txt  
    ......

pos  
    1.txt  
    2.txt  
    ....

12个文件的内容汇总起来如下所示：

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41520953)[在CODE上查看代码片](https://code.csdn.net/snippets/534073)

1. neg:
2. shit.
3. waste my money.
4. waste of money.
5. sb movie.
6. waste of time.
7. a shit movie.
8. pos:
9. nb! nb movie!
10. nb!
11. worth my money.
12. I love this movie!
13. a nb movie.
14. worth it!

## 2、文本特征

如何从这些英文中抽取情感态度而进行分类呢？

最直观的做法就是抽取单词。通常认为，很多关键词能够反映说话者的态度。比如上面这个简单的数据集，很容易发现，凡是说了“shit”的，就一定属于neg类。

当然，上面数据集是为了方便描述而简单设计的。现实中一个词经常会有穆棱两可的态度。但是仍然有理由相信，某个单词在neg类中出现的越多，那么他表示neg态度的概率越大。

同样我们注意到有些单词对情感分类是毫无意义的。比如上述数据中的“of”，“I”之类的单词。这类词有个名字，叫“**Stop\_Word**“（**停用词**）。这类词是可以完全忽略掉不做统计的。显然忽略掉这些词，词频记录的存储空间能够得到优化，而且构建速度也更快。

把每个单词的词频作为重要的特征也存在一个问题。比如上述数据中的”movie“，在12个样本中出现了5次，但是出现正反两边次数差不多，没有什么区分度。而”worth“出现了2次，但却只出现在pos类中，显然更具有强烈的刚晴色彩，即区分度很高。

因此，我们需要引入**TF-IDF**（Term Frequency-Inverse Document Frequency，**词频和逆向文件频率**）对每个单词做进一步考量。

**TF**（**词频**）的计算很简单，就是针对一个文件t，某个单词Nt 出现在该文档中的频率。比如文档“I love this movie”，单词“love”的TF为1/4。如果去掉停用词“I"和”it“，则为1/2。

**IDF**（**逆向文件频率**）的意义是，对于某个单词t，凡是出现了该单词的文档数Dt，占了全部测试文档D的比例，再求自然对数。

比如单词“movie“一共出现了5次，而文档总数为12，因此IDF为ln(5/12)。

很显然，IDF是为了凸显那种出现的少，但是占有强烈感情色彩的词语。比如“movie”这样的词的IDF=ln(12/5)=0.88，远小于“love”的IDF=ln(12/1)=2.48。

**TF-IDF**就是把二者简单的乘在一起即可。这样，求出每个文档中，每个单词的TF-IDF，就是我们提取得到的文本特征值。

## 3、向量化

有了上述基础，就能够将文档向量化了。我们先看代码，再来分析向量化的意义：

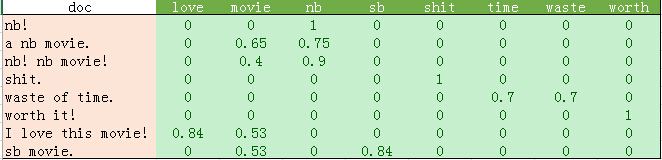
**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41520953)[在CODE上查看代码片](https://code.csdn.net/snippets/534073)

1. # -\*- coding: utf-8 -\*-
2. **import** scipy as sp
3. **import** numpy as np
4. **from** sklearn.datasets **import** load\_files
5. **from** sklearn.cross\_validation **import** train\_test\_split
6. **from** sklearn.feature\_extraction.text **import**  TfidfVectorizer
8. '''''加载数据集，切分数据集80%训练，20%测试'''
9. movie\_reviews = load\_files('endata')
10. doc\_terms\_train, doc\_terms\_test, y\_train, y\_test\
11. = train\_test\_split(movie\_reviews.data, movie\_reviews.target, test\_size = 0.3)
13. '''''BOOL型特征下的向量空间模型，注意，测试样本调用的是transform接口'''
14. count\_vec = TfidfVectorizer(binary = False, decode\_error = 'ignore',\
15. stop\_words = 'english')
16. x\_train = count\_vec.fit\_transform(doc\_terms\_train)
17. x\_test  = count\_vec.transform(doc\_terms\_test)
18. x       = count\_vec.transform(movie\_reviews.data)
19. y       = movie\_reviews.target
20. **print**(doc\_terms\_train)
21. **print**(count\_vec.get\_feature\_names())
22. **print**(x\_train.toarray())
23. **print**(movie\_reviews.target)

运行结果如下：

[b'waste of time.', b'a shit movie.', b'a nb movie.', b'I love this movie!', b'shit.', b'worth my money.', b'sb movie.', b'worth it!']  
['love', 'money', 'movie', 'nb', 'sb', 'shit', 'time', 'waste', 'worth']  
[[ 0.          0.          0.          0.          0.          0.   0.70710678  0.70710678  0.        ]  
 [ 0.          0.          0.60335753  0.          0.          0.79747081   0.          0.          0.        ]  
 [ 0.          0.          0.53550237  0.84453372  0.          0.          0.   0.          0.        ]  
 [ 0.84453372  0.          0.53550237  0.          0.          0.          0.   0.          0.        ]  
 [ 0.          0.          0.          0.          0.          1.          0.   0.          0.        ]  
 [ 0.          0.76642984  0.          0.          0.          0.          0.   0.          0.64232803]  
 [ 0.          0.          0.53550237  0.          0.84453372  0.          0.   0.          0.        ]  
 [ 0.          0.          0.          0.          0.          0.          0.   0.          1.        ]]  
[1 1 0 1 0 1 0 1 1 0 0 0]

python输出的比较混乱。我这里做了一个表格如下：



从上表可以发现如下几点：

1、停用词的过滤。

初始化count\_vec的时候，我们在count\_vec构造时传递了stop\_words = 'english'，表示使用默认的英文停用词。可以使用count\_vec.get\_stop\_words()查看TfidfVectorizer内置的所有停用词。当然，在这里可以传递你自己的停用词list（比如这里的“movie”）

2、TF-IDF的计算。

这里词频的计算使用的是sklearn的TfidfVectorizer。这个类继承于CountVectorizer，在后者基本的词频统计基础上增加了如TF-IDF之类的功能。

我们会发现这里计算的结果跟我们之前计算不太一样。因为这里count\_vec构造时默认传递了max\_df=1，因此TF-IDF都做了规格化处理，以便将所有值约束在[0,1]之间。

3、count\_vec.fit\_transform的结果是一个巨大的矩阵。我们可以看到上表中有大量的0，因此sklearn在内部实现上使用了稀疏矩阵。本例子数据较小。如果读者有兴趣，可以试试机器学习科研工作者使用的真实数据，来自康奈尔大学：<http://www.cs.cornell.edu/people/pabo/movie-review-data/>。这个网站提供了很多数据集，其中有几个2M左右的数据库，正反例700个左右。这样的数据规模也不算大，1分钟内还是可以跑完的，建议大家试一试。不过要注意这些数据集可能存在非法字符问题。所以在构造count\_vec时，传入了decode\_error = 'ignore'，以忽略这些非法字符。

上表的结果，就是训练8个样本的8个特征的一个结果。这个结果就可以使用各种分类算法进行分类了。

# [用Python开始机器学习（6：朴素贝叶斯分类器）](http://blog.csdn.net/lsldd/article/details/41542107)

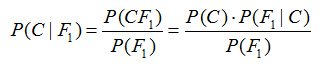
分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-27 14:52 2734人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41542107#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41542107#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41542107)

朴素贝叶斯分类器是一个以贝叶斯定理为基础，广泛应用于情感分类领域的优美分类器。本文我们尝试使用该分类器来解决上一篇文章中影评态度分类。

## 1、贝叶斯定理

假设对于某个数据集，随机变量C表示样本为C类的概率，F1表示测试样本某特征出现的概率，套用基本贝叶斯公式，则如下所示：



上式表示对于某个样本，特征F1出现时，该样本被分为C类的条件概率。那么如何用上式来对测试样本分类呢？

举例来说，有个测试样本，其特征F1出现了（F1=1），那么就计算P(C=0|F1=1)和P(C=1|F1=1)的概率值。前者大，则该样本被认为是0类；后者大，则分为1类。

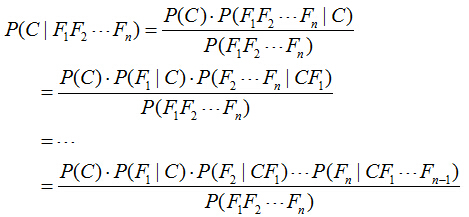
对该公示，有几个概念需要熟知：

**先验概率（Prior）**。P(C)是C的先验概率，可以从已有的训练集中计算分为C类的样本占所有样本的比重得出。

**证据（Evidence）**。即上式P(F1)，表示对于某测试样本，特征F1出现的概率。同样可以从训练集中F1特征对应样本所占总样本的比例得出。

**似然（likelihood）**。即上式P(F1|C)，表示如果知道一个样本分为C类，那么他的特征为F1的概率是多少。

对于多个特征而言，贝叶斯公式可以扩展如下：



分子中存在一大串似然值。当特征很多的时候，这些似然值的计算是极其痛苦的。现在该怎么办？

## 2、朴素的概念

为了简化计算，朴素贝叶斯算法做了一假设：“**朴素的认为各个特征相互独立**”。这么一来，上式的分子就简化成了：

P(C)\*P(F1|C)\*P(F2|C)...P(Fn|C)。

这样简化过后，计算起来就方便多了。

这个假设是认为各个特征之间是独立的，看上去确实是个很不科学的假设。因为很多情况下，各个特征之间是紧密联系的。然而在朴素贝叶斯的大量应用实践实际表明其工作的相当好。

其次，由于朴素贝叶斯的工作原理是计算P(C=0|F1...Fn)和P(C=1|F1...Fn)，并取最大值的那个作为其分类。而二者的分母是一模一样的。因此，我们又可以**省略分母计算**，从而进一步简化计算过程。

另外，贝叶斯公式推导能够成立有个重要前期，就是各个证据（evidence）不能为0。也即对于任意特征Fx，P(Fx)不能为0。而显示某些特征未出现在测试集中的情况是可以发生的。因此实现上通常要做一些小的处理，例如把所有计数进行+1（**加法平滑(additive smoothing**，又叫**拉普拉斯平滑(Laplace smothing**)）。而如果通过增加一个大于0的可调参数alpha进行平滑，就叫**Lidstone平滑**。

例如，在所有6个分为C=1的影评样本中，某个特征F1=1不存在，则P(F1=1|C=1)  = 0/6，P(F1=0|C=1)  = 6/6。

经过加法平滑后，P(F1=1|C=1)  = (0+1)/(6+2)=1/8，P(F1=0|C=1)  = (6+1)/(6+2)=7/8。

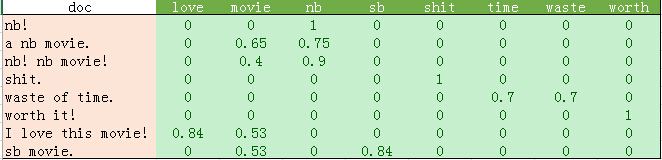
注意分母的+2，这种特殊处理使得2个互斥事件的概率和恒为1。

最后，我们知道，当特征很多的时候，大量小数值的小数乘法会有溢出风险。因此，通常的实现都是将其**转换为log**：

log[P(C)\*P(F1|C)\*P(F2|C)...P(Fn|C)] = log[P(C)]+log[P(F1|C)] + ... +log[P(Fn|C)]

将乘法转换为加法，就彻底避免了乘法溢出风险。

为确保掌握朴素贝叶斯分类原理，我们先使用上一篇文章最后的文本向量化结果做一个例子：



上述训练集中共8个样本，其中C=0的3个，C=1的5个。现在，假设给你一个测试样本"nb movie"，使用加一平滑进行朴素贝叶斯的分类过程如下：

 P(C=0)=3/8， P(C=1)=5/8。特征F1="nb", F2="movie"。

分为C=0的概率：P(F1=1, F2=1|C=0) = P(C=0)\*P(F1=1|C=0)\*P(F2=1|C=0) = 3/8 \* (0+1)/(3+2) \* (1+1)/(3+2) = 3/8 \* 1/5 \* 2/5 = 0.03。

分为C=1的概率：P(F1=1, F2=1|C=1) = P(C=1)\*P(F1=1|C=1)\*P(F2=1|C=1) = 5/8 \* (3+1)/(5+2) \* (3+1)/(5+2) = 5/8 \* 4/7 \* 4/7 = 0.20。

分为C=1的概率更大。因此将该样本分为C=1类。

（注意：实际计算中还要考虑上表中各个值的TF-IDF，具体计算方式取决于使用哪一类贝叶斯分类器。分类器种类见本文最后说明）

## 3、测试数据

本文使用上一篇博客中提到的康奈尔大学网站的2M影评数据集。每一个特征值就是一个单词的TF-IDF。当然，也可以简单的使用单词出现的次数。

使用这个比较大的数据集，可以做一点点数据预处理的优化来避免每次都去硬盘读取文件。第一次运行时，把读入的数据保存起来，以后就不用每次再去读取了。

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41542107)[在CODE上查看代码片](https://code.csdn.net/snippets/534696)

1. #保存
2. movie\_reviews = load\_files('endata')
3. sp.save('movie\_data.npy', movie\_data)
4. sp.save('movie\_target.npy', movie\_target)
6. #读取
7. movie\_data   = sp.load('movie\_data.npy')
8. movie\_target = sp.load('movie\_target.npy')

## 4、代码与分析

Python代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41542107)[在CODE上查看代码片](https://code.csdn.net/snippets/534696)

1. # -\*- coding: utf-8 -\*-
2. **from** matplotlib **import** pyplot
3. **import** scipy as sp
4. **import** numpy as np
5. **from** sklearn.datasets **import** load\_files
6. **from** sklearn.cross\_validation **import** train\_test\_split
7. **from** sklearn.feature\_extraction.text **import**  CountVectorizer
8. **from** sklearn.feature\_extraction.text **import**  TfidfVectorizer
9. **from** sklearn.naive\_bayes **import** MultinomialNB
10. **from** sklearn.metrics **import** precision\_recall\_curve
11. **from** sklearn.metrics **import** classification\_report
13. '''''
14. movie\_reviews = load\_files('data')
15. #保存
16. sp.save('movie\_data.npy', movie\_reviews.data)
17. sp.save('movie\_target.npy', movie\_reviews.target)
18. '''
20. #读取
21. movie\_data   = sp.load('movie\_data.npy')
22. movie\_target = sp.load('movie\_target.npy')
23. x = movie\_data
24. y = movie\_target
26. #BOOL型特征下的向量空间模型，注意，测试样本调用的是transform接口
27. count\_vec = TfidfVectorizer(binary = False, decode\_error = 'ignore',\
28. stop\_words = 'english')

31. #加载数据集，切分数据集80%训练，20%测试
32. x\_train, x\_test, y\_train, y\_test\
33. = train\_test\_split(movie\_data, movie\_target, test\_size = 0.2)
34. x\_train = count\_vec.fit\_transform(x\_train)
35. x\_test  = count\_vec.transform(x\_test)

38. #调用MultinomialNB分类器
39. clf = MultinomialNB().fit(x\_train, y\_train)
40. doc\_class\_predicted = clf.predict(x\_test)
42. #print(doc\_class\_predicted)
43. #print(y)
44. **print**(np.mean(doc\_class\_predicted == y\_test))
46. #准确率与召回率
47. precision, recall, thresholds = precision\_recall\_curve(y\_test, clf.predict(x\_test))
48. answer = clf.predict\_proba(x\_test)[:,1]
49. report = answer > 0.5
50. **print**(classification\_report(y\_test, report, target\_names = ['neg', 'pos']))

输出结果如下所示：

0.821428571429  
             precision    recall  f1-score   support  
        neg       0.78      0.87      0.83       135  
        pos       0.87      0.77      0.82       145  
avg / total     0.83      0.82      0.82       280  
如果进行多次交叉检验，可以发现朴素贝叶斯分类器在这个数据集上能够达到80%以上的准确率。如果你亲自测试一下，会发现KNN分类器在该数据集上只能达到60%的准确率，相信你对朴素贝叶斯分类器应该能够刮目相看了。而且要知道，情感分类这种带有主观色彩的分类准则，连人类都无法达到100%准确。

要注意的是，我们选用的朴素贝叶斯分类器类别：MultinomialNB，这个分类器以出现次数作为特征值，我们使用的TF-IDF也能符合这类分布。

其他的朴素贝叶斯分类器如GaussianNB适用于高斯分布（正态分布）的特征，而BernoulliNB适用于伯努利分布（二值分布）的特征。

# [用Python开始机器学习（7：逻辑回归分类）](http://blog.csdn.net/lsldd/article/details/41551797)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-27 22:12 2558人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41551797#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41551797#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41551797)

在本系列文章中提到过[用Python开始机器学习（3：数据拟合与广义线性回归）](http://blog.csdn.net/lsldd/article/details/41251583)中提到过回归算法来进行数值预测。逻辑回归算法本质还是回归，只是其引入了逻辑函数来帮助其分类。实践发现，逻辑回归在文本分类领域表现的也很优秀。现在让我们来一探究竟。

## 1、逻辑函数

假设数据集有n个独立的特征，x1到xn为样本的n个特征。常规的回归算法的目标是拟合出一个多项式函数，使得预测值与真实值的误差最小：

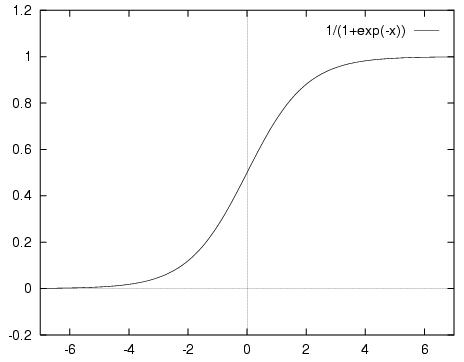
http://img.blog.csdn.net/20141127203300414

而我们希望这样的f(x)能够具有很好的逻辑判断性质，最好是能够直接表达具有特征x的样本被分到某类的概率。比如f(x)>0.5的时候能够表示x被分为正类，f(x)<0.5表示分为反类。而且我们希望f(x)总在[0, 1]之间。有这样的函数吗？

**sigmoid函数**就出现了。这个函数的定义如下：

http://img.blog.csdn.net/20141127205645539

先直观的了解一下，sigmoid函数的图像如下所示（来自<http://computing.dcu.ie/~humphrys/Notes/Neural/sigmoid.html>）：



sigmoid函数具有我们需要的一切优美特性，其定义域在全体实数，值域在[0, 1]之间，并且在0点值为0.5。

那么，如何将f(x)转变为sigmoid函数呢？令p(x)=1为具有特征x的样本被分到类别1的概率，则p(x)/[1-p(x)]被定义为**让步比(odds ratio)**。引入对数：

http://img.blog.csdn.net/20141127210115841

上式很容易就能把p(x)解出来得到下式：

http://img.blog.csdn.net/20141127221754187

现在，我们得到了需要的sigmoid函数。接下来只需要和往常的线性回归一样，拟合出该式中n个参数c即可。

## 2、测试数据

测试数据我们仍然选择康奈尔大学网站的2M影评数据集。

在这个数据集上我们已经测试过KNN分类算法、朴素贝叶斯分类算法。现在我们看看罗辑回归分类算法在处理此类情感分类问题效果如何。

同样的，我们直接读入保存好的movie\_data.npy和movie\_target.npy以节省时间。

## 3、代码与分析

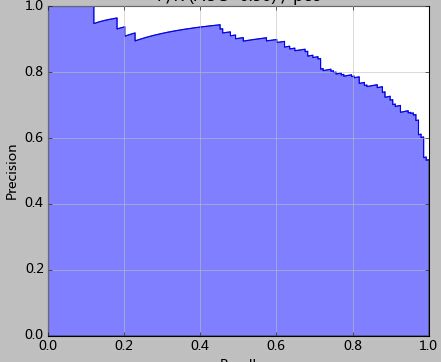
逻辑回归的代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41551797)

1. # -\*- coding: utf-8 -\*-
2. **from** matplotlib **import** pyplot
3. **import** scipy as sp
4. **import** numpy as np
5. **from** matplotlib **import** pylab
6. **from** sklearn.datasets **import** load\_files
7. **from** sklearn.cross\_validation **import** train\_test\_split
8. **from** sklearn.feature\_extraction.text **import**  CountVectorizer
9. **from** sklearn.feature\_extraction.text **import**  TfidfVectorizer
10. **from** sklearn.naive\_bayes **import** MultinomialNB
11. **from** sklearn.metrics **import** precision\_recall\_curve, roc\_curve, auc
12. **from** sklearn.metrics **import** classification\_report
13. **from** sklearn.linear\_model **import** LogisticRegression
14. **import** time
16. start\_time = time.time()
18. #绘制R/P曲线
19. **def** plot\_pr(auc\_score, precision, recall, label=None):
20. pylab.figure(num=None, figsize=(6, 5))
21. pylab.xlim([0.0, 1.0])
22. pylab.ylim([0.0, 1.0])
23. pylab.xlabel('Recall')
24. pylab.ylabel('Precision')
25. pylab.title('P/R (AUC=%0.2f) / %s' % (auc\_score, label))
26. pylab.fill\_between(recall, precision, alpha=0.5)
27. pylab.grid(True, linestyle='-', color='0.75')
28. pylab.plot(recall, precision, lw=1)
29. pylab.show()
31. #读取
32. movie\_data   = sp.load('movie\_data.npy')
33. movie\_target = sp.load('movie\_target.npy')
34. x = movie\_data
35. y = movie\_target
37. #BOOL型特征下的向量空间模型，注意，测试样本调用的是transform接口
38. count\_vec = TfidfVectorizer(binary = False, decode\_error = 'ignore',\
39. stop\_words = 'english')
40. average = 0
41. testNum = 10
42. **for** i **in** range(0, testNum):
43. #加载数据集，切分数据集80%训练，20%测试
44. x\_train, x\_test, y\_train, y\_test\
45. = train\_test\_split(movie\_data, movie\_target, test\_size = 0.2)
46. x\_train = count\_vec.fit\_transform(x\_train)
47. x\_test  = count\_vec.transform(x\_test)
49. #训练LR分类器
50. clf = LogisticRegression()
51. clf.fit(x\_train, y\_train)
52. y\_pred = clf.predict(x\_test)
53. p = np.mean(y\_pred == y\_test)
54. **print**(p)
55. average += p

58. #准确率与召回率
59. answer = clf.predict\_proba(x\_test)[:,1]
60. precision, recall, thresholds = precision\_recall\_curve(y\_test, answer)
61. report = answer > 0.5
62. **print**(classification\_report(y\_test, report, target\_names = ['neg', 'pos']))
63. **print**("average precision:", average/testNum)
64. **print**("time spent:", time.time() - start\_time)
66. plot\_pr(0.5, precision, recall, "pos")

代码运行结果如下：

0.8  
0.817857142857  
0.775  
0.825  
0.807142857143  
0.789285714286  
0.839285714286  
0.846428571429  
0.764285714286  
0.771428571429  
               precision    recall  f1-score   support  
        neg       0.74      0.80      0.77       132  
        pos       0.81      0.74      0.77       148  
avg / total     0.77      0.77      0.77       280  
average precision: 0.803571428571  
time spent: 9.651551961898804  


首先注意我们连续测试了10组测试样本，最后统计出准确率的平均值。另外一种好的测试方法是K折交叉检验（K-Fold）。这样都能更加准确的评估分类器的性能，考察分类器对噪音的敏感性。

其次我们注意看最后的图，这张图就是使用precision\_recall\_curve绘制出来的P/R曲线（precition/Recall）。结合P/R图，我们能对逻辑回归有更进一步的理解。

前文我们说过，通常我们使用0.5来做划分两类的依据。而结合P/R分析，阈值的选取是可以更加灵活和优秀的。

在上图可以看到，如果选择的阈值过低，那么更多的测试样本都将分为1类。因此召回率能够得到提升，显然准确率牺牲相应准确率。

比如本例中，或许我会选择0.42作为划分值——因为该点的准确率和召回率都很高。

最后给一些比较好的资源：

浙大某女学霸的博客！记录的斯坦福Andrew老师主讲的LR公开课笔记：<http://blog.csdn.net/abcjennifer/article/details/7716281>

一个总结LR还不错的博客：<http://xiamaogeng.blog.163.com/blog/static/1670023742013231197530/>

Sigmoid函数详解：<http://computing.dcu.ie/~humphrys/Notes/Neural/sigmoid.html>

# [用Python开始机器学习（8：SVM支持向量机）](http://blog.csdn.net/lsldd/article/details/41581315)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-29 02:13 2300人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41581315#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41581315#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41581315)

SVM支持向量机是建立于统计学习理论上的一种分类算法，适合与处理具备高维特征的数据集。

SVM算法的数学原理相对比较复杂，好在由于SVM算法的研究与应用如此火爆，CSDN博客里也有大量的好文章对此进行分析，下面给出几个本人认为讲解的相当不错的：

支持向量机通俗导论（理解SVM的3层境界）：http://blog.csdn.net/v\_july\_v/article/details/7624837

JULY大牛讲的是如此详细，由浅入深层层推进，以至于关于SVM的原理，我一个字都不想写了。。强烈推荐。

还有一个比较通俗的简单版本的：手把手教你实现SVM算法：<http://blog.csdn.net/alvine008/article/details/9097105>

SVN原理比较复杂，但是思想很简单，一句话概括，就是通过某种核函数，将数据在高维空间里寻找一个最优超平面，能够将两类数据分开。

针对不同数据集，不同的核函数的分类效果可能完全不一样。可选的核函数有这么几种：

线性函数：形如K(x,y)=x\*y这样的线性函数；

多项式函数：形如K(x,y)=[(x·y)+1]^d这样的多项式函数；

径向基函数：形如K(x,y)=exp(-|x-y|^2/d^2）这样的指数函数；

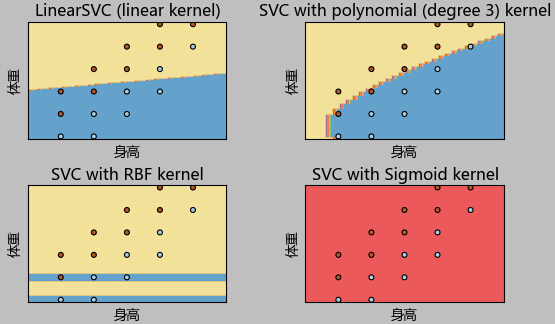
Sigmoid函数：就是上一篇文章中讲到的Sigmoid函数。

我们就利用之前的几个数据集，直接给出Python代码，看看运行效果：

## 测试1：身高体重数据

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41581315)

1. # -\*- coding: utf-8 -\*-
2. **import** numpy as np
3. **import** scipy as sp
4. **from** sklearn **import** svm
5. **from** sklearn.cross\_validation **import** train\_test\_split
6. **import** matplotlib.pyplot as plt
8. data   = []
9. labels = []
10. with open("data\\1.txt") as ifile:
11. **for** line **in** ifile:
12. tokens = line.strip().split(' ')
13. data.append([float(tk) **for** tk **in** tokens[:-1]])
14. labels.append(tokens[-1])
15. x = np.array(data)
16. labels = np.array(labels)
17. y = np.zeros(labels.shape)
18. y[labels=='fat']=1
19. x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.0)
21. h = .02
22. # create a mesh to plot in
23. x\_min, x\_max = x\_train[:, 0].min() - 0.1, x\_train[:, 0].max() + 0.1
24. y\_min, y\_max = x\_train[:, 1].min() - 1, x\_train[:, 1].max() + 1
25. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),
26. np.arange(y\_min, y\_max, h))
28. ''''' SVM '''
29. # title for the plots
30. titles = ['LinearSVC (linear kernel)',
31. 'SVC with polynomial (degree 3) kernel',
32. 'SVC with RBF kernel',
33. 'SVC with Sigmoid kernel']
34. clf\_linear  = svm.SVC(kernel='linear').fit(x, y)
35. #clf\_linear  = svm.LinearSVC().fit(x, y)
36. clf\_poly    = svm.SVC(kernel='poly', degree=3).fit(x, y)
37. clf\_rbf     = svm.SVC().fit(x, y)
38. clf\_sigmoid = svm.SVC(kernel='sigmoid').fit(x, y)
40. **for** i, clf **in** enumerate((clf\_linear, clf\_poly, clf\_rbf, clf\_sigmoid)):
41. answer = clf.predict(np.c\_[xx.ravel(), yy.ravel()])
42. **print**(clf)
43. **print**(np.mean( answer == y\_train))
44. **print**(answer)
45. **print**(y\_train)
47. plt.subplot(2, 2, i + 1)
48. plt.subplots\_adjust(wspace=0.4, hspace=0.4)
50. # Put the result into a color plot
51. z = answer.reshape(xx.shape)
52. plt.contourf(xx, yy, z, cmap=plt.cm.Paired, alpha=0.8)
54. # Plot also the training points
55. plt.scatter(x\_train[:, 0], x\_train[:, 1], c=y\_train, cmap=plt.cm.Paired)
56. plt.xlabel(u'身高')
57. plt.ylabel(u'体重')
58. plt.xlim(xx.min(), xx.max())
59. plt.ylim(yy.min(), yy.max())
60. plt.xticks(())
61. plt.yticks(())
62. plt.title(titles[i])
64. plt.show()

运行结果如下：  


可以看到，针对这个数据集，使用3次多项式核函数的SVM，得到的效果最好。

## 测试2：影评态度

下面看看SVM在康奈尔影评数据集上的表现：（代码略）

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='linear', max\_iter=-1, probability=False, random\_state=None,  
  shrinking=True, tol=0.001, verbose=False)  
0.814285714286

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='poly', max\_iter=-1, probability=False, random\_state=None,  shrinking=True, tol=0.001, verbose=False)  
0.492857142857

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='rbf', max\_iter=-1, probability=False, random\_state=None,  shrinking=True, tol=0.001, verbose=False)  
0.492857142857

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='sigmoid', max\_iter=-1, probability=False, random\_state=None,  
  shrinking=True, tol=0.001, verbose=False)  
0.492857142857  
可见在该数据集上，线性分类器效果最好。

## 测试3：圆形边界

最后我们测试一个数据分类边界为圆形的情况：圆形内为一类，原型外为一类。看这类非线性的数据SVM表现如何：

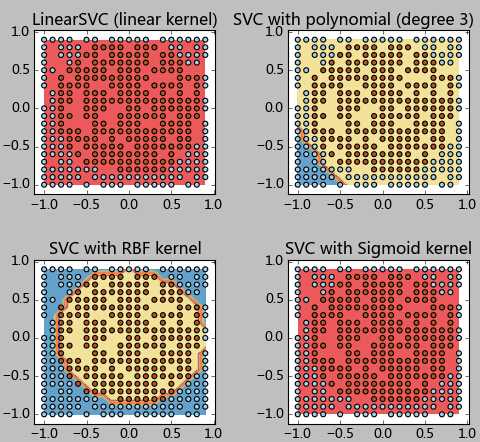
测试数据生成代码如下所示：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41581315)[在CODE上查看代码片](https://code.csdn.net/snippets/536069)

1. ''''' 数据生成 '''
2. h = 0.1
3. x\_min, x\_max = -1, 1
4. y\_min, y\_max = -1, 1
5. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),
6. np.arange(y\_min, y\_max, h))
7. n = xx.shape[0]\*xx.shape[1]
8. x = np.array([xx.T.reshape(n).T, xx.reshape(n)]).T
9. y = (x[:,0]\*x[:,0] + x[:,1]\*x[:,1] < 0.8)
10. y.reshape(xx.shape)
12. x\_train, x\_test, y\_train, y\_test\
13. = train\_test\_split(x, y, test\_size = 0.2)

测试结果如下：

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='linear', max\_iter=-1, probability=False, random\_state=None,  
  shrinking=True, tol=0.001, verbose=False)  
0.65  
SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='poly', max\_iter=-1, probability=False, random\_state=None,  
  shrinking=True, tol=0.001, verbose=False)  
0.675  
SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='rbf', max\_iter=-1, probability=False, random\_state=None,  
  shrinking=True, tol=0.001, verbose=False)  
0.9625  
SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, degree=3, gamma=0.0,  kernel='sigmoid', max\_iter=-1, probability=False, random\_state=None,  
  shrinking=True, tol=0.001, verbose=False)  
0.65

  
可以看到，对于这种边界，径向基函数的SVM得到了近似完美的分类结果。而其他的分类器显然束手无策。

# [用Python开始机器学习（9：推荐算法之推荐矩阵）](http://blog.csdn.net/lsldd/article/details/41598751)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-30 00:52 682人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41598751#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41598751#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41598751)

每个人都会有这样的经历：当你在电商网站购物时，你会看到天猫给你弹出的“和你买了同样物品的人还买了XXX”的信息；当你在SNS社交网站闲逛时，也会看到弹出的“你可能认识XXX“的信息；你在微博添加关注人时，也会看到“你可能对XXX也感兴趣”；等等。

所有这一切，都是背后的推荐算法运作的结果。最经典的关联规则算法是大名鼎鼎的Apriori算法，源自一个超市购物篮的故事：啤酒总是和尿布一起被购买。有兴趣的可以去看看。

本章我们来学习一种最简单的推荐算法：推荐矩阵。虽然简单，但是却被广泛应用着。

## 1、推荐矩阵

为描述方便，以下我们以“购物推荐”作为背景进行介绍。假设你有个卖商品的网站，拥有每个用户购买每个物品的数据。现在，某个用户A购买了商品a，如何向他推荐他最有可能感兴趣的其他商品呢？

为达到这个目的，通常有两种思路：

1：寻找与该用户（A）购买习惯最为相似的用户（B)，认为B购买的物品，A也最有可能感兴趣。

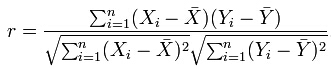
这种情况适用于A已经购买过一些商品，算法能够根据A已经购买的物品作为特征，去匹配与A购买习惯最相近的用户。这种方式是以用户为中心的，推荐出来的商品b可能跟商品a风流马不相及，因此更适合于类似SNS和微博这样的平台，根据用户的已知兴趣集合来向其推荐其他具有相同兴趣的用户；

2：寻找与商品（a）最为相似的商品（b)，认为A既然对a感兴趣，也有可能对b感兴趣；

这种情况是以商品为中心的，因此更适合购物推荐这样的场景。

 而要计算两个向量“最相似”的程度，有很多方法，如KNN中用到的欧式距离，或者海明距离等。但是欧式距离并不适用于本场合。比如用户A购买了5个商品a，5个商品b，用户B购买了5个商品a，0个商品b，用户C购买了10个商品a，10个商品b，用距离来度量的结果必然是A与B更近。而实际上A跟C是极其相似的。

因此这里，我们介绍**皮尔森相关系数**(Pearson correlation coefficient)。其定义如下：



该系数定义的是两个向量的线性相关程度，取值范围为[-1,+1]，0表示线性无关，绝对值越大线性相关程度越大，正/负表示正/负线性相关。

简单的给几个例子：

[1,2,3]，[4,5,6]：1

[1,2,3]，[6,5,4]：-1

[1,2,3]，[1,2,4]：0.98

[1,2,3]，[-1,-11,-111]：-0.9

因此，我们将构造一个矩阵来描述用户购买商品的情况，该矩阵以用户为行、商品为列。要计算某个商品a最相似的商品，我们通过计算商品a所在的列与其他的每一列的皮尔森相关系数，找出最大的前N个推荐给用户即可。

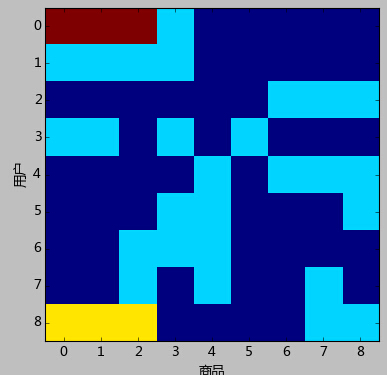
## 2、测试数据

数据为一份简单的购物清单，每一行对应着用户id——商品id这样的数据对。如下图所示：

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41598751)[在CODE上查看代码片](https://code.csdn.net/snippets/536898)

1. 1 1 3
2. 1 2 3
3. 1 3 3
4. 1 4 1
5. 2 1 1
6. 2 2 1
7. 2 3 1
8. 2 4 1
9. ......

如第一行对应着用户1购买了商品1，数量为3。可以认为该数据是一个稀疏矩阵。该矩阵可视化出来结果如下：



上图中每一行代表一个用户，每一列代表一个商品，对应的颜色不同表示购买的数量不同，深蓝色表示购买数为0。

从图上很容易看出，用户0与用户1同时购买了商品0，1，2，仅仅数量不一样；而商品0和商品1售出的情况一模一样——只被用户0，1，3，8购买过，看上去就像是捆绑销售的一般。

## 3、代码与分析

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41598751)[在CODE上查看代码片](https://code.csdn.net/snippets/536898)

1. # -\*- coding: utf-8 -\*-
2. **from** matplotlib **import** pyplot
3. **import** scipy as sp
4. **import** numpy as np
5. **from** matplotlib **import** pylab
6. **from** sklearn.datasets **import** load\_files
7. **from** sklearn.cross\_validation **import** train\_test\_split
8. **from** sklearn.metrics **import** precision\_recall\_curve, roc\_curve, auc
9. **from** sklearn.metrics **import** classification\_report
11. **import** time
12. **from** scipy **import** sparse
14. start\_time = time.time()
16. #计算向量test与data数据每一个向量的相关系数，data一行为一个向量
17. **def** calc\_relation(testfor, data):
18. **return** np.array(
19. [np.corrcoef(testfor, c)[0,1]
20. **for** c **in** data])
22. # luispedro提供的加速函数:
23. **def** all\_correlations(y, X):
24. X = np.asanyarray(X, float)
25. y = np.asanyarray(y, float)
26. xy = np.dot(X, y)
27. y\_ = y.mean()
28. ys\_ = y.std()
29. x\_ = X.mean(1)
30. xs\_ = X.std(1)
31. n = float(len(y))
32. ys\_ += 1e-5  # Handle zeros in ys
33. xs\_ += 1e-5  # Handle zeros in x
34. **return** (xy - x\_ \* y\_ \* n) / n / xs\_ / ys\_

37. #数据读入
38. data = np.loadtxt('1.txt')
39. x\_p = data[:, :2] # 取前2列
40. y\_p = data[:,  2] # 取前2列
41. x\_p -= 1          # 0为起始索引
42. y = (sparse.csc\_matrix((data[:,2], x\_p.T)).astype(float))[:, :].todense()
43. nUser, nItem = y.shape

46. #可视化矩阵
47. pyplot.imshow(y, interpolation='nearest')
48. pyplot.xlabel('商品')
49. pyplot.ylabel('用户')
50. pyplot.xticks(range(nItem))
51. pyplot.yticks(range(nUser))
52. pyplot.show()

55. #加载数据集，切分数据集80%训练，20%测试
56. x\_p\_train, x\_p\_test, y\_p\_train, y\_p\_test = \
57. train\_test\_split(data[:,:2], data[:,2], test\_size = 0.0)
58. x = (sparse.csc\_matrix((y\_p\_train, x\_p\_train.T)).astype(float))[:, :].todense()

61. Item\_likeness = np.zeros((nItem, nItem))
63. #训练
64. **for** i **in** range(nItem):
65. Item\_likeness[i] = calc\_relation(x[:,i].T, x.T)
66. Item\_likeness[i,i] = -1
68. **for** t **in** range(Item\_likeness.shape[1]):
69. item = Item\_likeness[t].argsort()[-3:]
70. **print**("Buy Item %d will buy item %d,%d,%d "%
71. (t, item[0], item[1], item[2]))
73. **print**("time spent:", time.time() - start\_time)

输出如下：

Buy Item 0, recommond item 3,2,1   
Buy Item 1, recommond item 3,2,0   
Buy Item 2, recommond item 3,0,1   
Buy Item 3, recommond item 0,1,5   
Buy Item 4, recommond item 6,7,8   
Buy Item 5, recommond item 0,1,3   
Buy Item 6, recommond item 4,7,8   
Buy Item 7, recommond item 4,8,6   
Buy Item 8, recommond item 4,7,6   
time spent: 1.9111089706420898

代码中，我们计算了每一个Item与其他所有item的相关性，然后排序选取最大的前3个作为推荐。

最需要注意的是，真正的应用中，大量的用户与大量的商品之间建立矩阵，计算量是巨大的。从皮尔森相关系数的定义来看，其计算量也是巨大的。因此代码中给了luispedro提供的一种计算相关系数的替换函数，效率能提高不少。

# [用Python开始机器学习（10：聚类算法之K均值）](http://blog.csdn.net/lsldd/article/details/41624305)

分类： [机器学习](http://blog.csdn.net/lsldd/article/category/2709209)2014-11-30 21:39 744人阅读 [评论](http://blog.csdn.net/lsldd/article/details/41624305#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lsldd/article/details/41624305#report)

目录[(?)[+]](http://blog.csdn.net/lsldd/article/details/41624305)

我们之前接触的所有机器学习算法都有一个共同特点，那就是分类器会接受2个向量：一个是训练样本的特征向量X，一个是样本实际所属的类型向量Y。由于训练数据必须指定其真实分类结果，因此这种机器学习统称为**有监督学习**。

然而有时候，我们只有训练样本的特征，而对其类型一无所知。这种情况，我们只能让算法尝试在训练数据中寻找其内部的结构，试图将其类别挖掘出来。这种方式叫做**无监督学习**。由于这种方式通常是将样本中相似的样本聚集在一起，所以又叫**聚类算法**。

下面我们介绍一个最常用的聚类算法：K均值聚类算法（K-Means）。

## 1、K均值聚类

K-Means算法思想简单，效果却很好，是最有名的聚类算法。聚类算法的步骤如下：

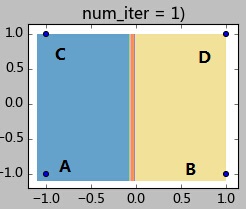
1：初始化K个样本作为初始聚类中心；

2：计算每个样本点到K个中心的距离，选择最近的中心作为其分类，直到所有样本点分类完毕；

3：分别计算K个类中所有样本的质心，作为新的中心点，完成一轮迭代。

通常的迭代结束条件为新的质心与之前的质心偏移值小于一个给定阈值。

下面给一个简单的例子来加深理解。如下图有4个样本点，坐标分别为A(-1,-1),B(1,-1),C(-1,1),D(1,1)。现在要将他们聚成2类，指定A、B作为初始聚类中心（聚类中心A0, B0），指定阈值0.1。K-Means迭代过程如下：



step 1.1：计算各样本距离聚类中心的距离：

样本A：d(A,A0) = 0; d(A,B0) = 2;因此样本A属于A0所在类；

样本B：d(B,A0) = 2; d(B,B0) = 0;因此样本B属于B0所在类；

样本C：d(C,A0) = 2; d(C,B0) = 2.8;;因此样本C属于A0所在类；

样本C：d(D,A0) = 2.8; d(D,B0) = 2;;因此样本C属于B0所在类；

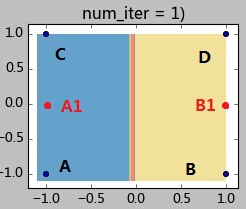
step 1.2：全部样本分类完毕，现在计算A0类（包含样本AC）和B0类（包含样本BD）的新的聚类中心：

A1 = (-1, 0); B1 = (1,0);

step 1.3：计算聚类中心的偏移值是否满足终止条件：

|A1-A0| = |(-1,0)-(-1,-1) | = |(0,1)| = 1 >0.1，因此继续迭代。

此时的状态如下图所示：



step 2.1：计算各样本距离聚类中心的距离：

样本A：d(A,A1) = 1; d(A,B1) = 2.2;因此样本A属于A1所在类；

样本B：d(B,A1) = 2.2; d(B,B1) = 1;因此样本B属于B1所在类；

样本C：d(C,A1) = 1; d(C,B1) = 2.2;;因此样本C属于A1所在类；

样本D：d(D,A1) = 2.2; d(D,B1) = 1;;因此样本C属于B1所在类；

step 2.2：全部样本分类完毕，现在计算A1类（包含样本AC）和B1类（包含样本BD）的新的聚类中心：

A2 = (-1, 0); B2 = (1,0);

step 2.3：计算聚类中心的偏移值是否满足终止条件：

|A2-A1| = |B2-B1| = 0 <0.1，因此迭代终止。

## 2、测试数据

下面这个测试数据有点类似SNS中的好友关系，假设是10个来自2个不同的圈子的同学的SNS聊天记录。显然，同一个圈子内的同学会有更密切的关系和互动。

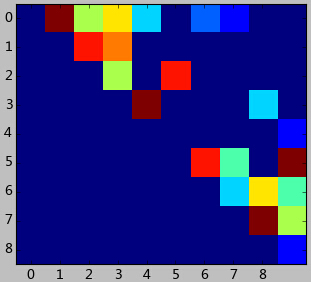
数据如下所示，每一行代表一个好友关系。如第一行表示同学0与同学1的亲密程度为9（越高表示联系越密切）。

显然，这个数据中并没有告知我们这10个同学分别属于哪个圈子。因此我们的目标是使用K-Means聚类算法，将他们聚成2类。

**[plain]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41624305)

1. 0 1 9
2. 0 2 5
3. 0 3 6
4. 0 4 3
5. 1 2 8
6. ......

这个例子设计的很简单。我们使用上一篇文章中提到的关系矩阵，将其可视化出来，会看到如下结果：



这是个上三角矩阵，因为这个数据中认为好友关系是对称的。上图其实很快能发现，0,1,2,3,4用户紧密联系在一起，而5,6,7,8,9组成了另外一个圈子。

下面我们看看K-Means算法能否找出这个答案。

## 3、代码与分析

K-Means算法的Python代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/lsldd/article/details/41624305)[在CODE上查看代码片](https://code.csdn.net/snippets/537637)

1. # -\*- coding: utf-8 -\*-
2. **from** matplotlib **import** pyplot
3. **import** scipy as sp
4. **import** numpy as np
5. **from** sklearn **import** svm
6. **import** matplotlib.pyplot as plt
7. **from** sklearn.cluster   **import** KMeans
8. **from** scipy **import** sparse
10. #数据读入
11. data = np.loadtxt('2.txt')
12. x\_p = data[:, :2] # 取前2列
13. y\_p = data[:,  2] # 取前2列
14. x = (sparse.csc\_matrix((data[:,2], x\_p.T)).astype(float))[:, :].todense()
15. nUser = x.shape[0]
17. #可视化矩阵
18. pyplot.imshow(x, interpolation='nearest')
19. pyplot.xlabel('用户')
20. pyplot.ylabel('用户')
21. pyplot.xticks(range(nUser))
22. pyplot.yticks(range(nUser))
23. pyplot.show()
25. #使用默认的K-Means算法
26. num\_clusters = 2
27. clf = KMeans(n\_clusters=num\_clusters,  n\_init=1, verbose=1)
28. clf.fit(x)
29. **print**(clf.labels\_)
31. #指定用户0与用户5作为初始化聚类中心
32. init = np.vstack([ x[0], x[5] ])
33. clf = KMeans(n\_clusters=2, init=init)
34. clf.fit(x)
35. **print**(clf.labels\_)

输出结果如下：

Initialization complete  
Iteration  0, inertia 581.000  
Iteration  1, inertia 417.643  
Converged at iteration 1  
[0 0 1 1 1 1 1 1 1]

[0 0 0 0 1 1 1 1 1]

可以看到，使用默认的K-Means算法将使用随机的初始值，因此每次执行的结果都不一样。

上面的输出中将0,1用户聚类到一起，效果并不理想。然而，如果我们可以确定用户0与用户5是有很大区别的，就可以指定用户0和用户5作为K-Means聚类算法的初始值。可以看到和我们的预期完全一致，这样效果就非常好了。

由于K-Means毕竟是无监督学习，在很多情况下自然无法与有监督学习的算法进行同样标准的比较。但其不需要监督的特性，广泛应用与社交图谱（如本例）、相似性匹配（如搜索相似的新闻、帖子）等引用场景。