机器学习——Logistic回归（下）

本篇主要对Logistic的梯度上升算法作了一定的改进，重新改成随机梯度上升算法；接着使用sklearn对一个实例进行了应用示范。

随机梯度上升算法：

在上一节我们的梯度上升算法中，我们需要遍历整一个数据集进行回归系数的更新：

```
def gradAscent(dataMatIn,classLabels):
    dataMatrix = np.mat(dataMatIn)
    labelMat = np.mat(classLabels).transpose()
    m,n =np.shape(dataMatrix)
    alpha =0.001
    maxCycles = 500
    weights=np.ones((n,1))
    for k in range(maxCycles):
        h=sigmoid(dataMatrix*weights)
        error=labelMat-h
        weights=weights+alpha*dataMatrix.transpose()*error
    return weights.getA()
```

在小样本数据这样是没有问题的，但是如果数据集太大，所要进行的乘法运算就太多了，因此需要我们每次更新回归系数的时候，采用一种新的方法，不采用所有样本，而一次只用一个样本点去更新回归系数，就减小了运算量，这就是随机梯度上升：

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

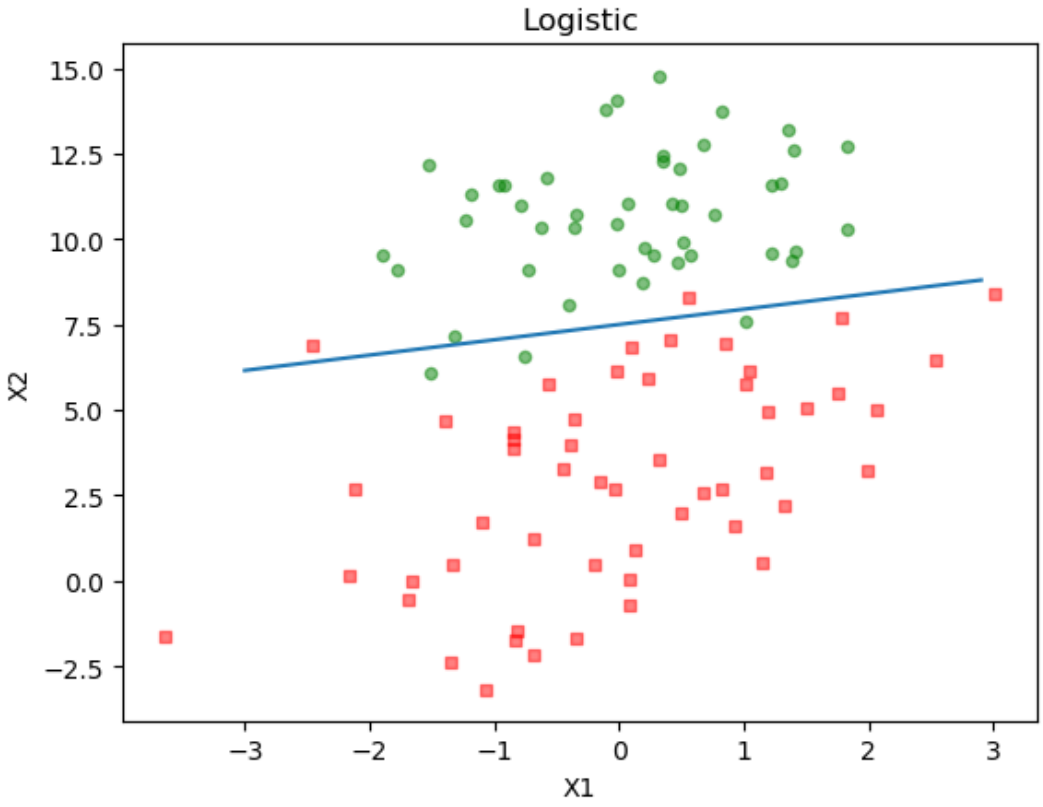
可以看到这个算法每次的alpha在迭代的时候是会不断变小的，但不会变成0，保证新数据还有影响，如果要处理的问题是动态的，那么可以适当加大常数项，确保获得更大的回归系数，另外，在降低alpha的方法中，我们这种方法是不严格下降的，避免这种情况的方法是模拟退火方法等，大家可以去了解下；同时这个算法另外的一个改进地方是随机选取样本来更新回归系数，减少了周期性的波动。实现方法就是每次随机从列表中选取一个值，然后从列表中删掉该值，再进行下一次迭代。

我们在上一次的基础上换上随机梯度上升算法，可以看到：

```
import numpy as np
import matplotlib.pyplot as plt
import random
```

```
"""
加载数据
把数据按一行一行都进去
然后返回数据矩阵以及相应的标签矩阵
"""
def loadDataSet():
    dataMat=[]
    labelMat=[]
    fr=open('testSet.txt')
    for line in fr.readlines() :
        lineArr = line.strip().split()
        dataMat.append([1.0,float(lineArr[0]),float(lineArr[1])])
        labelMat.append(int(lineArr[2]))
    fr.close()
    return dataMat,labelMat

"""
sigmoid 函数:
返回sigmoid函数的值
"""
def sigmoid(inx):
    return 1.0/(1+np.exp(-inx))


def gradAscent(dataMatIn,classLabels):
    dataMatrix = np.mat(dataMatIn)
    labelMat = np.mat(classLabels).transpose()
    m,n =np.shape(dataMatrix)
    alpha =0.001
    maxCycles = 500
    weights=np.ones((n,1))
    for k in range(maxCycles):
        h=sigmoid(dataMatrix*weights)
        error=labelMat-h
        weights=weights+alpha*dataMatrix.transpose()*error
    return weights.getA()

def stocGradAscent1(dataMatrix,classLabels,numIter=150):
    m,n=np.shape(dataMatrix)
    weights=np.ones(n)
    for j in range(numIter):
        dataIndex=list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01
            randIndex=int(random.uniform(0,len(dataIndex)))
            h=sigmoid(sum(dataMatrix[randIndex]*weights))
            error=classLabels[randIndex]-h
            weights=weights+alpha*error*dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights

dataMat,labelMat=loadDataSet()
print(gradAscent(dataMat,labelMat))

def plotFit(weights):
    dataMat,labelMat = loadDataSet()
    dataArr=np.array(dataMat)
    n=np.shape(dataMat)[0]
```

```
    xcord1=[];ycord1=[];
    xcord2=[];ycord2=[];
    for i in range(n):
        if int(labelMat[i])==1:
            xcord1.append(dataArr[i,1])
            ycord1.append(dataArr[i,2])
        else:
            xcord2.append(dataArr[i,1])
            ycord2.append(dataArr[i,2])
    fig = plt.figure()
    ax=fig.add_subplot(111)
    ax.scatter(xcord1, ycord1, s = 20, c = 'red', marker = 's',alpha=.5)
    ax.scatter(xcord2,ycord2,s=20,c='green',alpha=.5)
    x=np.arange(-3.0,3.0,0.1)
    y=(-weights[0]-weights[1]*x)/weights[2]
    ax.plot(x,y)
    plt.title('Logistic')
    plt.xlabel('X1'); plt.ylabel('X2')
    plt.show()

#weights=gradAscent(dataMat,labelMat)
weights=stocGradAscent1(np.array(dataMat),labelMat)
plotFit(weights)
```



实战演练：

数据处理：

对于数据中的缺失值，有很多种处理方法：

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来填补缺失值；

使用另外的机器学习算法预测缺失值；

针对我们使用的分类算法，如果测试集中的一条数据特征值已经确实，那么选择实数0来替换，因为Logistic回顾中sigmoid（0）=0.5，对于结果没有任何倾向性影响，如果标签缺失值，则需要把这个类别数据丢掉，因为很难确定采用某个合适的值来替换。

在这个章节中，我们使用了已经处理好的数据集进行处理，我们只需要搭建一个Logistic分类器，就可以进行分类：（实验使用Logistic回归来预测患疝气病的马的存活问题，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，数据集中有30%的值是缺失的。）

我们使用梯度上升的方法对Logistic进行回归分类：数据集在我的github上有提供（两个数据集：一个测试、一个训练）

```
import numpy as np
import random

def sigmoid(inX):
    return 1.0/(1+np.exp(-inX))

def gradAscent(dataMatIn,classLabels):
    dataMatrix = np.mat(dataMatIn)
    labelMat = np.mat(classLabels).transpose()
    m,n =np.shape(dataMatrix)
    alpha =0.001
    maxCycles = 500
    weights=np.ones((n,1))
    for k in range(maxCycles):
        h=sigmoid(dataMatrix*weights)
        error=labelMat-h
        weights=weights+alpha*dataMatrix.transpose()*error
    return weights.getA()

def stocGradAscent1(dataMatrix,classLabels,numIter=150):
    m,n=np.shape(dataMatrix)
    weights=np.ones(n)
    for j in range(numIter):
        dataIndex=list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01
            randIndex=int(random.uniform(0,len(dataIndex)))
            h=sigmoid(sum(dataMatrix[randIndex]*weights))
            error=classLabels[randIndex]-h
            weights=weights+alpha*error*dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights

def classifyVector(inX,weights):
    prob =sigmoid(sum(inX*weights))
    if prob>0.5:
        return 1.0
    else:
        return 0.0
```

```
def colicTest():
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainingSet =[]
    traininngLabels=[]
    for line in frTrain.readlines() :
        currLine = line.strip().split('\t')
        lineArr=[]
        for  i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        traininngLabels.append(float(currLine[-1]))
    trainWeights=gradAscent(np.array(trainingSet),traininngLabels)
    errorCount = 0
    numTestVec=0.0
    for line in frTest.readlines():
        numTestVec += 1
        currLine =line.strip().split('\t')
        lineArr=[]
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(np.array(lineArr),trainWeights[:,0]))!=
int(currLine[-1]):
            errorCount+=1
    errorRate=(float(errorCount)/numTestVec)*100
    print("测试集错误率为：%.2f%%" % errorRate)


colicTest()
```

> 测试集错误率为：29.85%

如果我们使用随机梯度上升算法：

```
import numpy as np
import random

def sigmoid(inX):
    return 1.0/(1+np.exp(-inX))

def gradAscent(dataMatIn,classLabels):
    dataMatrix = np.mat(dataMatIn)
    labelMat = np.mat(classLabels).transpose()
    m,n =np.shape(dataMatrix)
    alpha =0.001
    maxCycles = 500
    weights=np.ones((n,1))
    for k in range(maxCycles):
        h=sigmoid(dataMatrix*weights)
        error=labelMat-h
        weights=weights+alpha*dataMatrix.transpose()*error
    return weights.getA()

def stocGradAscent1(dataMatrix,classLabels,numIter=150):
    m,n=np.shape(dataMatrix)
```

```
    weights=np.ones(n)
    for j in range(numIter):
        dataIndex=list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01
            randIndex=int(random.uniform(0,len(dataIndex)))
            h=sigmoid(sum(dataMatrix[randIndex]*weights))
            error=classLabels[randIndex]-h
            weights=weights+alpha*error*dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights

def classifyVector(inX,weights):
    prob =sigmoid(sum(inX*weights))
    if prob>0.5:
        return 1.0
    else:
        return 0.0

def colicTest():
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainingSet =[]
    traininngLabels=[]
    for line in frTrain.readlines() :
        currLine = line.strip().split('\t')
        lineArr=[]
        for  i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        traininngLabels.append(float(currLine[-1]))
    trainWeights=stocGradAscent1(np.array(trainingSet),traininngLabels)
    errorCount = 0
    numTestVec=0.0
    for line in frTest.readlines():
        numTestVec += 1
        currLine =line.strip().split('\t')
        lineArr=[]
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(np.array(lineArr),trainWeights))!=
int(currLine[-1]):
            errorCount+=1
    errorRate=(float(errorCount)/numTestVec)*100
    print("测试集错误率为：%.2f%%" % errorRate)


colicTest()
```

> 测试集错误率为：32.84%

可以看到在数据集比较小时，我们采用梯度上升算法时相对较好的，但是对于数据集较大时，我们使用改进的随机梯度上升算法则体现出优越性，对应于sklearn，数据集小时使用liblinear，数据集大时使用sag和saga。

sklearn实战：

对于sklearn的应用，大家可以看一下官方的文档，对函数有一定的理解就很轻松学会了，下面直接给出代码（对于上述实战例子）：

```
from sklearn.linear_model import LogisticRegression

def colicTest():
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainingSet =[]
    traininngLabels=[]
    testSet=[]
    testLabels=[]
    for line in frTrain.readlines() :
        currLine = line.strip().split('\t')
        lineArr=[]
        for  i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        traininngLabels.append(float(currLine[-1]))

classifier=LogisticRegression(solver='liblinear',max_iter=10).fit(trainingSet,tr
aininngLabels)
    for line in frTest.readlines():
        currLine =line.strip().split('\t')
        lineArr=[]
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        testSet.append(lineArr)
        testLabels.append(float(currLine[-1]))
    test_accurcy=classifier.score(testSet,testLabels)*100
    print("测试准确率率为：%.2f%%" % test_accurcy)

colicTest()
```

> 测试准确率率为：73.13%
> [Finished in 4.5s]

总结：

总的来说，Logistic实现和核心思想是很简单的，计算代价也不高，但是容易欠拟合，分类精度不高。Logistic回归用到了梯度上升方法，我也给出了一些优化的方法，我们需要根据数据集的大小调整使用的方法，同时怎么去处理数据集也是我们平时经常遇到的问题，综合考虑这些问题，并试着去调节sklearn的参数，我们会发现更好的分类效果。