An Introduction to logistic regression

Introduction

In this tip, I will introduce about an optimization algorithm, logistic regression. In statistics, the logistic model is a statistical model that is usually taken to apply to a binary dependent variable. In regression analysis, logistic regression is estimating the parameters of a logistic model (wikipedia https://en.wikipedia.org/wiki/Logistic_regression). I'll also introduce about the gradient ascent and a its modified version called the stochastic gradient ascent.

Background

I have learned from many sources and you can refer to them in the References section. This is summary of my learning.

Logistic regression

Logistic regression is a classification algorithm that works by trying to learn a function that approximates $P(Y|X)$. It makes the central assumption that P(Y|X) can be approximated as a

sigmoid function applied to a linear combination of input features. This assumption is often written in the equivalent forms:

$$P(Y = 1 \mid \mathbf{X} = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$
$$P(Y = 0 \mid \mathbf{X} = \mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x})$$

where:

$$\theta^T \mathbf{x} = \sum_{i=1}^{m} \theta_i x_i = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

In logistic regression, **θ (theta)** is a vector of parameters of length *m* and we are going to learn the values of those parameters based off of *n* training examples. The number of parameters should be equal to the number of features (x_i) of each data point. The function **σ*(z)*** *(sigma z)* is called the **logistic function** (or *sigmoid function*).

Log Likelihood

In order to chose values for the parameters of logistic regression, we use maximum likelihood estimation (MLE). We can write the likelihood of all the data:

$$L(\theta) = \prod_{i=1}^{n} P(Y = y^{(i)} \mid X = \mathbf{x}^{(i)}) \qquad \text{the likelihood of independent training labels}$$

$$= \prod_{i=1}^{n} \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} \cdot \left[1 - \sigma(\theta^T \mathbf{x}^{(i)})\right]^{(1-y^{(i)})} \qquad \text{substituting the likelihood of a Bernoulli}$$

The log likelihood equation is:

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Gradient of Log Likelihood

Now that we have a function for log-likelihood, we simply need to chose the values of theta that maximize it. We can find the best values of theta by using an optimization algorithm. The optimization algorithm we will use requires the partial derivative of log likelihood with respect to each parameter:

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)})\right] x_j^{(i)}$$

We are ready for our optimization algorithm. To do so we employ an algorithm called gradient ascent . The idea behind gradient ascent is that gradients point “uphill”. If you continuously take small steps in the direction of your gradient, you will eventually make it to a local maximum. The update to our parameters that results in each small step can be calculated as:

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)})\right] x_j^{(i)}$$

Where η (eta) is the magnitude of the step size that we take or also called the **learning rate**. If you keep updating θ using the equation above, you will converge on the best values of θ. The expression in the square bracket pair is also known as the **error** between the actual class

and the predicted class.

Using the code

We will focus on the **binary classification** problem in which y can take on only two values, 0 and 1. 0 is also called the **negative class**, and 1 the **positive class**, and they are sometimes also denoted by the symbols "-" and "+." Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** for the training example.

Suppose we have 10 data points and each point has three numeric features: X1, X2 and X3. To simplify, we will assume X1 = 1. So we have:

$$\theta^T x = \sum_{i=1}^{3} \theta_i x_i = \theta_1 + \theta_2 x_2 + \theta_3 x_3$$

Our data points will be stored in the **mySample.txt** file, its content can look like this:

X2	X3	Label
-3	-2	0
-2	3	0
-1	-4	0
2	3	0
3	4	0
-1	9	1
2	14	1
1	17	1
3	12	1
0	8	1

X2	X3	Label
-3	-2	0
-2	3	0
-1	-4	0
2	3	0
3	4	0
-1	9	1
2	14	1

1	17	1
3	12	1
0	8	1

We will insert the X0 =1 to each data point later by using the Python code. The following **loadDataSet**() will load data from the **mySample.txt**:

```
def loadDataSet():
      # storing Xi, where i = 1,2,3
      dataMat = [];
      # storing labels, 0 or 1
      labelMat = []
      fr = open('mySample.txt')
      for line in fr.readlines():
            lineArr = line.strip().split()
            # notes: we are inserting X1 = 1 to the dataMat
            dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
            labelMat.append(int(lineArr[2]))
      return dataMat,labelMat
```

We can try to test the result of the **dataMat** and the **lableMat** by using the following the lines of code:

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

The **dataMat** looks like this:

```
[[1.0, -3.0, -2.0], [1.0, -2.0, 3.0], [1.0, -1.0, -4.0], [1.0, 2.0, 3.0],
[1.0, 3.0, 4.0], [1.0, -1.0, 9.0], [1.0, 2.0, 14.0], [1.0, 1.0, 17.0], [1.0,
3.0, 12.0], [1.0, 0.0, 8.0]]
```

The **labelMat**:

```
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

We'll try to use gradient ascent to fit the best parameters or weights (θ_i) for the logistic regression model to our data. The theta can be calculated as follows:

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

The lines of Python code can look like this:

```
dataMat, labelMat = loadDataSet()

# convert the dataMat to a matrix object

dataMatrix = mat(dataMat)

# convert the dataMat to a matrix object and transpose this matrix

labelMatrix = mat(labelMat).transpose()

# get m, n from the dataMatrix

# m is the number of data points, n is the number of features (xi) of each
data point

m,n = shape(dataMatrix)

eta = 0.001

thetas = ones((n,1))

# using the sigmoid function

sig = sigmoid(dataMatrix*thetas)

error = (labelMatrix - h)

# calculate thetas

thetas = thetas + eta * dataMatrix.transpose()* error
```

The learning process must be repeated many times (such as 500 times), so the lines of code can be rewritten as follows:

```
...

numIter = 500

for k in range(numIter):

      # using the sigmoid function
```

```
      sig = sigmoid(dataMatrix*thetas)
      error = (labelMatrix - h)
      # calculate thetas
      thetas = thetas + eta * dataMatrix.transpose()* error
```

All of above code can be put into the **gradAscent**() function:

```
      def gradAscent(dataMat, labelMat, numIter):
            # convert the dataMat to a matrix object
            dataMatrix = mat(dataMat)
            # convert the dataMat to a matrix object and transpose this
            # matrix
            labelMatrix = mat(labelMat).transpose()
            # get m, n from the dataMatrix
            # m is the number of data points, n is the number of features
            # (xi) of each data point
            m,n = shape(dataMatrix)
            eta = 0.001
            thetas = ones((n,1))
            for k in range(numIter):
                  # using the sigmoid function
                  sig = sigmoid(dataMatrix*thetas)
                  error = (labelMatrix - sig)
                  # calculate thetas
                  thetas = thetas + eta * dataMatrix.transpose()* error
            return thetas
```

We're solving for a set of weights (or parameters) used to make a line that separates the different classes of data (0 and 1) and we can make a plot this line by using matplotlib. The following lines of code will make a scatter plot of data points in two classes (0 and 1):

```
dataMat,labelMat=loadDataSet()
thetas = gradAscent(dataMat,labelMat)
```

```
thetaArr = np.array(thetas)
dataArr = array(dataMat)
# get the number of data points
m = shape(dataArr)[0]
xcord1 = []; ycord1 = []
xcord2 = []; ycord2 = []
# classifying data points in two classes (0 and 1)
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])
# making a scatter plot
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')
for j in range(0,len(xcord1)):
            ax.annotate("1", (xcord1[j],ycord1[j]))
ax.scatter(xcord2, ycord2, s=30, c='green')
for k in range(0,len(xcord2)):
            ax.annotate("0", (xcord2[k],ycord2[k]))
```

Next, to make a line that separates the different classes of data (0 and 1), we must calculate x_i (i = 1,2,3) by solving the followinng the equation:

$$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 = 0$$

where $x_1 = 1$ (our assumption), x_2 is given, and so x_3:

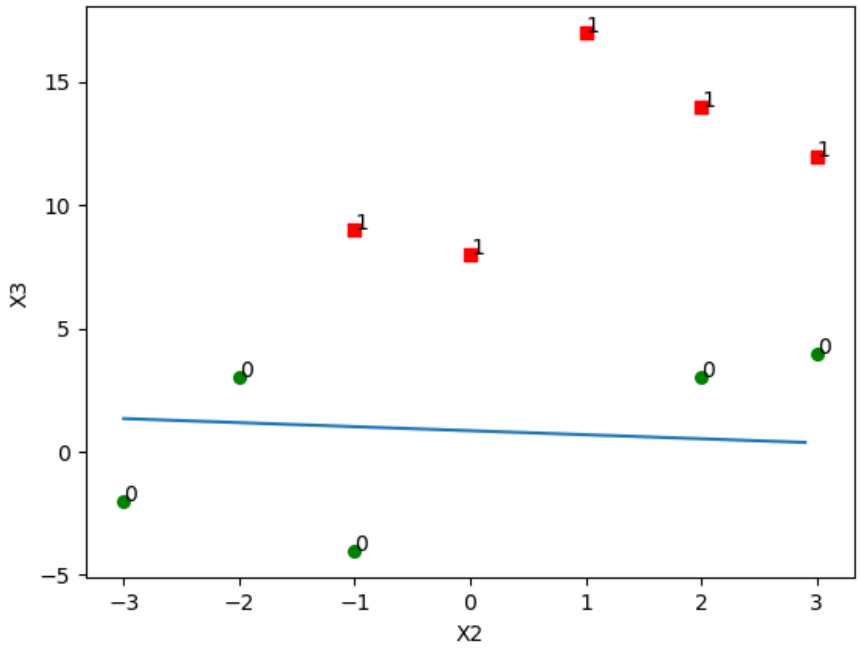
$$x_3 = \frac{-\theta_1 - \theta_2 x_2}{\theta_3}$$

Our the lines of code will look like this:

```
# X2 is given
```

```
X2 = arange(-3.0, 3.0, 0.1)
# calculating X3
X3 = (-thetaArr[0]-thetaArr[1]*X2)/thetaArr[2]
ax.plot(X2, X3)
plt.xlabel('X2'); plt.ylabel('X3');
plt.show()
```

The result:



As you see, the best-fit line is not a good separator of the data. We will improve this by using **stochastic gradient ascent**. Stochastic gradient ascent is an example of an online learning algorithm. This is known as online because we can incrementally update the classifier as new data comes in rather than all at once.

We will modify the **gradAscent**() function with the following notes:

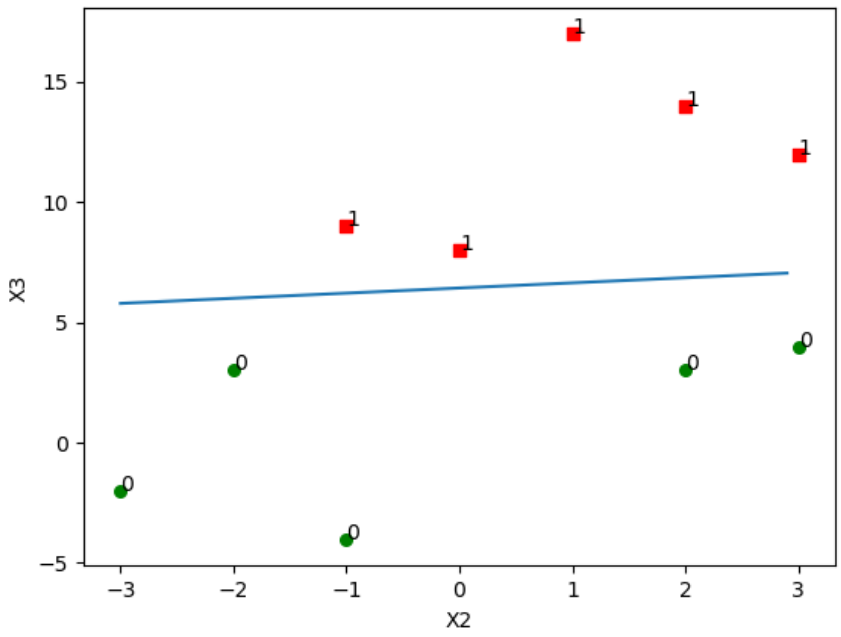
- update the thetas using only one instance at a time
- the variables **sig** and **error** are now single values rather than vectors
- **eta** changes on each iteration
- select randomly each instance to use in updating the thetas

The modified version of the **gradAscent**() function called the **stocGradAscent**() can look like this:

```
def stocGradAscent(dataMat, labelMat, numIter):
```

```
    dataMatrix = array(dataMat)
    m,n = shape(dataMatrix)
    thetas= ones(n)
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            eta = 4/(1.0+j+i)+0.01
            randIndex = int(random.uniform(0,len(dataIndex)))
            sig = sigmoid(sum(dataMatrix[randIndex]*thetas))
            error = labelMat[randIndex] - sig
            thetas = thetas + eta * error * dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return thetas
```

So far, the result will:



Points of interest

In this tip, I introduced about logistic regression – the algorithm find best-fit parameters to a nonlinear function called the sigmoid. I also introduced gradient ascent and its modified version stochastic gradient ascent. You can view all of source code in this tip here (https://github.com/TranNgocMinh/Machine-Learning-and-Tensorflow/blob/master/LogisticRegression/Readme.md).

References

https://support.sas.com/rnd/app/stat/papers/logistic.pdf

https://web.stanford.edu/~jurafsky/slp3/7.pdf

http://cs229.stanford.edu/notes/cs229-notes1.pdf

https://math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/grad/grad.html

https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/the-gradient

http://cs.wellesley.edu/~sravana/ml/logisticregression.pdf

https://web.stanford.edu/class/archive/cs/cs109/cs109.1178/lectureHandouts/220-logistic-regression.pdf

https://web.stanford.edu/class/archive/cs/cs109/cs109.1176/lectures/23-LogisticRegression.pdf