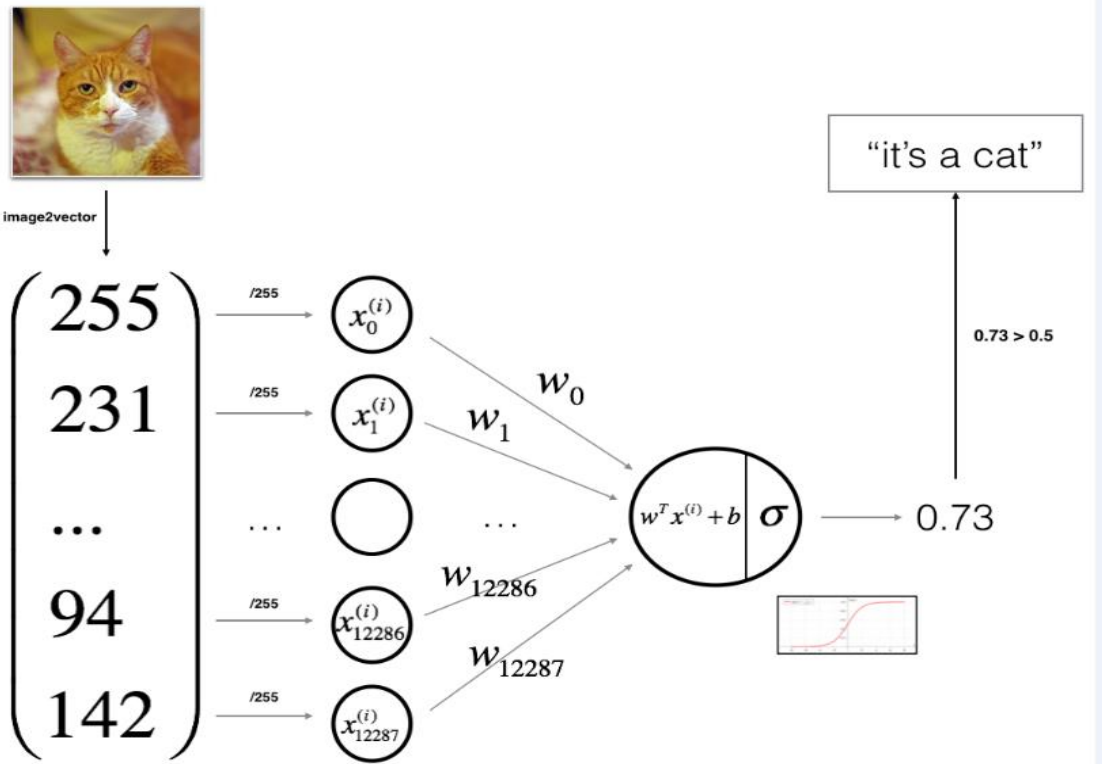
Logistic Regression

Author: 贺志尧
Email:2282815808@qq.com

1- Logistic Regression

1）General Architecture of Logistic Regression

The following Figure explains why **Logistic Regression is actually a very simple Neural Network!**



Mathematical expression of the algorithm:

For one example $x^{(i)}$：

$$z^{(i)} = w^T x^{(i)} + b \tag{1}$$

$$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)}) \tag{2}$$

$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1-y^{(i)}) \log(1-a^{(i)}) \tag{3}$$

The cost is then computed by summing over all training examples:

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)}) \tag{4}$$

For one example $X = [x^{(1)}, x^{(2)}, x^{(3)} \ldots x^{(m)}]$：

$$Z = w^T X + b \tag{5}$$

$$A = \sigma(w^T X + b) = \sigma(Z) = (a^{(0)}, a^{(1)}, ..., a^{(m-1)}, a^{(m)}) \tag{6}$$

The cost is then computed by summing over all training examples:

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(a^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)})] \tag{7}$$

2）The main steps for building a Neural Network

(1) Define the model structure (such as number of input features)
(2) Initialize the model's parameters
(3) Learn the parameters for the model by minimizing the cost :

Loop:

a) Calculate current cost (forward propagation)
b) Calculate current gradient (backward propagation)
c) Update parameters (gradient descent)

(4) Use the learned parameters to make predictions (on the test set)
(5) Analyse the results and conclude

3）Detailed steps for building a Neural Network

(1) Define the model structure (such as number of input features)

Common steps for pre-processing a new dataset are:

- Figure out the dimensions and shapes of the problem (m_train, m_test, num_px, ...)
- Reshape the datasets such that each example is now a vector of size (num_px * num_px * 3, 1)
- "Standardize" the data (train_set_x = train_set_x_flatten/255 , test_set_x = test_set_x_flatten/255.)

(2) Initialize the model's parameters

```
### START CODE HERE ### (≈ 1 line of code)

    w = np.zeros((dim,1))

    b = 0

### END CODE HERE ###
```

(3) Learn the parameters for the model by minimizing the cost

a) Calculate current cost (forward propagation):

- You get X

- You compute $A = \sigma(w^T X + b) = (a^{(0)}, a^{(1)}, ..., a^{(m-1)}, a^{(m)})$

- You calculate the cost function: $J = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})$

【注释】详见附录：Explanation of logistic regression cost function

【code】A = sigmoid(np.add(np.dot(w.T, X), b))   # compute activation

【code】cost = -(np.dot(Y, np.log(A).T) + np.dot(1 - Y, np.log(1 - A).T)) / m   # compute cost

b) Calculate current gradient (backward propagation):

$$\frac{\partial J}{\partial w} = \frac{1}{m} X (A - Y)^T \tag{8}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)}) \tag{9}$$

【注释】详见附录：Logistic Regression Gradient Descent

【code】

```
### START CODE HERE ### (≈ 2 lines of code)
    dw = np.dot(X, (A - Y).T) / m
    db = np.sum(A - Y) / m
### END CODE HERE ###
```

c) Update parameters (gradient descent)

The goal is to learn w and b by minimizing the cost function J. For a parameter θ, the update rule is $\theta = \theta - \alpha\ d\theta$, where α is the learning rate.

$$w = w - \alpha\ dw \tag{10}$$

$$b = b - \alpha\ db \tag{11}$$

【code】

```
### START CODE HERE ###

w = w- learning_rate*dw

b = b- learning_rate*db

### END CODE HERE ###
```

(4) Use the learned parameters to make predictions (on the test set)

a) Calculate $\hat{Y} = A = \sigma(w^T X + b)$

b) Convert the entries of a into 0 (if activation <= 0.5) or 1 (if activation > 0.5), stores the predictions in a vector `Y_prediction`.

【code】

```
### START CODE HERE ###
  m = X.shape[1]     #样本数
  Y_prediction = np.zeros((1,m))
  w = w.reshape(X.shape[0], 1)

  # Compute vector "A" predicting the probabilities of a cat being present in the picture
    A =   sigmoid(np.add(np.dot(w.T, X), b))   #(1,m)

  for i in range(A.shape[1]):
      # Convert probabilities A[0,i] to actual predictions p[0,i]
      if A[0,i]<=0.5:
            Y_prediction[0,i]= 0
      else:
            Y_prediction[0,i]= 1
 ### END CODE HERE ###
```

(5) Analyse the results and conclude

2- Appendix

1) Explanation of logistic regression cost function

（1）在 logistic 回归中，需要预测的结果 $\hat{y}$，可以表示为$\hat{y} = \sigma(w^T x + b)$， σ 是我们熟悉的函数， $\sigma(z) = \frac{1}{1+e^{-z}}$。

（2）我们约定 $\hat{y}$=P(y = 1 ｜ x)，即算法的输出 $\hat{y}$ 是给定训练样本 x 条件下，y 等于 1 的概率。换句话说，

如果 y=1，那么在给定 x 得到 y=1 的概率等于 $\hat{y}$，

反过来说，

如果 y=0，在给定 x 得到 y=0 的概率是 1-$\hat{y}$，

因此 $\hat{y}$ 表示的是 y=1 的概率，那么 1-$\hat{y}$ 就是 y=0 的概率。即：

If y=1：$P(y|x) = \hat{y}$

If y=0：$P(y|x) = 1 - \hat{y}$

需要指出的是，我们讨论的是二分分类问题的成本函数，因此 y 的取值只能是 0 或者 1，上述的两个条件概率公式可以合并成下面这样：

$$P(y|x) = \hat{y}^{y}(1-y)^{(1-y)}$$

我们需要最大化 P(y|x)。因为 y=0 时，对应的样本被分到 0 标签对应的类；y=1 时，对应的样本被分到 1 标签对应的类。无论样本被分到哪一类，都希望 P(y|x)最大。

由于 log 函数是严格单调递增的函数，最大化$\log(P(y|x))$ 等价于最大化 P(y|x)，即最大化：

$$\log\big(P(y|x)\big) = \log \hat{y}^{y}(1-y)^{(1-y)}$$

化简：

$$y\log\hat{y} + (1-y)\log(1-\hat{y})$$

当你训练学习算法时，希望算法输出值的概率是最大的，然而在 logistic 回归中，我们需要最小化损失函数（Loss function），因此损失函数定义为：

$$L(\hat{y}, y) = -\log\big(P(y|x)\big) = -[y log\hat{y} + (1-y)\log(1-\hat{y})]$$

前面有一个负号的原因是，当你训练学习算法，算法输出值的概率是最大时，在 logistic 回归中，损失函数就是最小的。因此这就是单个训练样本的损失函数表达式。

那么 m 个训练样本的总体成本函数(Cost function)如何表示?

假设所有的训练样本服从同一分布且相互独立，也即独立同分布的，所有这些样本的联合概率就是每个样本概率的乘积，即：

$$P(\text{labels in training set}) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$$

我们需要寻找一组参数，使得给定样本的观测值概率最大，即，想做最大似然估计。令这个概率最大化等价于令其对数最大化，在等式两边取对数：

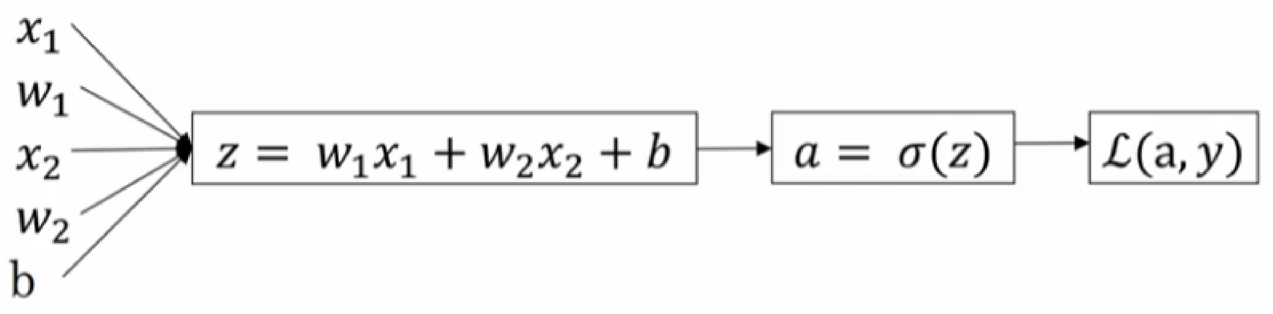
$$\log P(\text{labels in training set}) = \log \prod_{i=1}^{m} P(y^{(i)}|x^{(i)}) = \sum_{i=1}^{m} log P(y^{(i)}|x^{(i)}) = \sum_{i=1}^{m} -L(\hat{y}^{(i)}, y^{(i)})$$

由于训练模型时，目标是让成本函数最小化，所以我们不是直接用最大似然概率，要去掉这里的负号。最后为了方便，可以对成本函数进行适当的缩放，我们就在前面加一个额外的常数因子$\frac{1}{m}$，得到个训练样本的成本函数 J:

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L\left(\hat{y}^{(i)},y^{(i)}\right)$$

2）Logistic Regression Gradient Descent

对单个样本：



假设样本只有两个特征 x_1 和 x_2，为了计算 z，我们需要输入参数 w_1、w_2 和 b，除此之外还有特征值 x_1 和 x_2。因此 z 的计算公式为：

$$z = w_1 * x_1 + w_2 * x_2 + b$$

回想一下逻辑回归的公式定义如下：

$$\hat{y} = a = \sigma(z)$$

其中:

$$z = w^T x + b$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

单个样本的损失函数定义如下：

$$L(a,y) = -(y\log(a) + (1-y)\log(1-a))$$

其中 a 是逻辑回归的输出， y 是样本的标签值。

根据梯度下降法， w 和 b 的修正量可以表达如下：

$$w = w - \alpha\frac{\partial J(w,b)}{\partial w} \quad , \quad b = b - \alpha\frac{\partial J(w,b)}{\partial b}$$

因为我们想要计算出的代价函数 $L(a,y)$ 的导数。根据计算图，首先我们需要反向计算出代价函数 $L(a,y)$ 关于 a 的导数，在编写代码时，你只需要用 da 来表示 $\frac{dL(a,y)}{da}$ 。

通过微积分得到：

$$da = dL(a,y)/da = -y/a + (1-y)/(1-a)$$

现在可以再反向一步，计算 dz ，即代价函数 L 关于 z 的导数 $\frac{dL}{dz}$ ，也可以写成 $\frac{dL(a,y)}{dz}$ 。

因为:

$$dz = \frac{dL(a,y)}{dz} = \frac{dL}{dz} = (\frac{dL}{da})*(\frac{da}{dz}),$$

并且

$$\frac{da}{dz} = (\frac{1}{1+e^{-z}})' = \frac{-(-e^{z})}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}(\frac{1-1+e^{z}}{1+e^{-z}}) = \frac{1}{1+e^{-z}}(1-\frac{1}{1+e^{-z}}) = a(1-a)$$

而

$$\frac{dL}{da} = (-y/a+(1-y)/(1-a)),$$

因此将这两项相乘

$$dz = \frac{dL(a,y)}{dz} = \frac{dL}{dz} = (\frac{dL}{da})*(\frac{da}{dz}) = a*(1-a)*(-y/a+(1-y)/(1-a)) = a-y$$

这个推导的过程就是链式法则。

现在进行最后一步反向推导，也就是计算 w 和 b 变化对代价函数 L 的影响，

$$dw_1 = \frac{\partial L}{\partial w_1} = x_1 \cdot dz$$

$$dw_2 = \frac{\partial L}{\partial w_2} = x_2 \cdot dz$$

$$db = dz$$

因此，关于单个样本的梯度下降算法，你所需要做的就是如下的事情：

首先使用以下公式计算 dz ，dw_1 ，dw_2 ，db ：

$$dz = (a-y)$$

$$dw_1 = x_1 \cdot dz$$

$$dw_2 = x_2 \cdot dz$$

$$db = dz$$

然后使用梯度下降算法更新 w_1 ，w_2 ，b ：

$$w_1 = w_1 - \alpha dw_1$$

$$w_2 = w_2 - \alpha dw_2$$

$$b = b - \alpha db$$

这就是关于单个样本实例的梯度下降算法中参数更新一次的步骤。

---华丽的分割线---

对于 m 个训练样本：

我们已经知道了怎么应用梯度下降在逻辑回归的一个训练样本上。现在我们想要把它应用在 m 个训练样本上

现在你已经知道了怎样计算导数，并且实现针对单个训练样本的逻辑回归的梯度下降算法。但是，训练逻辑回归模型不仅仅只有一个训练样本，而是有 m 个训练样本的整个训练集。现在我们门应用梯度下降在逻辑回归的 m 个训练样本上。

首先，让我们回忆下代价函数 J(w,b)的定义：

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(a^{(i)},y^{(i)})$$

其中 $a^{(i)}$ 是训练样本的预测值， $y^{(i)}$ 是训练样本。

从定义的式子可以看出，全局代价函数，实际上是 1 到 m 项各个损失函数的平均。所以它表明，全局代价函数对 w_1 的微分，也就是各样本对应的损失函数对 w_1 微分的平均。即：

对单个样本 $x^{(i)}$：

$$dz^{(i)}=(a^{(i)}-y^{(i)})$$

$$dw_1^{(i)}=x_1^{(i)}\cdot dz^{(i)}$$

$$dw_2^{(i)}=x_2^{(i)}\cdot dz^{(i)}$$

$$db^{(i)}=dz^{(i)}$$

对 m 个样本：

$$dz=\frac{1}{m}\sum_{i=1}^{m}(a^{(i)}-y^{(i)})$$

$$dw_1=\frac{1}{m}\sum_{i=1}^{m}x_1^{(i)}\cdot dz^{(i)}$$

$$dw_2=\frac{1}{m}\sum_{i=1}^{m}x_2^{(i)}\cdot dz^{(i)}$$

$$db=\frac{1}{m}\sum_{i=1}^{m}dz^{(i)}$$

然后使用梯度下降算法更新 w_1，w_2，b：

$$w_1 = w_1 \text{-} \alpha dw_1$$

$$w_2 = w_2 \text{-} \alpha dw_2$$

$$b = b \text{-} \alpha db$$

这就是关于 m 个样本实例的梯度下降算法中参数更新一次的步骤。

附：以下是 m 个样本实例的梯度下降算法中参数更新 m 次的代码

```
J=0; dw1=0; dw2=0; db=0;
for i = 1 to m
    z(i) = wx(i)+b;
    a(i) = sigmoid(z(i));
    J += -[y(i)log(a(i))+(1-y(i)）log(1-a(i));
    dz(i) = a(i)-y(i);
    dw1 += x1(i)dz(i);
    dw2 += x2(i)dz(i);
    db += dz(i);
J /= m;
dw1 /= m;
dw2 /= m;
db /= m;
w=w-alpha*dw
b=b-alpha*db
```

References:

- http://www.cnblogs.com/hezhiyao/tag/Coursera%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0%E8%AF%BE%E7%A8%8B%E7%AC%94%E8%AE%B0/
- https://www.coursera.org/specializations/deep-learning