softmax 回归推导.

模型形式为 $h_\theta(x^{(i)}) = \begin{bmatrix} P(y^{(i)}=1 \mid x^{(i)};\theta) \\ P(y^{(i)}=2 \mid x^{(i)};\theta) \\ \cdots\cdots \\ P(y^{(i)}=k \mid x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \cdots\cdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$

代价函数为

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)}=j\}\cdot \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right]$$

直观语义就是使正确输出单元的激活值 $h_\theta(x^{(i)})_{y^{(i)}}$ 尽可能大

跟 LR 相似；LR 本质上就是二分类的 softmax. 详见 UFLDL.

求梯度. 注意 θ_j 是向量.

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial -\frac{1}{m}\sum_{i=1}^{m} \log \frac{e^{\theta_{y^{(i)}}^T x^{(i)}}}{\Sigma}}{\partial \theta_j}$$

$$= -\frac{1}{m}\cdot \frac{\partial \sum_{i=1}^{m} \theta_{y^{(i)}}^T x^{(i)} - \log \sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}{\partial \theta_j}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[1\{y^{(i)}=j\}\cdot x^{(i)} - \frac{1}{\Sigma}\cdot e^{\theta_j^T x^{(i)}}\cdot x^{(i)}\right]$$

$$= -\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left(1\{y^{(i)}=j\} - \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left(1\{y^{(i)}=j\} - h_\theta(x^{(i)})\right)$$

参考UFLDL. softmax是个过度参数化的模型，最优解不唯一. 可以通过 Weight Decay 解决

$$J(\theta) \triangleq -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)}=j\}\cdot\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_l e^{\theta_l^T x^{(i)}}}\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=1}^{n}\theta_{ij}^2$$

引入罚项后 $J(\theta)$ 变成严格凸函数，有唯一最优解. 且 Hessian 可逆.

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)}=j\} - h_\theta(x^{(i)})\right)\right] + \lambda\theta_j$$

代码如下：

```
class Softmax:
    # __init__(self, k=10) 省略
    def fit(self, x, y, n_epoch=200, λ=0.01, η=1e-4):
        m = x.shape[0]  样本个数
        self.θ = zeros((self.k, x.shape[1]))
        y = one-hot(y, range(self.k))
        for _ in range(n_epoch):  注意 n_epoch 不能用科学计数法, float
            h = self.predict_proba(x)
            grad = (h - y.T) @ x/m + λ * self.θ
            self.θ -= η * grad
    def predict_proba(self, x):
        return l1_norm(exp(self.θ @ x.T), axis=0)
            第i行j列表示第j个样本取值i的概率.
    def predict(self, x):
        return argmax(self.predict_proba(x), axix=0)
```

建议琢磨一下向量化的运算细节. 注意公式中的一画. 负号和除m缺一不可. 否则梯度爆炸. 出现 inf.

以上です.

koxo922@qq.com

2018.04.20