BDMI-课程编号-01510243

SaturnLab

iCenter 清华大学iCenter

# 大数据与机器智能

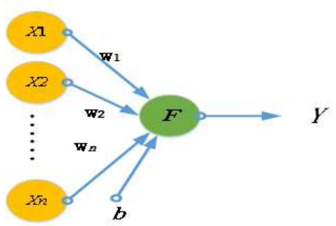
# 逻辑斯蒂回归实践

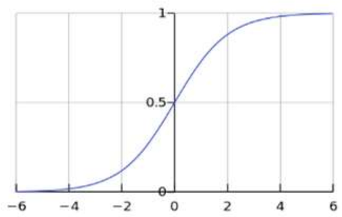
清华大学iCenter



## 人工神经元进行二元分类

- 具体方法采用逻辑斯蒂模型（logistic regression），输出Y：性别的概率
  - 取40人的数据，输入每个人的身体特征：X1，X2，X3，…，
  - 权重：W1，W2，W3，…，表示不同特征对判断结果的贡献率
- 如何找到合适的权重？经验调参法



# 想一想

- 单个人工神经元-判断性别-二元分类
- 如何进行**手动**调整权重参数?
- 经验法则:
  - 判断输入与结果是正相关或负相关，确定权重W是正还是负。
  - 判断输入与结果的重要程度，越重要，权重越大。

# 用逻辑斯蒂回归单元进行二元分类案例

- 读入数据
  - pandas+numpy

```
In [1]: import pandas as pd
        import numpy as np

In [2]: data = pd.read_excel('data.xlsx')

In [3]: data.head()
Out[3]:
```

| | Q1_性别 | Q2_身高（厘米） | Q3_体重 （公斤） | Q4_头发长度（厘米） |
|---|---|---|---|---|
| 0 | 男 | 190 | 70 | 7 |
| 1 | 女 | 160 | 45 | 20 |
| 2 | 男 | 179 | 61 | 5 |
| 3 | 女 | 173 | 60 | 50 |
| 4 | 男 | 175 | 70 | 15 |

# 用逻辑斯蒂回归单元进行二元分类案例

- 数据整理

```
In [4]: data = data.rename(columns={'Q1_性别': 'label',
                                    'Q2_身高（厘米）': 'height',
                                    'Q3_体重 （公斤）': 'weight',
                                    'Q4_头发长度（厘米）': 'hair'})

In [5]: data['label'] = data['label'].apply(lambda x : {'男': 0, '女': 1}[x])

In [6]: data.head()
Out[6]:
```

| | label | height | weight | hair |
|---|---|---|---|---|
| 0 | 0 | 190 | 70 | 7 |
| 1 | 1 | 160 | 45 | 20 |
| 2 | 0 | 179 | 61 | 5 |
| 3 | 1 | 173 | 60 | 50 |
| 4 | 0 | 175 | 70 | 15 |

# 用逻辑斯蒂回归单元进行二元分类案例

- 数据整理
  - 归一化

```
In [7]: features = data[['height', 'weight', 'hair']].to_numpy()

In [8]: mean = np.mean(features, axis=0)
        std = np.std(features, axis=0)

In [9]: features = (features - mean)/std

In [10]: label = data['label'].to_numpy()
```

# 用逻辑斯蒂回归单元进行二元分类案例

## 特征

```
In [11]: features
Out[11]: array([[ 9.86969285e-01, -2.25614423e-03, -4.63916995e-01],
       [-7.03854590e-01, -8.10707828e-01, -1.44545566e-01],
       [ 3.67000531e-01, -2.93298750e-01, -5.13051061e-01],
       [ 2.88357560e-02, -3.25636818e-01,  5.92465423e-01],
       [ 1.41557348e-01, -2.25614423e-03, -2.67380731e-01],
       [-5.91132998e-01, -5.52003289e-01,  1.01124764e-01],
       [-1.40246631e-01,  3.21124529e-01, -5.13051061e-01],
       [-2.52968223e-01, -4.87327154e-01, -5.13051061e-01],
       [ 1.41557348e-01, -1.63946481e-01, -5.37618094e-01],
       [ 1.41557348e-01, -2.25614423e-03, -5.37618094e-01],
       [-8.16576182e-01, -7.46031693e-01,  5.92465423e-01],
       [ 4.23361327e-01, -6.69322789e-02, -5.62185127e-01],
       [ 4.23361327e-01,  1.59434192e-01, -5.13051061e-01],
       [-7.03854590e-01, -6.81355558e-01, -1.44545566e-01],
       [-3.09329019e-01, -3.25636818e-01, -3.41081830e-01],
       [-7.03854590e-01, -6.81355558e-01,  3.46795093e-01],
       [ 1.41557348e-01, -1.31608414e-01, -2.67380731e-01],
       [ 8.74247694e-01, -2.25614423e-03, -2.67380731e-01],
       [-8.38858357e-02, -2.28622616e-01, -3.90215896e-01],
```

## 标签

```
In [12]: label
Out[12]: array([0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0])
```

# 用逻辑斯蒂回归单元进行二元分类案例

- 激活函数（计算预估概率）

```
In [24]: def sigmoid(scores):
             return 1 / (1 + np.exp(-scores))
```

$$p(x;b,w)=\frac{e^{\beta_0+x\cdot\beta}}{1+e^{\beta_0+x\cdot\beta}}=\frac{1}{1+e^{-(\beta_0+x\cdot\beta)}}$$

- 计算对数似然

$$L(\beta_0,\beta)=\prod_{i=1}^{n}p(x_i)^{y_i}(1-p(x_i))^{1-y_i}$$

$$\begin{aligned}\ell(\beta_0,\beta) &= \sum_{i=1}^{n}y_i\log p(x_i)+(1-y_i)\log 1-p(x_i)\\ &= \sum_{i=1}^{n}\log 1-p(x_i)+\sum_{i=1}^{n}y_i\log\frac{p(x_i)}{1-p(x_i)}\\ &= \sum_{i=1}^{n}\log 1-p(x_i)+\sum_{i=1}^{n}y_i(\beta_0+x_i\cdot\beta)\\ &= \sum_{i=1}^{n}-\log 1+e^{\beta_0+x_i\cdot\beta}+\sum_{i=1}^{n}y_i(\beta_0+x_i\cdot\beta)\end{aligned}$$

# 用逻辑斯蒂回归单元进行二元分类案例

$$\begin{aligned}\ell(\beta_0,\beta) &= \sum_{i=1}^{n} y_i \log p(x_i) + (1-y_i)\log 1 - p(x_i) \\ &= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1-p(x_i)} \\ &= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta) \\ &= \sum_{i=1}^{n} -\log 1 + e^{\beta_0 + x_i \cdot \beta} + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta)\end{aligned}$$

- 损失函数：负对数似然函数

### Calculating the Log-Likelihood

The log-likelihood can be viewed as as sum over all the training data. Mathematically,

$$ll = \sum_{i=1}^{N} y_i \beta^T x_i - log(1 + e^{\beta^T x_i})$$

where $y$ is the target class, $x_i$ represents an individual data point, and $\beta$ is the weights vector.

I can easily turn that into a function and take advantage of matrix algebra.

```
def log_likelihood(features, target, weights):
    scores = np.dot(features, weights)
    ll = np.sum( target*scores - np.log(1 + np.exp(scores)) )
    return ll
```

# 用逻辑斯蒂回归单元进行二元分类案例

- 计算梯度

$$ll = \sum_{i=1}^{N} y_i \beta^T x_i - log(1 + e^{\beta^T x_i})$$

$$\begin{aligned}\frac{\partial \ell}{\partial \beta_j} &= -\sum_{i=1}^{n} \frac{1}{1 + e^{\beta_0 + x_i \cdot \beta}} e^{\beta_0 + x_i \cdot \beta} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=1}^{n} (y_i - p(x_i; \beta_0, \beta)) x_{ij}\end{aligned}$$

$$\nabla ll = X^T(Y - Predictions)$$

# 用逻辑斯蒂回归单元进行二元分类案例

- 逻辑斯蒂回归（batch 梯度下降）

```
In [29]: def logistic_regression(features, target, num_steps, learning_rate, add_intercept = False):
             if add_intercept:
                 intercept = np.ones((features.shape[0], 1))
                 features = np.hstack((intercept, features))

             weights = np.zeros(features.shape[1])

             for step in range(num_steps):
                 scores = np.dot(features, weights)
                 predictions = sigmoid(scores)

                 # Update weights with log likelihood gradient
                 output_error_signal = target - predictions

                 gradient = np.dot(features.T, output_error_signal)
                 weights += learning_rate * gradient

                 # Print log-likelihood every so often
                 if step % 10000 == 0:
                     print(log_likelihood(features, target, weights))

             return weights
```

# 用逻辑斯蒂回归单元进行二元分类案例

- 训练以及权重

```
In [16]: weights = logistic_regression(features, label,
                                       num_steps = 50000, learning_rate = 5e-5, add_intercept=True)

         -29.794300659677482
         -12.17665666438134
         -10.368615637493027
         -9.699007253564737
         -9.355548233950786

In [17]: print(weights)

         [-1.62788588 -3.1418227  -2.31358577  2.1596935 ]
```

bias　身高　体重　头发长度

# 用逻辑斯蒂回归单元进行二元分类案例

- 预测
  - 同学1：身高、体重、头发长度

```
In [38]: student1 = np.array([[188, 85, 2]])
         print(predict(student1, weights))

         [0.00115921]
```

  - 同学2：身高、体重、头发长度

```
In [41]: student2 = np.array([[165, 50, 25]])
         print(predict(student2, weights))

         [0.76002054]
```

```
def predict(features, weights):
    global mean
    global std
    features = (features - mean)/std
    intercept = np.ones((features.shape[0], 1))
    features = np.hstack((intercept, features))
    scores = np.dot(features, weights)
    predictions = sigmoid(scores)

    return predictions
```

谢谢指正！