机器学习 **Lecture 2**

广义线性模型

一、 introduction

Defination 1 (广义线性模型)**.**

$$
\begin{aligned}
y &= g^{-1}(\mathbf{w}^\top \mathbf{x}) && (1)\\
\Leftrightarrow g(y) &= \mathbf{w}^\top \mathbf{x} && (2)
\end{aligned}
$$

预测使用（1）式，训练使用（2）式

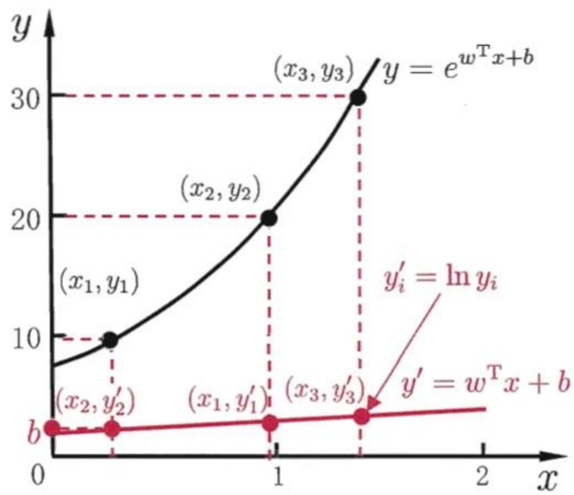

图 1: GLM

二、 Logistic Regression

1. 模型

Defination 2 (逻辑回归)**.**

$$
h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}),
$$
$$
\sigma(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}
$$

推导 1. 假设 y 服从伯努利分布，$p(+1|\mathbf{x}) = p_i$ 则有：

$$\begin{aligned} P(b_i|p_i) &= p_i^{b_i}(1-p_i)^{1-b_i} \\ &= \exp(b_i \ln p_i + (1-b_i)\ln(1-p_i)) \\ &= \exp(b_i \ln \frac{p_i}{1-p_i} + \ln(1-p_i)) \end{aligned}$$

由广义线性模型（指数族分布）知：

$$\mu_i = \ln \frac{p_i}{1-p_i} = \mathbf{w}^\top \mathbf{x}_i$$

带入得：

$$p_i = \frac{e^{\mu_i}}{1+e^{\mu_i}} = \sigma(\mu_i) \tag{3}$$

所以逻辑回归 $\Leftrightarrow \sigma(\mathbf{w}^\top \mathbf{x})$

2. 目标函数

Defination 3 (loss function for Logistic Regression)**.**

$$\min_{\mathbf{w}} \quad \frac{1}{m}\sum_{i=1}^{m} \ln(1+\exp(-y\mathbf{w}^\top \mathbf{x}_i))$$

推导 2.

由（3）式知：

$$\begin{aligned} p(+1|\mathbf{x}_i) &= \sigma(\mu_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i) \\ p(-1|\mathbf{x}_i) &= 1-\sigma(\mu_i) = -\sigma(\mu_i) = \sigma(-\mathbf{w}^\top \mathbf{x}_i) \end{aligned}$$

即：

$$p(y_i|\mathbf{x}_i) = \sigma(y\mathbf{w}^\top \mathbf{x}_i)$$

最大似然估计

$$\begin{aligned} \max \prod_{i=1}^{m} p(\mathbf{x}_i)p(y_i|\mathbf{x}_i) &= \max_{\mathbf{w}} \prod_{i=1}^{m} p(\mathbf{x}_i)\sigma(y_i\mathbf{w}^\top \mathbf{x}_i) \\ &= \max_{\mathbf{w}} \prod_{i=1}^{m} \sigma(y_i\mathbf{w}^\top \mathbf{x}_i) \\ &= \max_{\mathbf{w}} \ln\{\prod_{i=1}^{m} \sigma(y_i\mathbf{w}^\top \mathbf{x}_i)\} \\ &= \max_{\mathbf{w}} \sum_{i=1}^{m} \ln \frac{1}{1+\exp(-y_i\mathbf{w}^\top \mathbf{x}_i)} \\ &= \min_{\mathbf{w}} \sum_{i=1}^{m} \ln(1+\exp(-y_i\mathbf{w}^\top \mathbf{x}_i)) \\ &= \min_{\mathbf{w}} \frac{1}{m}\sum_{i=1}^{m} \ln(1+\exp(-y_i\mathbf{w}^\top \mathbf{x}_i)) \end{aligned}$$

Theorem 1. 逻辑回归的损失函数是在概率为 σ 的情况下，交叉熵的二元形式

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \ln(1+\exp(-y\mathbf{w}^\top \mathbf{x_i})) \Leftrightarrow \min_{\mathbf{w}} -\frac{1}{m}\sum_{c=1}^{2}\sum_{i=1}^{m} y_{ic}\ln(p_{ic})$$

证明.

$$\begin{aligned}
-\frac{1}{m}\sum_{c=1}^{2}\sum_{i=1}^{m} y_{ic}\ln(p_{ic}) &= -\frac{1}{m}\sum_{i=1}^{m}[y_{ic}=1]\ln p(+1|\mathbf{x}_i) + [y_{ic}=-1]\ln p(-1|\mathbf{x}_i) \\
&= -\frac{1}{m}\sum_{i=1}^{m}[y_{ic}=1]\ln \sigma(\mathbf{w}^\top\mathbf{x}_i) + [y_{ic}=-1]\ln \sigma(-\mathbf{w}^\top\mathbf{x}_i) \\
&= -\frac{1}{m}\sum_{i=1}^{m}\ln\sigma(y_i\mathbf{w}^\top\mathbf{x}_i) \\
&= \frac{1}{m}\sum_{i=1}^{m}\ln(1+\exp(-y_i\mathbf{w}^\top\mathbf{x}_i))
\end{aligned}$$

□

3. 迭代方法

采用随机梯度下降法

$$\begin{aligned}
g &= \nabla_{i,\mathbf{w}} L(\mathbf{w},\mathbf{x},y) \\
&= \frac{\exp(-y_i\mathbf{w}^\top\mathbf{x}_i)(-y_i\mathbf{x}_i)}{1+\exp(-y_i\mathbf{w}^\top\mathbf{x}_i)} \\
&= \sigma(-y_i\mathbf{w}^\top\mathbf{x}_i)(-y_i\mathbf{x}_i)
\end{aligned}$$

4. 算法

Algorithm 1 Logistic Regression with SGD

Input: $\mathbf{x}, y, \alpha_0, T, \mathbf{w} = \mathbf{0}$

repeat
  随机在 $[1, m]$ 中选取一个 idx
  compute $g = \sigma(-y_i\mathbf{w}^\top\mathbf{x}_i)(-y_i\mathbf{x}_i)$
  update $\mathbf{w} \leftarrow \mathbf{w} - \alpha_t g$
until step $\geq T$

5. 运行结果

在鸢尾花数据集的前两个维度上运行逻辑回归，精度为 99.33%，其分类结果如图2所示，损失随步数变化如图3所示。在鸢尾花数据集的所有四个维度上运行逻辑回归，精度为 100%，模型及参数如下

$$p = \sigma(-4.23x_1 - 11.07x_2 + 15.81x_3 7.20x_4 - 2.18)$$

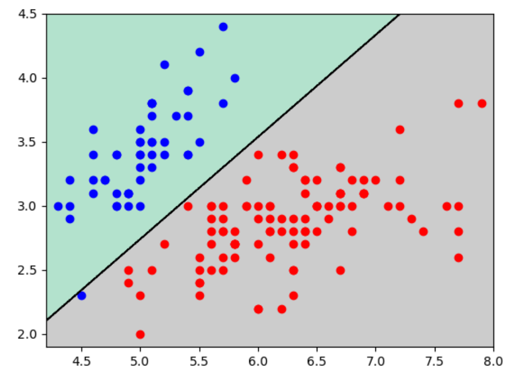

图 2: 分类结果

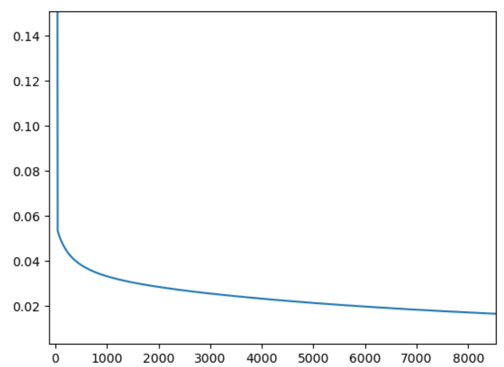

图 3: 损失随步数变化

6. 代码

```
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
class LogisticRegression:
    def __init__(self, X, y, lr):
        '''
        :param X: shape ---> (m,n) m个数据 n个参数
        :param y: shape ---> m
        '''
        self.y = y
        self.m = np.shape(X)[0]
        self.n = np.shape(X)[1] + 1
        self.X = np.append(X, np.ones(self.m).reshape(self.m,1), axis=1)
        self.w = np.zeros(self.n)
        self.lr = lr
        self.history = []
```

```

    def loss(self):
        return np.sum(np.log(1+np.exp(-self.y * self.X.dot(self.w))))/self.m

    def train(self):
        while True:
            gd = np.zeros(self.n)
            for i in range(self.m):
                gd += self.sigmoid(-self.y[i]*self.X[i].dot(self.w))*(-self.y[i]*self.X[i])
            gd = gd / self.m
            self.w -= self.lr*gd
            loss = self.loss()
            print(loss,gd)
            self.history.append(loss)
            print('# ---------------------------------------------------- #')
            if norm(gd, 2) < 1e-3:
                print(self.w)
                print(self.sigmoid(self.X.dot(self.w)))
                break
        return self.w
    def sigmoid(self, x):
        return 1/(1 + np.exp(-x))
    def plot_decision_boundaries(self, resolustion=1000):

        # 取这个用来画网格的
        mins = self.X.min(axis=0) - 0.1
        maxs = self.X.max(axis=0) + 0.1
        xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolustion),
                             np.linspace(mins[1], maxs[1], resolustion))
        grid = np.c_[xx.ravel(), yy.ravel()]
        predict = []
        for i in grid:
            predict.append(self.predict(i))
        predict = np.array(predict)
        predict = predict.reshape(xx.shape)
        # print(predict[:,0])
        plt.contourf(predict, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                     cmap='Pastel2')

        plt.contour(predict, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                    linewidths=1, colors='k')
        plt.scatter(self.X[:50, 0], self.X[:50, 1], c='b')
        plt.scatter(self.X[50:, 0],self.X[50:, 1], c='r')
        plt.show()

    def predict(self, x):
        return np.sign(self.sigmoid(np.hstack([x, 1]).dot(self.w)) - 0.5)

    def acc(self, x,y):
        cnt = 0
```

```python
        for i in range(x.shape[0]):
            predict = self.predict(x[i,:])
            if predict == y[i]:
                cnt+=1
        return cnt/x.shape[0]
```