逻辑回归实战

1. 理论快速回顾

1.1 模型函数

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

其中：$z = \theta^T x$

一般而言，当 y>0.5 时，z 被归类为真（True）或阳性（Positive），否则当 y<=0.5 时，z 被归类为假（False）或阴性（Negative）。

所以，在模型输出预测结果时，不必输出 y 的具体取值，而是根据上述判别标准，输出1（真）或0（假）。

1.2 损失函数

训练逻辑回归函数，我们已知了样本点(x,y)，我们的目的是求出一组参数θ。

模型函数y=1表示样本点为阳性，故而我们可以得到：
$P(y=1|x) = h(x); P(y=0|x) = 1 - h(x)$

对于这个二项分布，我们有：
$P(y|x) = h(x)^y (1-h(x))^{(1-y)}$

对于m个数据，有：

$$L(\theta) = \frac{1}{m}\prod_{i=1}^{m} P(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m}(h_\theta(x^{(i)}))^{y^{(i)}}(1-h_\theta(x^{(i)}))^{(1-y^{(i)})}$$

为了方便计算，取负对数得到：

$$J(\theta) = -\frac{1}{m}\log(L(\theta)) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

1.3 优化算法：梯度下降法

求导推导不展开，结果是：

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^{(i)}) - y^{(i)}]x_j^{(i)}; \quad j = 1, 2, ..., n$$

我们将之向量化以后：

$$\nabla J(\theta) = \frac{1}{m}X^T(h(X\theta) - y)$$

其中：h是sigmoid函数

2. 实战

以下所有代码，若要测试运行请参考提供python源文件。

2.1 首先实现一个自己的逻辑回归类

2.1.1 sigmoid函数的实现

```
def sigmoid(self, t):
    return 1. / (1. + np.exp(-t))
```

2.1.2 实现一个训练进行拟合的函数：

其中J函数表示损失函数，dJ表示损失函数求导后的形式，gradient_descent表示梯度下降函数。

```
def fit(self, X_train, y_train, eta=0.01, n_iters=1e4):
    """使用梯度下降法训练logistic Regression模型"""
    assert X_train.shape[0] == y_train.shape[0], \
        "the size of X_train must be equal to the size of y_train"
    def J(theta, X_b, y):
        y_hat = self.sigmoid(X_b.dot(theta))
        try:
            return -np.sum(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat)) / len(y)
        except:
            return float('inf')
    def dJ(theta, X_b, y):
        # 向量化后的公式
        return X_b.T.dot(self.sigmoid(X_b.dot(theta)) - y) / len(y)
    def gradient_descent(X_b, y, initial_theta, eta, n_iters=1e4, epsilon=1e-8):
        theta = initial_theta
        cur_iter = 0
        while cur_iter < n_iters:
            gradient = dJ(theta, X_b, y)
            last_theta = theta
            theta = theta - eta * gradient
            if abs(J(theta, X_b, y) - J(last_theta, X_b, y)) < epsilon:
                break
            cur_iter += 1
        return theta
    X_b = np.hstack([np.ones((len(X_train), 1)), X_train])
    initial_theta = np.zeros(X_b.shape[1])
    self._theta = gradient_descent(X_b, y_train, initial_theta, eta, n_iters)
    # 截距
    self.intercept = self._theta[0]
    # x_i前的参数
    self.coef = self._theta[1:]
    return self
```

2.1.3 实现一个预测函数

其中需要注意，最终的预测函数predict返回的是0/1，而不是具体的概率。predict_proba函数中返回的是具体的概率。

```
def predict_proba(self, X_predict):
    """给定待预测数据集X_predict，返回表示X_predict的结果概率向量"""
    assert self.intercept is not None and self.coef is not None, \
        "must fit before predict"
    assert X_predict.shape[1] == len(self.coef), \
        "the feature number of X_predict must be equal to X_train"
    X_b = np.hstack([np.ones((len(X_predict), 1)), X_predict])
    return self.sigmoid(X_b.dot(self._theta))

def predict(self, X_predict):
    """给定待预测数据集X_predict，返回表示X_predict的结果向量"""
    assert self.intercept is not None and self.coef is not None, \
        "must fit before predict!"
    assert X_predict.shape[1] == len(self.coef), \
        "the feature number of X_predict must be equal to X_train"
    prob = self.predict_proba(X_predict)
    return np.array(prob >= 0.5, dtype='int')
```

2.1.4 效果评测函数

使用sklearn中的accuracy_score进行效果的评测。

```
def score(self, X_test, y_test):
    """根据测试数据集X_test和y_test确定当前模型的准确度"""
    y_predict = self.predict(X_test)
    return accuracy_score(y_test, y_predict)
```

2.2 小数据集效果测试

使用sklearn中提供的鸢尾花数据集的，并且为了便于查看只取特征只取一个维度数据，具体代码如下(复制代码请打开iris_lr_demo.py进行操作)：

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from LogisticRegression import LogisticRegression
# 取鸢尾花数据集
iris = datasets.load_iris()
X = iris.data
y = iris.target
# 筛选特征
X = X[y < 2, :2]
y = y[y < 2]
# 绘制出图像
plt.scatter(X[y == 0, 0], X[y == 0, 1], color="red")
plt.scatter(X[y == 1, 0], X[y == 1, 1], color="blue")
plt.show()
# 切分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=666)
# 调用我们自己的逻辑回归函数
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
print("final score is :%s" % log_reg.score(X_test, y_test))
print("actual prob is :")
print(log_reg.predict_proba(X_test))
```