Logistic regression

1. 实验目的

用来处理分类问题，预测当前的值属于哪个组

2. 算法原理

1)从线性回归到逻辑回归

我们知道，线性回归的模型是求出输出特征向量 Y 和输入样本矩阵 X 之间的线性关系系数 θ，满足 Y=Xθ。此时我们的 Y 是连续的，所以是回归模型。如果我们想要 Y 是离散的话，怎么办呢？一个可以想到的办法是，我们对于这个 Y 再做一次函数转换，变为 g(Y)。如果我们令 g(Y)的值在某个实数区间的时候是类别 A，在另一个实数区间的时候是类别 B，以此类推，就得到了一个分类模型。如果结果的类别只有两种，那么就是一个二元分类模型了。逻辑回归的出发点就是从这来的。下面我们开始引入二元逻辑回归。

2)二元逻辑回归

上一节我们提到对线性回归的结果做一个在函数 g 上的转换，可以变化为逻辑回归。这个函数 g 在逻辑回归中我们一般取为 sigmoid 函数，形式如下：

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

它有一个非常好的性质，即当 z 趋于正无穷时，g(z)趋于 1，而当 z 趋于负无穷时，g(z)趋于 0，这非常适合于我们的分类概率模型。另外，它还有一个很好的导数性质：

$$g'(z)=g(z)(1-g(z))$$

这个通过函数对 g(z)求导很容易得到，后面我们会用到这个式子。

如果我们令 g(z)中的 z 为：z=xθ，这样就得到了二元逻辑回归模型的一般形式：

$$h_\theta(x) = \frac{1}{1 + e^{-x\theta}}$$

其中 x 为样本输入，hθ(x)为模型输出，可以理解为某一分类的概率大小。而 θ 为分类模型的要求出的模型参数。对于模型输出 hθ(x，我们让它和我们的二元样本输出 y（假设为 0 和 1）有这样的对应关系，如果 hθ(x)>0.5hθ(x)>0.5，即 xθ>0, 则 y 为 1。如果 hθ(x)<0.5，即 xθ<0, 则 y 为 0。y=0.5 是临界情况，此时 xθ=0 为， 从逻辑回归模型本身无法确定分类。

hθ(x)的值越小，而分类为 0 的的概率越高，反之，值越大的话分类为 1 的的概率越高。如果靠近临界点，则分类准确率会下降。

此处我们也可以将模型写成矩阵模式：

$$h_\theta(X) = \frac{1}{1 + e^{-X\theta}}$$

其中 hθ(X)为模型输出，为 mx1 的维度。X 为样本特征矩阵，为 mxn 的维度。θ 为分类的模型系数，为 nx1 的向量。

理解了二元分类回归的模型，接着我们就要看模型的损失函数了，我们的目标是极小化损失函数来得到对应的模型系数 θ。

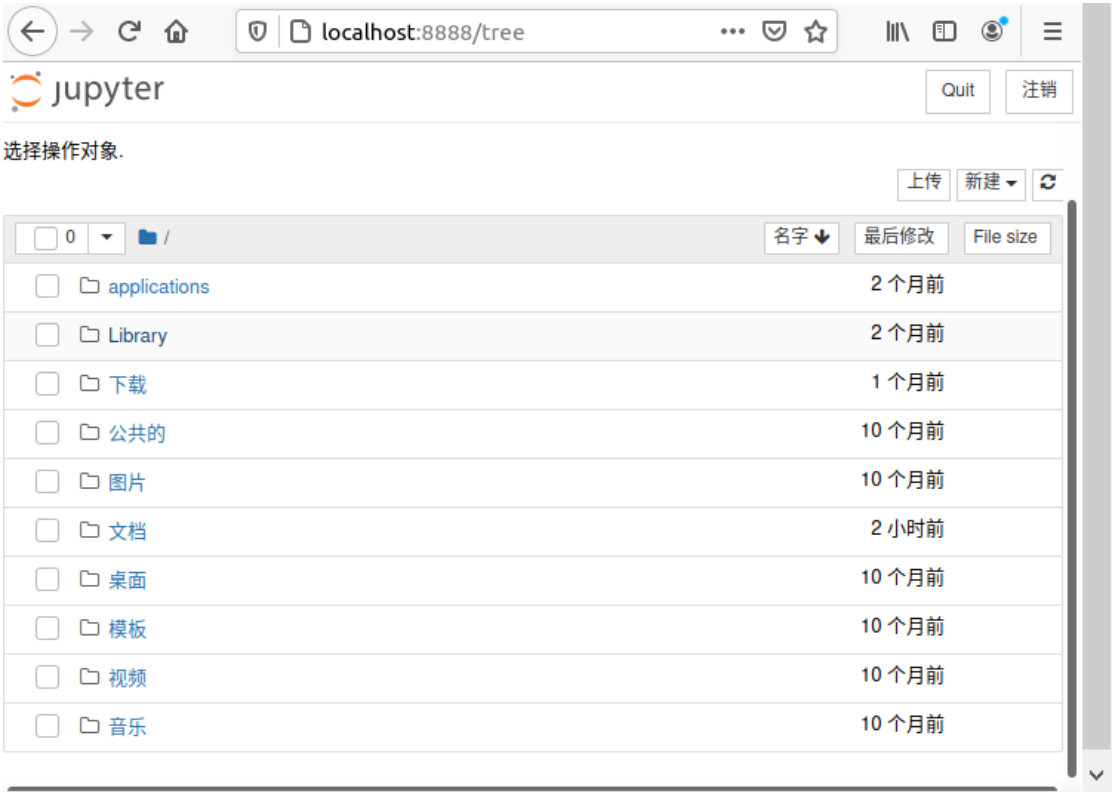
3. 实验环境

Ubuntu 20.04

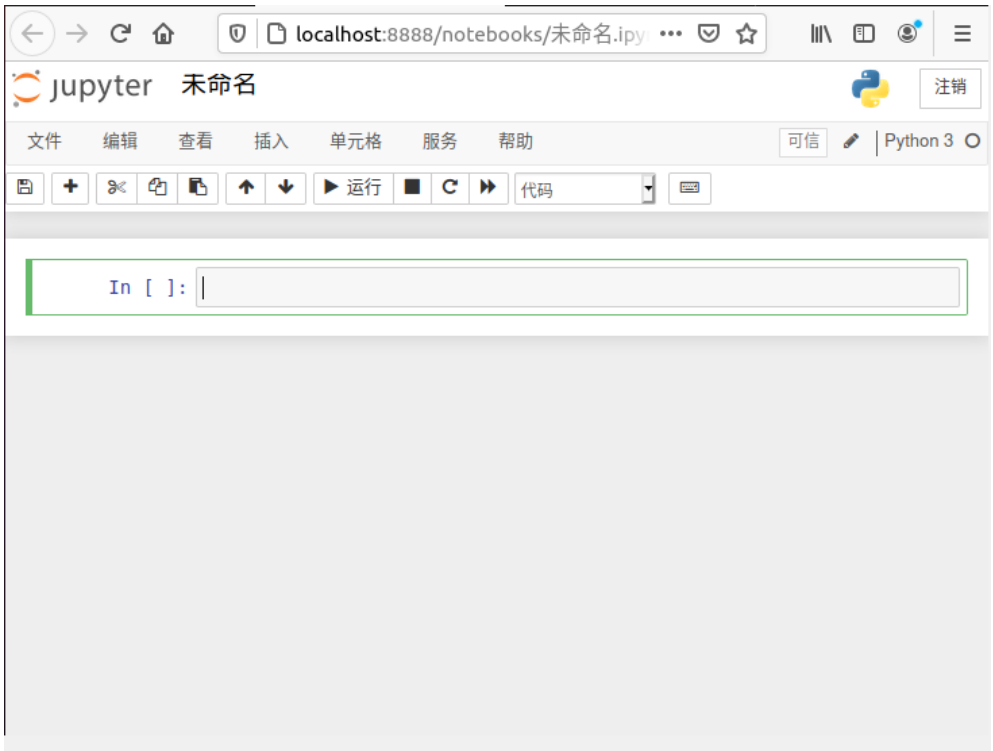
Python 3.6
Jupyter notebook

4. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面



2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件



5. 实操

Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 检查缺失数据
4. 分割数据集
5. 特征缩放

```
#导入库
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#导入数据集
dataset = pd.read_csv('Social_Network_Ads.csv')

#查看数据，检查是否缺失数据
X = dataset.iloc[:, [2, 3]].values     ## 选取 2，3 两列--Age+ salary
y = dataset.iloc[:, 4].values      ## 选取最后一列

#分割数据集
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 0)

#特征缩放
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Step 2:逻辑回归模型

- 逻辑回归应用于分类好的数据集

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

Step 3:预测

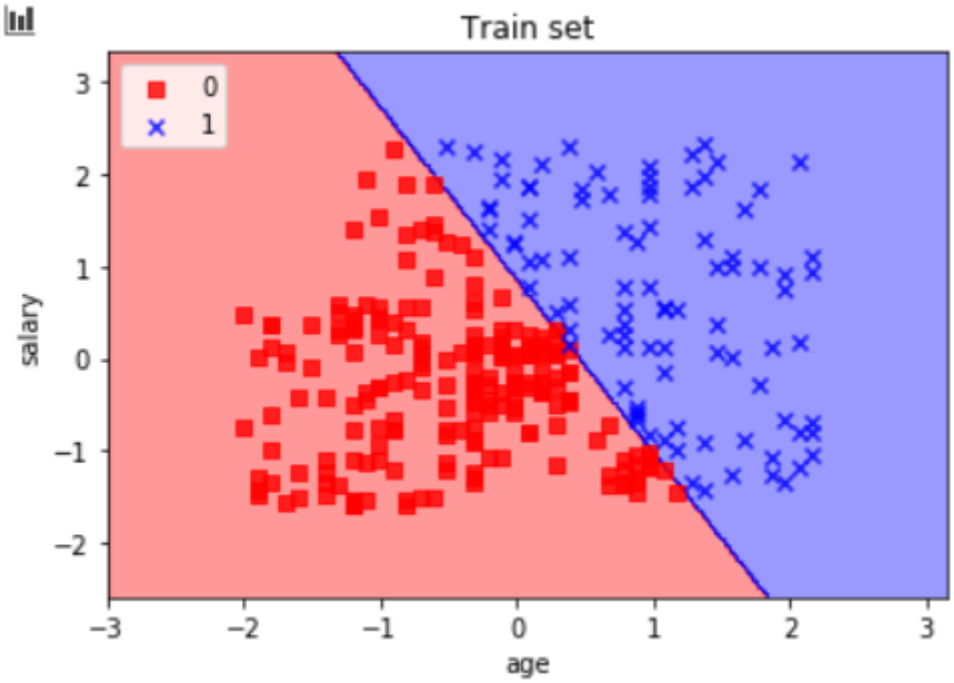
```
y_pred = classifier.predict(X_test)
```

Step 4：可视化

```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0
.02):
    # setup marker generator and color map
```

```
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution), np.ar
ange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8, c=cmap(
idx),marker=markers[idx], label=cl)
    # highlight test samples
    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], c='', alpha=1.0, linewi
dth=1, marker='o', s=55, label='test set')


plot_decision_regions(X_train, y_pred, classifier=classifier)
plt.xlabel('age')
plt.ylabel('salary')
plt.legend(loc='upper left')
plt.title("Train set")
plt.show()
```




```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0
.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution), np.ar
ange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8, c=cmap(
idx),marker=markers[idx], label=cl)
    # highlight test samples
    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], c='', alpha=1.0, linewi
dth=1, marker='o', s=55, label='test set')
```

```
plot_decision_regions(X_test, y_pred, classifier=classifier)
plt.xlabel('age')
plt.ylabel('salary')
plt.legend(loc='upper left')
plt.title("Test set")
plt.show()
```