第 5 章 Logistic 回归

内容提要

- Logistic 回归分类器
- 梯度下降最优化算法
- 最优化理论初步
- 多类别分类

5.1 Logistic 回归

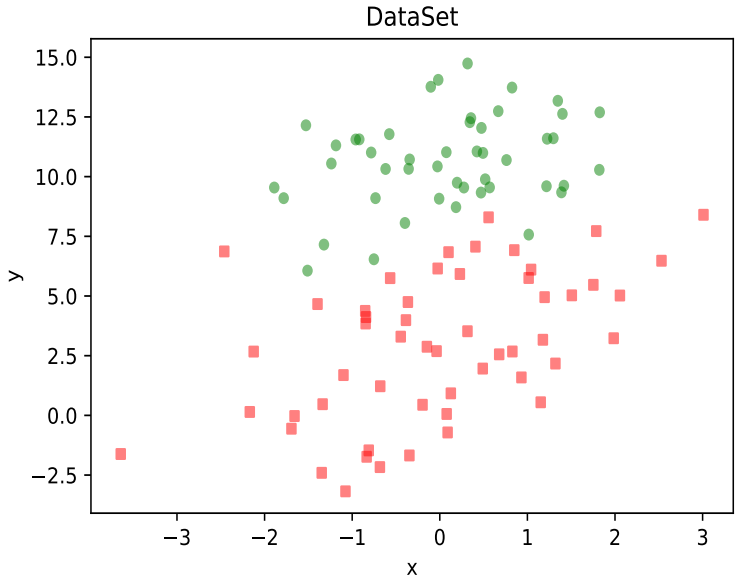

图 5.1: 一个简单数据集

考虑二分类问题，给定数据集 $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$，$x_i \subset \mathbb{R}^n, y_i \in \{0, 1\}, i = 1, 2, ..., N$

考虑到 $w^T x + b$ 取值是连续的，因此它不能拟合离散变量。可以考虑用它来拟合条件概率 $p(Y = 1|x)$，因为概率的取值也是连续的。

但是对于 $w \neq 0$（若等于零向量则没有什么求解的价值），$w^T x + b$ 取值为 $\mathbb{R}$，不符合概率取值为 0 到 1，因此考虑采用广义线性模型。

最理想的是阶跃函数，但阶跃函数不可微，对数几率函数是一个常用的替代函数：

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \tag{5.1}$$

于是有：

$$ln \frac{y}{1 - y} = w^T x + b \tag{5.2}$$

我们将 y 视为 x 为正例的概率，则 $1 - y$ 为 x 为其反例的概率。两者的比值称为几率 (odds)，指该事件发生与不发生的概率比值，若事件发生的概率为 p。则对数几率：

$$ln(odds) = ln\frac{y}{1-y} \tag{5.3}$$

将 y 视为类后验概率估计，重写公式有：

$$\begin{aligned} w^T x + b &= ln\frac{P(Y=1|x)}{1-P(Y=1|x)} \\ P(Y=1|x) &= \frac{1}{1+e^{-(w^T x+b)}} \end{aligned} \tag{5.4}$$

也就是说，输出 $Y=1$ 的对数几率是由输入 x 的线性函数表示的模型，这就是逻辑回归模型。当 $w^T x+b$ 的值越接近正无穷，$P(Y=1|x)$ 概率值也就越接近 1。因此逻辑回归的思路是，先拟合决策边界 (不局限于线性，还可以是多项式)，再建立这个边界与分类的概率联系，从而得到了二分类情况下的概率。

在这我们思考个问题，我们使用对数几率的意义在哪？通过上述推导我们可以看到 Logistic 回归实际上是使用线性回归模型的预测值逼近分类任务真实标记的对数几率，其优点有：

- 直接对分类的概率建模，无需实现假设数据分布，从而避免了假设分布不准确带来的问题（区别于生成式模型）；
- 不仅可预测出类别，还能得到该预测的概率，这对一些利用概率辅助决策的任务很有用；
- 对数几率函数是任意阶可导的凸函数，有许多数值优化算法都可以求出最优解。

5.1.1 代价函数

逻辑回归模型的数学形式确定后，剩下就是如何去求解模型中的参数。在统计学中，常常使用极大似然估计法来求解，即找到一组参数，使得在这组参数下，我们的数据的似然度（概率）最大。

设：

$$\begin{aligned} P(Y=1|x) = p(x) &= \frac{1}{1+e^{-(w^T x+b)}} \\ P(Y=0|x) &= 1-p(x) \end{aligned} \tag{5.5}$$

似然函数：

$$L(w) = \Pi[p(x_i)]_i^y [1-p(x_i)]^{1-y_i} \tag{5.6}$$

为了更方便求解，我们对等式两边同取对数，写成对数似然函数：

$$\begin{aligned} L(w) = &\sum [y_i lnp(x_i) + (1-y_i)\ln(1-p(x_i))] \\ &\sum [y_i \ln\frac{p(x_i)}{1-p(x_i)} + \ln(1-p(x_i))] \\ &\sum [y_i(w\dot{x}_i) - \ln(1+e^{w\dot{x}_i})] \end{aligned} \tag{5.7}$$

在机器学习中我们有损失函数的概念，其衡量的是模型预测错误的程度。如果取整个数据集上的平均对数似然损失，我们可以得到：

$$J(w) = -\frac{1}{N}\ln L(w) \tag{5.8}$$

即在逻辑回归模型中，我们最大化似然函数和最小化损失函数实际上是等价的。

5.1.2 梯度下降 (上升) 法

求解逻辑回归的方法有非常多，我们这里主要介绍梯度下降法。优化的主要目标是找到一个方向，参数朝这个方向移动之后使得损失函数的值能够减小，这个方向往往由一阶偏导或者二阶偏导各种组合求得。逻辑回归的损失函数是：

$$J(w) = -\frac{1}{N}\ln\sum[y_i lnp(x_i) + (1-y_i)\ln(1-p(x_i))] \tag{5.9}$$

梯度下降是通过 $J(w)$ 对 w 的一阶导数来找下降方向，并且以迭代的方式来更新参数，更新方式为

$$\begin{aligned} g_i &= \frac{\partial J(w)}{\partial w_i} = (p(x_i) - y_i)x_i \\ w_i^{k+1} &= w_i^k - \alpha g_i \end{aligned} \tag{5.10}$$

其中 k 为迭代次数。每次更新参数后，可以通过比较 $\|J(w^{k+1} - J(w^k))\|$ 小于阈值或者到达最大迭代次数来停止迭代。

```
'''
Parameters:
    inX - 数据
Returns:
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    dataMatIn - 数据集
    classLabels - 数据标签
Returns:
'''
# 函数说明:梯度上升算法
def gradAscent(dataMatIn, classLabels):
    dataMatrix = np.mat(dataMatIn)                          #转换成numpy的mat
    labelMat = np.mat(classLabels).transpose()              #转换成numpy的mat,并进行转置
    m, n = np.shape(dataMatrix)                             #返回dataMatrix的大小。m为行数,n为列数。
    alpha = 0.001                                           #移动步长,也就是学习速率,控制更新的幅度。
    maxCycles = 500                                         #最大迭代次数
    weights = np.ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix * weights)                   #梯度上升矢量化公式
        error = labelMat - h
        weights = weights + alpha * dataMatrix.transpose() * error
```

```
    return weights.getA()                                    #将矩阵转换为数组，返回权重数组
```

5.1.3 分析数据：画出决策边界

上面已经解出了一组回归系数，它确定了不同类别数据之间的分隔线。那么怎样画出该分隔线，从而使得优化的过程便于理解呢?

```
'''
Parameters:
    weights - 权重参数数组
Returns:
    无
'''
# 函数说明:绘制数据集
def plotBestFit(weights):
    dataMat, labelMat = loadDataSet()                        #加载数据集
    dataArr = np.array(dataMat)                              #转换成numpy的array数组
    n = np.shape(dataMat)[0]                                 #数据个数
    xcord1 = []; ycord1 = []                                 #正样本
    xcord2 = []; ycord2 = []                                 #负样本
    for i in range(n):                                       #根据数据集标签进行分类
        if int(labelMat[i]) == 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2]) #1为正样本
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2]) #0为负样本
    fig = plt.figure()
    ax = fig.add_subplot(111)                                #添加subplot
    ax.scatter(xcord1, ycord1, s = 20, c = 'red', marker = 's',alpha=.5)#绘制正样本
    ax.scatter(xcord2, ycord2, s = 20, c = 'green',alpha=.5)     #绘制负样本
    x = np.arange(-3.0, 3.0, 0.1)
    y = (-weights[0] - weights[1] * x) / weights[2]
    ax.plot(x, y)
    plt.title('BestFit')                                     #绘制title
    plt.xlabel('X1'); plt.ylabel('X2')                       #绘制label
    plt.show()
```

这个分类结果相当不错，从图上看只错分了两到四个点。但是，尽管例子简单且数据集很小，这个方法却需要大量的计算（300 次乘法）。因此下一节将对该算法稍作改进，从而使它可以用在真实数据集上。

5.1.4 随机梯度下降（上升）

梯度上升算法在每次更新回归系数时都需要遍历整个数据集，该方法在处理 100 个左右的数据集时尚可，但如果有数十亿样本和成千上万的特征，那么该方法的计算复杂度就太高了。一种改进方法是一次仅用一个样本点来更新回归系数，该方法称为**随机梯度上升算法**。由于可以在新样本到来时对分类器进行增量式更新，因而随机梯度上升算法是一个在线学习算法。与“在线学习”相对应，一次处理所有数据被称作是“批处理”。

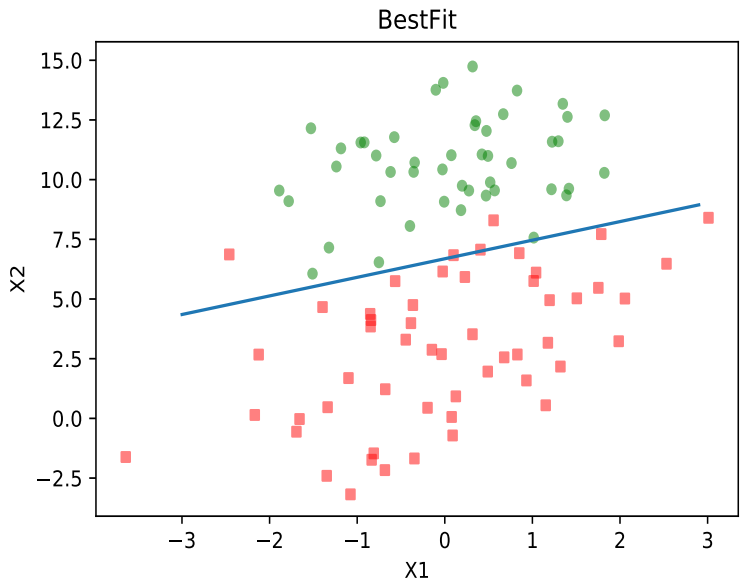

图 5.2: 梯度上升算法在 500 次迭代后得到的 Logistic 回归最佳拟合直线

随机梯度上升算法可以写成如下的伪代码:

```
所有回归系数初始化为1
对数据集中每个样本
    计算该样本的梯度
    使用alpha ℄ gradient更新回归系数值
返回回归系数值
```

```
'''
Parameters:
    dataMatrix - 数据数组
    classLabels - 数据标签
    numIter - 迭代次数
Returns:
    weights - 求得的回归系数数组(最优参数)
'''
# 函数说明:改进的随机梯度上升算法
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = np.shape(dataMatrix)                                  #返回dataMatrix的大小。m为行数,n为列数。
    weights = np.ones(n)                                        #参数初始化
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01                            #降低alpha的大小，每次减小1/(j+i)。
            randIndex = int(random.uniform(0,len(dataIndex)))   #随机选取样本
            h = sigmoid(sum(dataMatrix[randIndex]*weights))     #选择随机选取的一个样本，计算h
            error = classLabels[randIndex] - h                  #计算误差
            weights = weights + alpha * error * dataMatrix[randIndex]#更新回归系数
            del(dataIndex[randIndex])                           #删除已经使用的样本
    return weights
```

可以看到，随机梯度上升算法与梯度上升算法在代码上很相似，但也有一些区别：第一，后者的变量 h 和误差 error 都是向量，而前者则全是数值；第二，前者没有矩阵的转换过程，所有变量的数据类型都是 NumPy 数组。

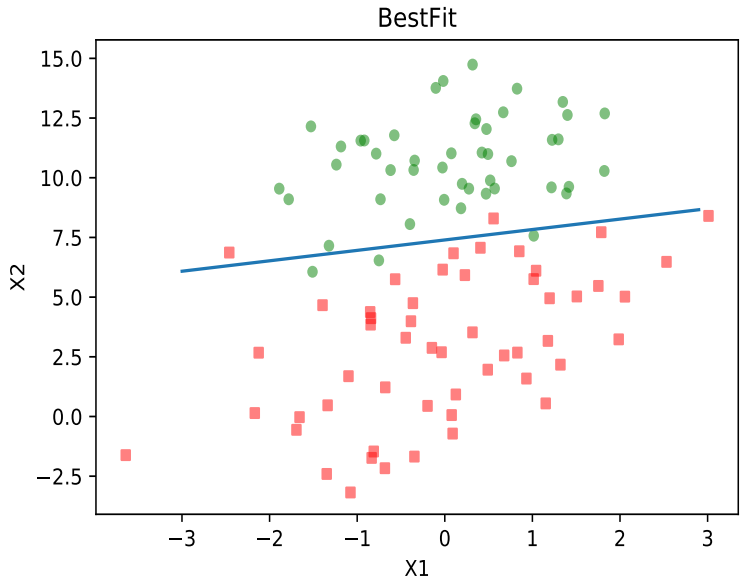

图 5.3: 随机梯度上升算法在上述数据集上的执行结果

5.2 Sklearn 示例：垃圾信息过滤

我们使用 UCI 机器学习仓库中的垃圾信息数据集，该数据集可以从学在浙大上下载。

```
import pandas as pd
df = pd.read_csv('./SMSSpamCollection', delimiter='\t', header=None)
print(df.head())

     0                                               1
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...

print('Number of spam messages: %s' % df[df[0] == 'spam'][0].count())
print('Number of ham messages: %s' % df[df[0] == 'ham'][0].count())

Number of spam messages: 747
Number of ham messages: 4825
```

我们使用 sklearn 中的 LogisticRegression 类来进行预测。首先，将数据集分为训练集和测试集。默认情况下，train_test_split 将 75% 的样本分为训练集，将剩余 25% 的样本分为测试集。

```
import numpy as np
```

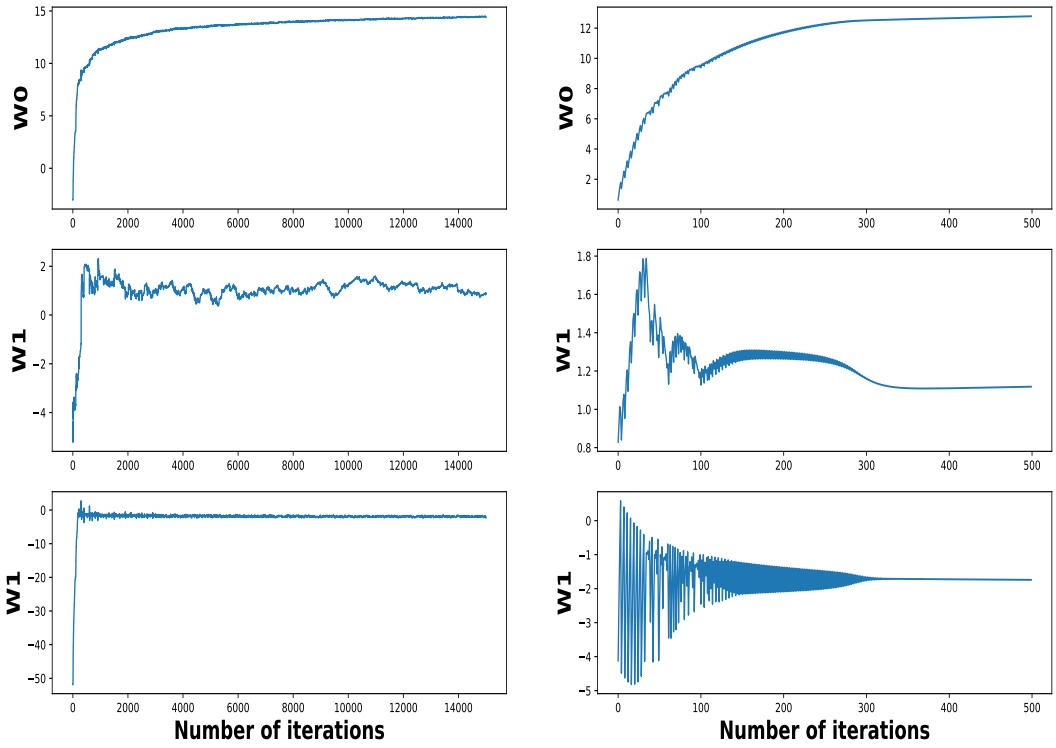

图 5.4: 运行随机梯度上升算法，在数据集的一次遍历中回归系数与迭代次数的关系图。左边：梯度上升算法，右边：随机梯度上升算法。

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score

X = df[1].values
y = df[0].values
X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y)
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train_raw)
X_test = vectorizer.transform(X_test_raw)
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
for i, prediction in enumerate(predictions[:5]):
    print('Predicted: %s, message: %s' % (prediction, X_test_raw[i]))

Predicted: ham, message: Now thats going to ruin your thesis!
Predicted: ham, message: Ok...
Predicted: ham, message: Its a part of checking IQ
Predicted: spam, message: Ringtone Club: Gr8 new polys direct to your mobile every week !
Predicted: ham, message: Talk sexy!! Make new friends or fall in love in the worlds most discreet text
     dating service. Just text VIP to 83110 and see who you could meet.
```

分类性能如何呢？我们在线性回归中使用的性能指标在该任务中不太适用，我们仅关注预

测的类是否正确，以及预测结果离决策边界有多远。

5.2.1 性能评估

利用准确率进行评估:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

df = pd.read_csv('./sms.csv')
X_train_raw, X_test_raw, y_train, y_test = train_test_split(df['message'], df['label'], random_state
    =11)
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train_raw)
X_test = vectorizer.transform(X_test_raw)
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
scores = cross_val_score(classifier, X_train, y_train, cv=5)
print('Accuracies: %s' % scores)
print('Mean accuracy: %s' % np.mean(scores))
```

利用**混淆矩阵**进行评估:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

y_test = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
y_pred = [0, 1, 0, 0, 0, 0, 0, 1, 1, 1]
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
plt.matshow(confusion_matrix)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

精准率和召回率

```
precisions = cross_val_score(classifier, X_train, y_train, cv=5, scoring='precision')
print('Precision: %s' % np.mean(precisions))
recalls = cross_val_score(classifier, X_train, y_train, cv=5, scoring='recall')
print('Recall: %s' % np.mean(recalls))

Precision: 0.992542742398
Recall: 0.683605030275
```

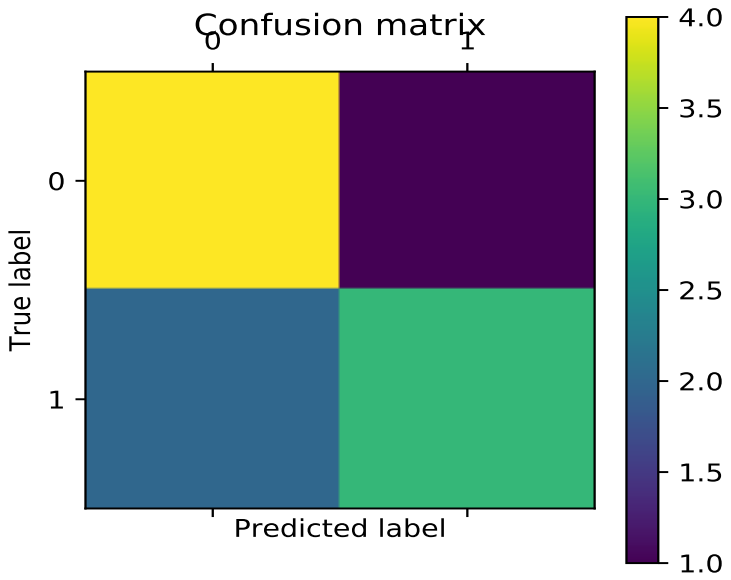

图 5.5: 混淆矩阵。

分类器的精确率为 0.992，表明几乎所有被预测为垃圾短信的信息实际上都是垃圾短信。它的召回率很低，说明接近 32% 的垃圾短信呗预测为非垃圾短信。

ROC 曲线

ROC（接收者操作特征）曲线中，给出了两条线，一条虚线一条实线。图中的横轴是伪正例的比例（假阳率 =FP/(FP+TN)），而纵轴是真正例的比例（真阳率 =TP/(TP+FN)）。ROC 曲线给出的是当阈值变化时假阳率和真阳率的变化情况。左下角的点所对应的是将所有样例判为反例的情况，而右上角的点对应的则是将所有样例判为正例的情况。虚线给出的是随机猜测的结果曲线。

ROC 曲线不但可以用于比较分类器，还可以基于成本效益（cost-versus-benefit）分析来做出决策。由于在不同的阈值下，不同的分类器的表现情况可能各不相同，因此以某种方式将它们组合起来或许会更有意义。如果只是简单地观察分类器的错误率，那么我们就难以得到这种更深入的洞察效果了。

在理想的情况下，最佳的分类器应该尽可能地处于左上角，这就意味着分类器在假阳率很低的同时获得了很高的真阳率。例如在垃圾邮件的过滤中，这就相当于过滤了所有的垃圾邮件，但没有将任何合法邮件误识为垃圾邮件而放入垃圾邮件的文件夹中。

对不同的 ROC 曲线进行比较的一个指标是曲线下的面积（Area Under the Curve，AUC）。AUC 给出的是分类器的平均性能值，当然它并不能完全代替对整条曲线的观察。一个完美分类器的 AUC 为 1.0，而随机猜测的 AUC 则为 0.5。

```
predictions = classifier.predict_proba(X_test)
false_positive_rate, recall, thresholds = roc_curve(y_test, predictions[:, 1])
roc_auc = auc(false_positive_rate, recall)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, recall, 'b', label='AUC = %0.2f' % roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.0])
plt.ylabel('Recall')
plt.xlabel('Fall-out')
plt.show()
```

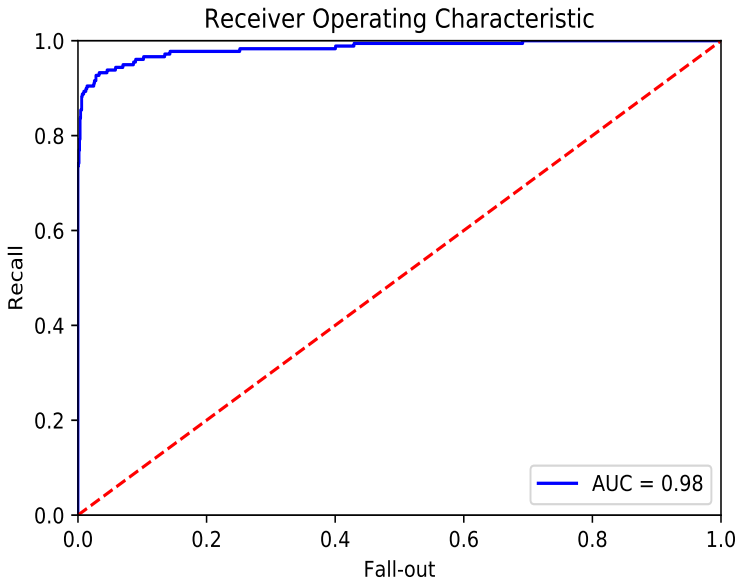

图 5.6: 垃圾信息过滤任务的 ROC 曲线

第 5 章 练习

1. 从疝气病症预测病马的死亡率

   本节将使用 Logistic 回归来预测患有疝病的马的存活问题。这里的数据（From UCI）包含 368 个样本和 28 个特征。疝病是描述马胃肠痛的术语。然而，这种病不一定源自马的胃肠问题，其他问题也可能引发马疝病。该数据集中包含了医院检测马疝病的一些指标，有的指标比较主观，有的指标难以测量，例如马的疼痛级别。

   另外需要说明的是，除了部分指标主观和难以测量外，该数据还存在一个问题，数据集中有 30% 的值是缺失的。下面将首先介绍如何处理数据集中的数据缺失问题，然后再利用 Logistic 回归和随机梯度上升算法来预测病马的生死。

   准备数据：处理数据中的缺失值

   数据中的缺失值是个非常棘手的问题，有很多文献都致力于解决这个问题。那么，数据缺失究竟带来了什么问题？假设有 100 个样本和 20 个特征，这些数据都是机器收集回来的。若机器上的某个传感器损坏导致一个特征无效时该怎么办？此时是否要扔掉整个数据？这种情况下，另外 19 个特征怎么办？它们是否还可用？答案是肯定的。因为有时候数据相当昂贵，扔掉和重新获取都是不可取的，所以必须采用一些方法来解决这个问题。下面给出了一些可选的做法：

   - 使用可用特征的均值来填补缺失值；
   - 使用特殊值来填补缺失值，如 1；
   - 忽略有缺失值的样本；

- 使用相似样本的均值添补缺失值；
- 使用另外的机器学习算法预测缺失值。

在预处理阶段需要做两件事：第一，所有的缺失值必须用一个实数值来替换，因为我们使用的 NumPy 数据类型不允许包含缺失值。这里选择实数 0 来替换所有缺失值，恰好能适用于 Logistic 回归。这样做的直觉在于，我们需要的是一个在更新时不会影响系数的值。

预处理中做的第二件事是，如果在测试数据集中发现了一条数据的类别标签已经缺失，那么我们的简单做法是将该条数据丢弃。这是因为类别标签与特征不同，很难确定采用某个合适的值来替换。处理好的数据放在学在浙大。

测试算法：用 Logistic 回归进行分类

请独立编写代码。

评估和分析算法

- 准确率
- 精确率
- 召回率
- 混淆矩阵
- ROC 曲线

5.3 多类别分类

多类别分类问题是将一个实例分配到类集合中的某一个。sklearn 使用一种**一对全**的策略来支持多类别分类。LogisticRegression 类本身就使用一对全策略支持多类别分类。

假设你想看一部电影，但对烂片特别厌恶。使用 sklearn 找出评论较好的电影。我们将取自烂片数据库中影评的情绪短语进行分类，每个短语分为几种情绪：负向，略负向，中立，略正向，正向。

数据集可以从学在浙大上下载。

首先我们使用 pandas 类库探索该数据集。

```
import pandas as pd
df = pd.read_csv('./train.tsv', header=0, delimiter='\t')
print(df.count())

PhraseId     156060
SentenceId   156060
Phrase       156060
Sentiment    156060
dtype: int64

print(df.head())

   PhraseId SentenceId                                           Phrase \
0         1          1  A series of escapades demonstrating the adage ...
```

```
1        2          1  A series of escapades demonstrating the adage ...
2        3          1                                          A series
3        4          1                                                A
4        5          1                                           series

   Sentiment
0          1
1          2
2          2
3          2
4          2
```

Sentiment 列包含响应变量。标签 0 对应情绪负向，1 对应略负向。phrase 列包含原始文本。来自电影评论的每一个句子已经被解析为短语。

```
print(df['Phrase'].head(10))

0    A series of escapades demonstrating the adage ...
1    A series of escapades demonstrating the adage ...
2                                           A series
3                                                  A
4                                             series
5    of escapades demonstrating the adage that what...
6                                                 of
7    escapades demonstrating the adage that what is...
8                                         escapades
9    demonstrating the adage that what is good for ...
Name: Phrase, dtype: object

print(df['Sentiment'].describe())

count   156060.000000
mean         2.063578
std          0.893832
min          0.000000
25%          2.000000
50%          2.000000
75%          3.000000
max          4.000000
Name: Sentiment, dtype: float64

print(df['Sentiment'].value_counts())

2    79582
3    32927
1    27273
4     9206
0     7072
Name: Sentiment, dtype: int64
```

```
print(df['Sentiment'].value_counts()/df['Sentiment'].count())

2    0.509945
3    0.210989
1    0.174760
4    0.058990
0    0.045316
Name: Sentiment, dtype: float64
```

最常见的中立类超过 50% 实例。如果一个很差的分类器将所有的实例都预测为中立类，那么准确率接近 50%。接近四分之一的影评是正向或略正向，接近五分之一的影评是负向。

使用 sklearn 训练一个分类器：

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV


df = pd.read_csv('./train.tsv', header=0, delimiter='\t')
X, y = df['Phrase'], df['Sentiment'].as_matrix()
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5)
grid_search = main(X_train, y_train)
pipeline = Pipeline([
    ('vect', TfidfVectorizer(stop_words='english')),
    ('clf', LogisticRegression())
])
parameters = {
    'vect__max_df': (0.25, 0.5),
    'vect__ngram_range': ((1, 1), (1, 2)),
    'vect__use_idf': (True, False),
    'clf__C': (0.1, 1, 10),
}
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, scoring='accuracy')
grid_search.fit(X_train, y_train)
print('Best score: %0.3f' % grid_search.best_score_)
print('Best parameters set:')
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print('t%s: %r' % (param_name, best_parameters[param_name]))
```

多类别分类性能衡量指标

```
predictions = grid_search.predict(X_test)

print('Accuracy: %s' % accuracy_score(y_test, predictions))
print('Confusion Matrix:')
```

```
print(confusion_matrix(y_test, predictions))
print('Classification Report:')
print(classification_report(y_test, predictions))

Accuracy: 0.636255286428
Confusion Matrix:
[[ 1124 1725  628   65   10]
 [  923 6049 6132  583   34]
 [  197 3131 32658 3640  137]
 [   15  398 6530 8234 1301]
 [    3   43  530 2358 1582]]
Classification Report:
             precision    recall  f1-score   support

          0       0.50      0.32      0.39      3552
          1       0.53      0.44      0.48     13721
          2       0.70      0.82      0.76     39763
          3       0.55      0.50      0.53     16478
          4       0.52      0.35      0.42      4516

avg / total       0.62      0.64      0.62     78030
```

第 5 章 练习

1. 通过 Logistic Regression 预测 Titanic 乘客是否能在事故中生还。
   从学在浙大上下载数据。

```
import numpy as np
import pandas as pd

from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white") #设置seaborn画图的背景为白色
sns.set(style="whitegrid", color_codes=True)

# 将数据读入 DataFrame
df = pd.read_csv("./titanic_data.csv")

# 预览数据
df.head()

print('数据集包含的数据个数 {}.'.format(df.shape[0]))
```

   - 导入工具库和数据
   - 查看缺失数据
   - 年龄

```
print('"age" 缺失的百分比 %.2f%%' %((df['age'].isnull().sum()/df.shape[0])*100))

ax = df["age"].hist(bins=15, color='teal', alpha=0.6)
ax.set(xlabel='age')
plt.xlim(-10,85)
plt.show()
```

- 仓位

```
# 仓位缺失的百分比
print('"Cabin" 缺失的百分比 %.2f%%' %((df['cabin'].isnull().sum()/df.shape[0])*100))
```

- 登船地点

```
# 登船地点的缺失率
print('"Embarked" 缺失的百分比 %.2f%%' %((df['embarked'].isnull().sum()/df.shape[0])*100)
    )
```

- 对数据进行调整
  对缺失的数据如何处理？年龄，仓位，登陆地点？
- 额外的变量
  其他数据如何处理
- 数据分析
  性别分布、仓位分布、哪个地点登陆的乘客容易生还
- Logistic Regression
  如何建立模型？如何评估？