电子科技大学信息与软件工程学院

实 验 报 告

学　　号 2018091618008

姓　　名 袁昊男

（实验）课程名称 机器学习技术与应用

理论教师 黄　俊

实验教师 杨　珊

电子科技大学教务处制表

电 子 科 技 大 学

实　验　报　告

学生姓名：袁昊男　　学号：2018091618008　　指导教师：杨珊

实验地点：在线实验　　实验时间：2020.05.13

一、实验室名称：信息与软件工程学院实验中心

二、实验名称：评分卡预测信用风险的逻辑回归模型

三、实验学时：4 学时

四、实验原理：

对数几率回归（也称“逻辑回归”，英语：Logistic Regression 或 Logit Regression），即对数几率模型（英语：Logit Model，也译作“逻辑模型”、“评定模型”、“分类评定模型”）是离散选择法模型之一，属于多重变量分析范畴，是社会学、生物统计学、临床、数量心理学、计量经济学、市场营销等统计实证分析的常用方法。

$$P\left(Y=1\middle|X=x\right)=\frac{\mathrm{e}^{x'\beta}}{1+\mathrm{e}^{x'\beta}}$$

输入：观测数据 data，分为训练集和测试集，模型 $P\left(Y=1\middle|X=x\right)$。

输出：使用评分卡模型来判断该用户的信用，以判断是否借钱给他。

(1) 对于所有的 $i\in\{1,2,\cdots,n\}$，取初值 $W_i=0$；

(2) 对每个 $i\in\{1,2,\cdots,n\}$，使用

$$J(\theta)=-\sum_{i=1}^{m}(y_i\cdot\log(y_\theta(x_i))+(1-y_i)\cdot\log(1-y_\theta(x_i)))$$

函数，求解出损失值；

(3) 使用下列函数求解出正则项；

$$J(\theta)_{L1}=C\cdot J(\theta)+\sum_{j=1}^{n}\left|\theta_j\right|(j\geq 1)$$

$$J(\theta)_{L2} = C \cdot J(\theta) + \sqrt{\sum_{j=1}^{n} (\theta_j)^2} \, (j \geq 1)$$

(4) 使用梯度下降法更新θ；

$$\frac{\partial}{\partial \theta_j} J(\theta) = d_j = \sum_{i=1}^{m} (y_\theta(x_i) - y_i) x_{ij}$$

(5) 重复这几个过程，直至得出客户的评分卡为止。

五、实验目的：

1、 理解逻辑回归、二项回归模型和模型参数估计。

2、 掌握评分卡的建模与制作流程的基本原理和算法实现。

六、实验内容：

1、 导库，获取数据。

2、 探索数据与数据预处理。

3、 分箱。

4、 计算各箱的 WOE 并映射到数据中。

5、 建模与模型验证。

6、 制作评分卡。

七、实验器材（设备、元器件）：

1、 PC 电脑。

2、 Python。

八、实验步骤：

1、 导库，获取数据

```
1. import numpy as np
2. import pandas as pd
3. from sklearn.linear_model import LogisticRegression as LR
4. import matplotlib.pyplot as plt
5. import seaborn as sns
6. from sklearn.ensemble import RandomForestRegressor as rfr
7. from imblearn.over_sampling import SMOTE
8. from sklearn.model_selection import train_test_split
9. import matplotlib.pyplot as plt
10. import scipy
11. import scikitplot as skplt
12.
13. data = pd.read_csv(r"rankingcard.csv", index_col=0)
```

说明：日常在导库的时候，并不是一次性能够知道我们要用的所有库的。通常都是在建模过程中逐渐导入需要的库。数据集特征含义如下表所示：

特征/标签	含义
SeriousDlqin2yrs	出现 90 天或更长时间的逾期行为（即定义好坏客户）
RevolvingUtilizationOfUnsecuredLines	贷款以及信用卡可用额度与总额度比例
age	借款人借款年龄
NumberOfTime30-59DaysPastDueNotWorse	过去两年内出现35-59天逾期但是没有发展得更坏的次数
DebtRatio	每月偿还债务，赡养费，生活费用除以月总收入
MonthlyIncome	月收入
NumberOfOpenCreditLinesAndLoans	开放式贷款和信贷数量
NumberOfTimes90DaysLate	过去两年内出现90天逾期或更坏的次数
NumberRealEstateLoansOrLines	抵押贷款和房地产贷款数量，包括房屋净值信贷额度
NumberOfTime60-89DaysPastDueNotWorse	过去两年内出现60-89天逾期但是没有发展得更坏的次数
NumberOfDependents	家庭中不包括自身的家属人数（配偶，子女等）

2、 去除重复值

```
print("特征样本数量：")
print(data.dtypes.value_counts())
print("重复项按特征统计：")
print(data[data.duplicated()].count())
nodup = data[-data.duplicated()]
print(len(nodup))
print(len(data.drop_duplicates()))
print(len(data) - len(nodup))
data.drop_duplicates(inplace=True)
print(data.info())

# 重设索引
data = data.reset_index(drop=True)
# data.index = range(data.shape[0])
print(data.info())
```

说明：银行业数据可能会存在的一个问题就是样本重复，即有超过一行的样本所显示的所有特征都一样。有时候可能是人为输入重复，有时候可能是系统录入重复，总而言之我们必须对数据进行去重处理。查看特征样本数量与统计重复项。

3、 填补缺失值

```
def missing_values_table(df):
    # 缺失值计数
    mis_val = df.isnull().sum()

    # 缺失值占比
    mis_val_percent = 100 * mis_val / len(df)
```

```
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
    mis_val_table_ren_columns = mis_val_table.rename(
        columns={0: 'Missing Values', 1: '% of Total Values'})

    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:, 1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

    # 打印统计信息
    print("共" + str(df.shape[1]) + "行\n"
              "其中" + str(mis_val_table_ren_columns.shape[0]) +"行的数据缺失。")
    return mis_val_table_ren_columns

missing_values = missing_values_table(data)
missing_values.head(20)
```

说明：统计数据中的缺失值。通过统计结果发现：我们需要填补的特征是“收入”和“家属人数”。“家属人数”缺失很少，仅缺失了大约 2.5%，可以考虑直接删除，或者使用均值来填补。“收入”缺失了几乎 20%，并且我们知道，“收入”必然是一个对信用评分来说很重要的因素，因此这个特征必须要进行填补。在这里，我们使用均值填补“家属人数”。

```
data['NumberOfDependents'].fillna(data['NumberOfDependents'].mean(), inplace=True)
```

说明：使用均值填充“家属人数”。

```
def fill_missing_rf(X, y, to_fill):
    # 构建新特征矩阵和新标签
    df = X.copy()
    fill = df.loc[:, to_fill]
    df = pd.concat([df.loc[:, df.columns != to_fill], pd.DataFrame(y)], axis=1)

    # 训练集和测试集
    Ytrain = fill[fill.notnull()]
    Ytest = fill[fill.isnull()]
    Xtrain = df.iloc[Ytrain.index, :]
    Xtest = df.iloc[Ytest.index, :]

    # 用随机森林回归来填补缺失值
    from sklearn.ensemble import RandomForestRegressor as rfr
    rfr = rfr(n_estimators=100)  # random_state=0,n_estimators=200,max_depth=3,n_jobs=-1
    rfr = rfr.fit(Xtrain, Ytrain)
    Ypredict = rfr.predict(Xtest)

    return Ypredict

X = data.iloc[:, 1:]
y = data["SeriousDlqin2yrs"]  # y = data.iloc[:,0]
X.shape  # (149391, 10)
```

```
y_pred = fill_missing_rf(X, y, "MonthlyIncome")

temp_b = (y_pred.shape == data.loc[data.loc[:, "MonthlyIncome"].isnull(), "MonthlyIncome"].shape)
if temp_b:
    data.loc[data.loc[:, "MonthlyIncome"].isnull(), "MonthlyIncome"] = y_pred

data.info()
```

说明：随机森林填充缺失的月收入。定义函数 fill_missing_rf(X, y, to_fill)，其中参数的含义是：X 为要填补的特征矩阵；y 为完整、没有缺失值的标签；to_fill 为要填补的那一列的名称字符串。在函数后将待处理数据作为参数传入函数，进行随机森林填充。之后检验操作前后数据数量是否一致，若结果合理一致，则将原数据覆盖。

4、描述性统计处理异常值

```
data.describe()
temp_desc = data.describe([0.01, 0.1, 0.25, .5, .75, .9, .99]).T
print(temp_desc[['min', 'max']])
temp_desc.to_csv("temp_desc.csv")
```

说明：现实数据永远都会有一些异常值，首先我们要去把他们捕捉出来，然后观察他们的性质。日常处理异常值，我们使用箱线图或 3σ 法则来找到异常值。但在银行数据中，我们希望排除的“异常值”不是一些超高或超低的数字，而是一些不符合常理的数据：比如，收入不能为负数，但是一个超高水平的收入却是合理的，可以存在的。所以在银行业中，我们往往就使用普通的描述性统计来观察数据的异常与否与数据的分布情况。特征量有限时使用。通过上述代码我们观察到，年龄的最小值为 0，这不符合银行的业务需求，即便是儿童账户也要至少 8 岁，我们可以查看年龄为 0 的人数。

```
(data["age"] == 0).sum()
data = data[data["age"] != 0]
```

说明：发现只有一个人年龄为 0，可以判断这肯定是录入失误造成的，可以当成是缺失值来处理，直接删除掉这个样本。

```
print(data['NumberOfTime30-59DaysPastDueNotWorse'].value_counts())
print(data['NumberOfTime60-89DaysPastDueNotWorse'].value_counts())
print(data['NumberOfTimes90DaysLate'].value_counts())
print(data.loc[data['NumberOfTime30-59DaysPastDueNotWorse'] >= 90, 'SeriousDlqin2yrs'].value_counts())
```

```
print(data.loc[data['NumberOfTime60-89DaysPastDueNot-Worse'] >= 90, 'SeriousDlqin2yrs'].value_counts())
print(data.loc[data['NumberOfTimes90DaysLate'] >= 90, 'Seri-ousDlqin2yrs'].value_counts())
# 删除异常值
data = data[data.loc[:, "NumberOfTime30-59DaysPastDueNot-Worse"] < 90]
data = data[data.loc[:, "NumberOfTime60-89DaysPastDueNot-Worse"] < 90]
data = data[data.loc[:, "NumberOfTimes90DaysLate"] < 90]
# 恢复索引
data.index = range(data.shape[0])
data.info()
```

说明：样本数据中有三个指标看起来很奇怪："NumberOfTime30-59DaysPastDueNotWorse"、"NumberOfTime60-89DaysPastDueNotWorse"、"NumberOfTimes90DaysLate"，这三个指标分别是“过去两年内出现 35-59 天逾期但是没有发展的更坏的次数”，“过去两年内出现 60-89 天逾期但是没有发展的更坏的次数”，“过去两年内出现 90 天逾期的次数”。这三个指标，在 99%的分布的时候依然是 2，最大值却是 98，不符合常理。通过代码统计异常样本数，发现有 225 个样本存在这样的情况，并且这些样本标签并不都是 1，说明他们并不都是坏客户。因此，我们基本可以判断，这些样本是某种异常，应该把它们删除。最后一定记得恢复索引。

5、样本不均衡问题

```
X_temp = data.iloc[:, 1:]
y_temp = data["SeriousDlqin2yrs"]  # y = data.iloc[:,0]

sm = SMOTE(random_state=42)  # 实例化
X, y = sm.fit_sample(X_temp, y_temp)

n_sample_ = X.shape[0]
pd.Series(y).value_counts()
n_1_sample = pd.Series(y).value_counts()[1]
n_0_sample = pd.Series(y).value_counts()[0]
print('样本共：{}个；1 占{:.2%}；0 占{:.2%}'.format(n_sam-ple_, n_1_sample / n_sample_, n_0_sample / n_sample_))
```

说明：对于银行来说，真正想要被判别出来的其实是“恶意违约”的人，而这部分人数非常非常少，样本就会不均衡。我们有着处理样本不均衡的各种方法，其中主流的是采样法，是通过重复样本的方式来平衡标签，可以进行上采样（增加少数类的样本），比如 SMOTE，或者下采样（减少多数类的样本）。对于逻辑回归来说，上采样是最好的办法。执行本段代码后发现：样本容量为 278584，其中 1 占 50.00%，0 占 50.00%。

6、 分训练集与测试集

```
X = pd.DataFrame(X)
y = pd.DataFrame(y)

X_train, X_vali, Y_train, Y_vali = train_test_split(X, y, test_size=0.3, random_state=420)
model_data = pd.concat([Y_train, X_train], axis=1)  # 训练数据构建模型
model_data.index = range(model_data.shape[0])
model_data.columns = data.columns

vali_data = pd.concat([Y_vali, X_vali], axis=1)  # 测试集
vali_data.index = range(vali_data.shape[0])
vali_data.columns = data.columns

n_sample_ = len(Y_train)
n_1_sample = Y_train.iloc[:, 0].value_counts()[1]
n_0_sample = Y_train.iloc[:, 0].value_counts()[0]
print('样本共：{}个；1 占{:.2%}；0 占{:.2%}'.format(n_sample_, n_1_sample / n_sample_, n_0_sample / n_sample_))

n_sample_ = len(Y_vali)
n_1_sample = Y_vali.iloc[:, 0].value_counts()[1]
n_0_sample = Y_vali.iloc[:, 0].value_counts()[0]
print('样本共：{}个；1 占{:.2%}；0 占{:.2%}'.format(n_sample_, n_1_sample / n_sample_, n_0_sample / n_sample_))

print(model_data.shape)
print(vali_data.shape)
model_data.to_csv(r"model_data.csv") # 训练数据
vali_data.to_csv(r"vali_data.csv") # 测试数据

model_data = pd.read_csv(r"model_data.csv")
vali_data = pd.read_csv("vali_data.csv")
print(model_data.shape)
print(vali_data.shape)
model_data.drop('Unnamed: 0', inplace=True, axis=1)
vali_data.drop('Unnamed: 0', inplace=True, axis=1)
print(model_data.shape)
print(vali_data.shape)
```

说明：上采样之后，划分为训练集、测试集；保存前期处理结果。划分结果为：训练集样本个数：195008，1 占 49.90%，0 占 50.10%；测试集样本个数：83576，1 占 50.22%，0 占 49.78%。将训练集保存为 model_data.csv、测试集保存为 vali_data.csv。

7、 分箱

```
model_data["qcut"], updown = pd.qcut(model_data["age"], retbins=True, q=20)  # 等频分箱

# 在这里时让 model_data 新添加一列叫做“分箱”，这一列其实就是每个样本所对应的箱子
print(model_data["qcut"].value_counts())
# 所有箱子的上限和下限
```

```
print(updown)

coount_y0 = model_data[model_data["SeriousDlqin2yrs"] == 0].groupby(by="qcut").count()["SeriousDlqin2yrs"]
coount_y1 = model_data[model_data["SeriousDlqin2yrs"] == 1].groupby(by="qcut").count()["SeriousDlqin2yrs"]

num_bins = [*zip(updown, updown[1:], coount_y0, coount_y1)]

print(num_bins)

for i in range(20):  # 20个箱子
    # 如果第一个组没有包含正样本或负样本，向后合并
    if 0 in num_bins[0][2:]:
        num_bins[0:2] = [(
            num_bins[0][0],
            num_bins[1][1],
            num_bins[0][2] + num_bins[1][2],
            num_bins[0][3] + num_bins[1][3])]
        continue

    for i in range(len(num_bins)):
        if 0 in num_bins[i][2:]:
            num_bins[i - 1:i + 1] = [(
                num_bins[i - 1][0],
                num_bins[i][1],
                num_bins[i - 1][2] + num_bins[i][2],
                num_bins[i - 1][3] + num_bins[i][3])]
            break
    else:
        break
```

说明：首先进行等频分箱，对需要分箱的列进行处理。pd.qcut 函数是基于分位数的分箱函数，本质是将连续型变量离散化，只能够处理一维数据。返回箱子的上限和下限，参数 q 是要分箱的个数；参数 retbins 为返回箱子上下限数组。然后统计每个分箱中 0 和 1 的数量（这里使用了数据透视表的功能 groupby），num_bins 值分别为每个区间的上界、下界、0 出现的次数、1 出现的次数。后面两个循环确保每个箱中都有 0 和 1（都含有正负样本，否则 IV 值将无法计算）。

```
def get_woe(num_bins):
    columns = ["min", "max", "count_0", "count_1"]
    df = pd.DataFrame(num_bins, columns=columns)

    df["total"] = df.count_0 + df.count_1  # 一个箱子当中所有的样本数： 按列相加
    df["percentage"] = df.total / df.total.sum()  # 一个箱子里的样本数，占所有样本的比例
    df["bad_rate"] = df.count_1 / df.total  # 一个箱子坏样本的数量占一个箱子里边所有样本数的比例
    df["good%"] = df.count_0 / df.count_0.sum()
```

```
9.      df["bad%"] = df.count_1 / df.count_1.sum()
10.     df["woe"] = np.log(df["good%"] / df["bad%"])
11.     return df
```

说明：get_woe 函数通过 num_bins 数据计算 WOE。WOE（Weight of Evidence）叫做证据权重，表示的实际上是“当前分组中坏客户占所有坏客户的比例”和“当前分组中好客户占所有坏客户的比例”的差异。计算公式如下：

$$WOE_i = \ln\left(\frac{good\%}{bad\%}\right)$$

转化公式以后，也可以理解为：当前这个组中坏客户和好客户的比值，和所有样本中这个比值的差异。这个差异为这两个比值的比值，再取对数来表示的。WOE 越大，这种差异越大，这个分组里的样本坏样本可能性就越大，WOE 越小，差异越小，这个分组里的坏样本可能性就越小。

```
1.  def get_iv(df):
2.      rate = df["good%"] - df["bad%"]
3.      iv = np.sum(rate * df.woe)
4.      return iv
```

说明：get_iv 函数计算 IV 值。IV（Information Value）即信息价值，用来衡量自变量的预测能力，IV 的计算基于 WOE，可以看成对 WOE 的加权求和。计算公式如下：

$$IV = \sum_{i=1}^{N}(good\% - bad\%)\cdot WOE_i$$

IV 是对整个特征来说的，IV 代表的意义是我们特征上的信息量以及这个特征对模型的贡献，由下表来控制：

IV	特征对预测函数的贡献
< 0.03	特征几乎不带有效信息，对模型没有贡献，这种特征可以删除
0.03 ~ 0.09	有效信息很少，对模型的贡献度很低
0.1 ~ 0.29	有效信息一般，对模型的贡献度中等
0.3 ~ 0.49	有效信息较多，对模型的贡献度较高
>= 0.5	有效信息非常多，对模型的贡献度超高并且可疑

可见，IV 并非越大越好，我们想要找到 IV 的大小和箱子个数的平衡点。箱子越多，IV 必然越小，因为信息损失会非常多，所以，我们会对特征进行分箱，然后计算每个特征在每个箱子数目下的 WOE 值，利用 IV 值的曲线，找出合适的分箱个数。

```
1.  num_bins_ = num_bins.copy()
2.  IV = []
```

```
axisx = []
PV = []

while len(num_bins_) > 2:  # 大于设置的最低分箱个数
    pvs = []
    for i in range(len(num_bins_) - 1):
        x1 = num_bins_[i][2:]
        x2 = num_bins_[i + 1][2:]
        pv = scipy.stats.chi2_contingency([x1, x2])[1]  # p 值
        # chi2 = scipy.stats.chi2_contingency([x1,x2])[0] # 计算卡方值
        pvs.append(pv)

    i = pvs.index(max(pvs))
    num_bins_[i:i + 2] = [(
        num_bins_[i][0],
        num_bins_[i + 1][1],
        num_bins_[i][2] + num_bins_[i + 1][2],
        num_bins_[i][3] + num_bins_[i + 1][3])]

    bins_df = get_woe(num_bins_)
    axisx.append(len(num_bins_))
    IV.append(get_iv(bins_df))
    PV.append(max(pvs))  # 卡方 p 值

plt.figure()
plt.plot(axisx, IV)
# plt.plot(axisx,PV)
plt.xticks(axisx)
plt.xlabel("number of box")
plt.ylabel("IV")
plt.show()

num_bins_ = num_bins.copy()
x1 = num_bins_[0][2:]
x2 = num_bins_[0 + 1][2:]

pv = scipy.stats.chi2_contingency([x1, x2])[1]
chi2 = scipy.stats.chi2_contingency([x1, x2])[0]
```

说明：本段代码进行卡方检验，合并箱体，画 IV 曲线。首先获取 num_bins_两两之间的卡方检验的置信度（或卡方值），通过卡方 p 值进行处理。合并卡方 p 值最大的两组。画出 IV 曲线，选择转折点处（下坠最快的点）。6→5 折线点最陡峭，所以这里对 age 来说选择箱数为 6。经过计算可以得到：p 值为 0.0005943767678537757，卡方值为 11.793506471136046。

```
def get_bin(num_bins_, n):
    while len(num_bins_) > n:
        pvs = []
        for i in range(len(num_bins_) - 1):
            x1 = num_bins_[i][2:]
            x2 = num_bins_[i + 1][2:]
            pv = scipy.stats.chi2_contingency([x1, x2])[1]
```

```
            # chi2 = scipy.stats.chi2_contingency([x1,x2])[0]
            pvs.append(pv)

        i = pvs.index(max(pvs))
        num_bins_[i:i + 2] = [(
            num_bins_[i][0],
            num_bins_[i + 1][1],
            num_bins_[i][2] + num_bins_[i + 1][2],
            num_bins_[i][3] + num_bins_[i + 1][3])]
    return num_bins_

afterbins = num_bins.copy()
get_bin(afterbins, 6)
```

说明：函数 get_bin 用来合并箱体。num_bins_值分别为每个区间的上界、下界、0 出现的次数、1 出现的次数；n 为箱数，这里示例选择 6。

```
hand_bins = {"NumberOfTime30-59DaysPastDueNot-Worse": [0, 1, 2, 13]
    , "NumberOfTimes90DaysLate": [0, 1, 2, 17]
    , "NumberRealEstateLoansOrLines": [0, 1, 2, 4, 54]
    , "NumberOfTime60-89DaysPastDueNotWorse": [0, 1, 2, 8]
    , "NumberOfDependents": [0, 1, 2, 3]}

hand_bins = {k: [-np.inf, *v[:-1], np.inf] for k, v in hand_bins.items()}

bins_of_col = {}
# 生成自动分箱的分箱区间和分箱后的 IV 值
for col in auto_col_bins:
    print(col)
    bins_df_temp = graphforbestbin(model_data, col
                                   , "SeriousDlqin2yrs"
                                   , n=auto_col_bins[col]
                                   , q=20
                                   , graph=True)
    bins_list = sorted(set(bins_df_temp["min"]).union(bins_df_temp["max"]))
    bins_list[0], bins_list[-1] = -np.inf, np.inf
    bins_of_col[col] = bins_list

# 合并手动分箱数据
bins_of_col.update(hand_bins)
bins_of_col
```

说明：样本特征中存在不能自动分箱的特征，如"NumberOfTime30-59DaysPastDueNotWorse"、"NumberOfTimes90DaysLate"、"NumberRealEstateLoansOrLines"、"NumberOfDependents"，需要手动分箱。为了保证区间覆盖使用 np.inf 替换最大值，用-np.inf 替换最小值，原因是比如一些新的值出现，例如家庭人数为 30，在以前没出现过，改成范围为极大值之后，这些新值就都能分到箱里边了。

8、 建模与模型验证

```
train_X = model_woe.iloc[:, :-1]
train_y = model_woe.iloc[:, -1]
text_X = vali_woe.iloc[:, :-1]
text_y = vali_woe.iloc[:, -1]
lr = LR().fit(train_X, train_y)
lr.score(text_X, text_y)
vali_proba_df = pd.DataFrame(lr.predict_proba(vali_X))
skplt.metrics.plot_roc(vali_y, vali_proba_df,
                       plot_micro=False,figsize=(6,6),
                       plot_macro=False)
```

说明：根据划分的训练集与测试集调用 LogisticRegression 进行拟合，并使用准确率和 ROC 曲线验证模型的预测能力。经测试，准确率为 0.8656671771800517。

```
c_1 = np.linspace(0.01, 1, 20)
score = []
for i in c_1:  # 先 c_1 大范围，再 c_2 小范围
    lr = LR(solver='liblinear', C=i).fit(train_X, train_y)
    score.append(lr.score(text_X, text_y))
plt.figure()
plt.plot(c_1, score)
plt.show()
c_2 = np.linspace(0.01, 0.2, 20)
score = []
for i in c_2:  # 先 c_1 大范围，再 c_2 小范围：  最高点在 0.06
    lr = LR(solver='liblinear', C=i).fit(train_X, train_y)
    score.append(lr.score(text_X, text_y))
plt.figure()
plt.plot(c_2, score)
plt.show()
score = []
iter_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in iter_list:
    lr = LR(solver='libline-
ar', C=0.06, max_iter=i).fit(train_X, train_y)
    score.append(lr.score(text_X, text_y))

plt.figure()
plt.plot(iter_list, score)
plt.show()
print(max(score), iter_list[score.index(max(score))])
print(lr.n_iter_)
```

说明：迭代参数调整。调整后，准确率略有提升，为 0.86569110749497477。

9、 制作评分卡

```
B = 20/np.log(2)
A = 600 + B*np.log(1/60)
base_score = A - B*lr.intercept_
base_score
score_age = woeall["age"] * (-B*lr.coef_[0][0])
score_age
file = "ScoreData.csv"
```

```
8.  with open(file,"w") as fdata:
9.        fdata.write("base_score,{}\n".format(base_score))
10. for i,col in enumerate(X.columns):
11.       score = woeall[col] * (-B*lr.coef_[0][i])
12.       score.name = "Score"
13.       score.index.name = col
14.       score.to_csv(file,header=True,mode="a")
```

说明：标准评分卡的分数由以下公式计算：

$$Score = A - B \cdot \log(odds)$$

其中 A、B 是常数，A 叫做“补偿”，B 叫做“刻度”，$\log(odds)$ 代表了一个人违约的可能性。其实逻辑回归的结果取对数几率形式会得到 $\theta^T x$，即参数×特征矩阵，所以其实就是参数 $\log(odds)$。两个常数可以通过两个假设的分值代入公式求出，这两个假设分别是：某个特定违约概率下的预期分值、指定的违约概率翻倍的分数（PDO）。假设设定特定违约概率下的预期分值为 600，PDO 为 2。首先计算基准分数，之后使用循环，每次生成一组 score_age 类似的分档和分数，不断写入 ScoreData.csv 文件之中。

九、实验数据及结果分析

1、 样本数据可视化

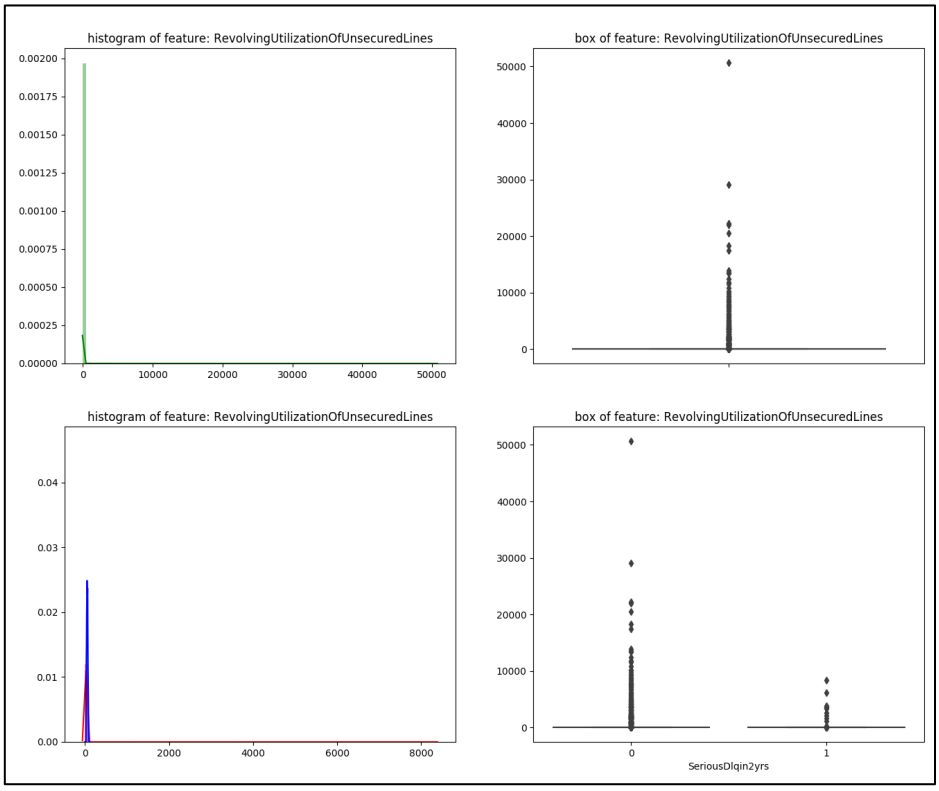

图 1 “贷款以及信用卡可用额度与总额度比例”特征分布

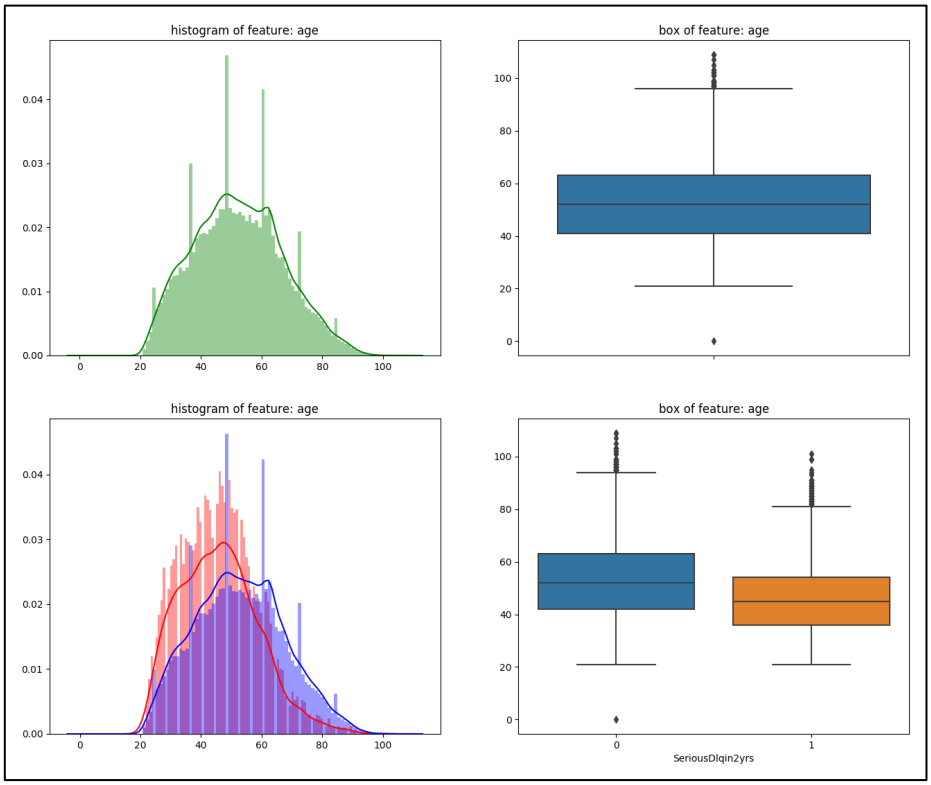

图 2 “借款人借款年龄”特征分布

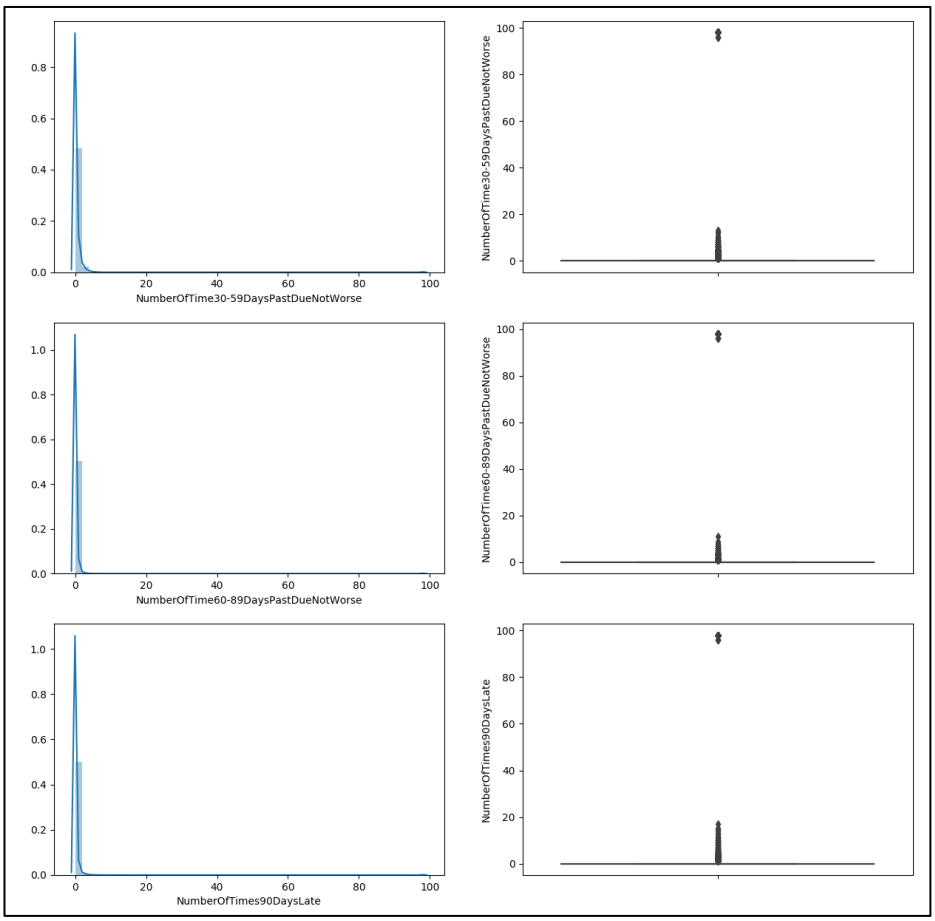

图 3 “35-59 天逾期”特征分布

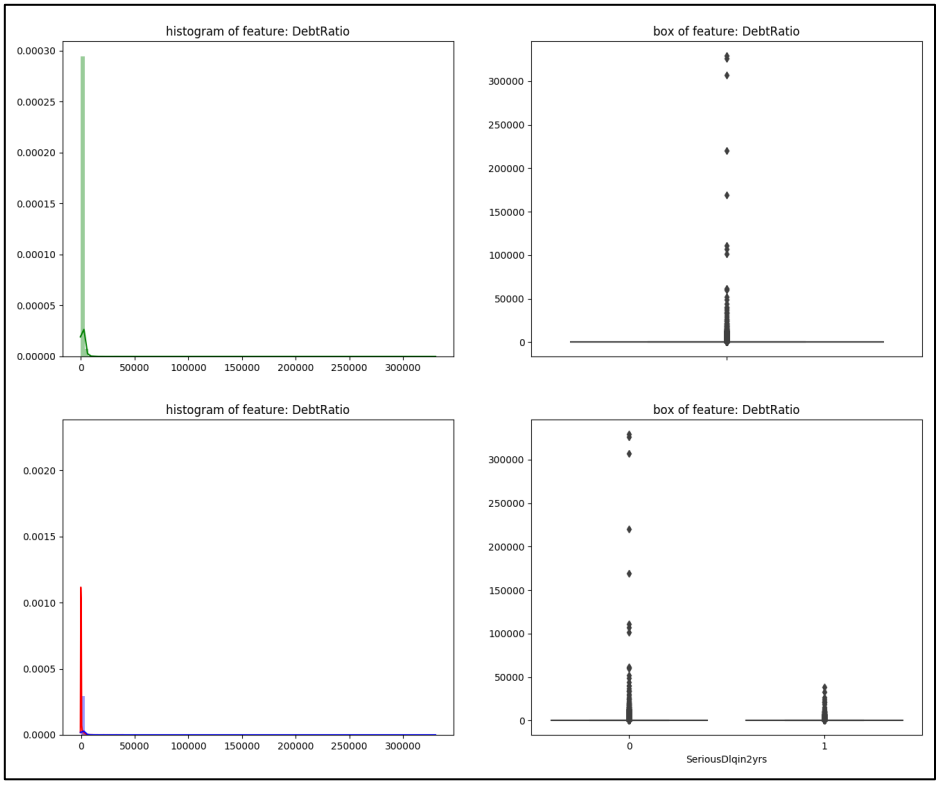

图 4 “负债率”特征分布

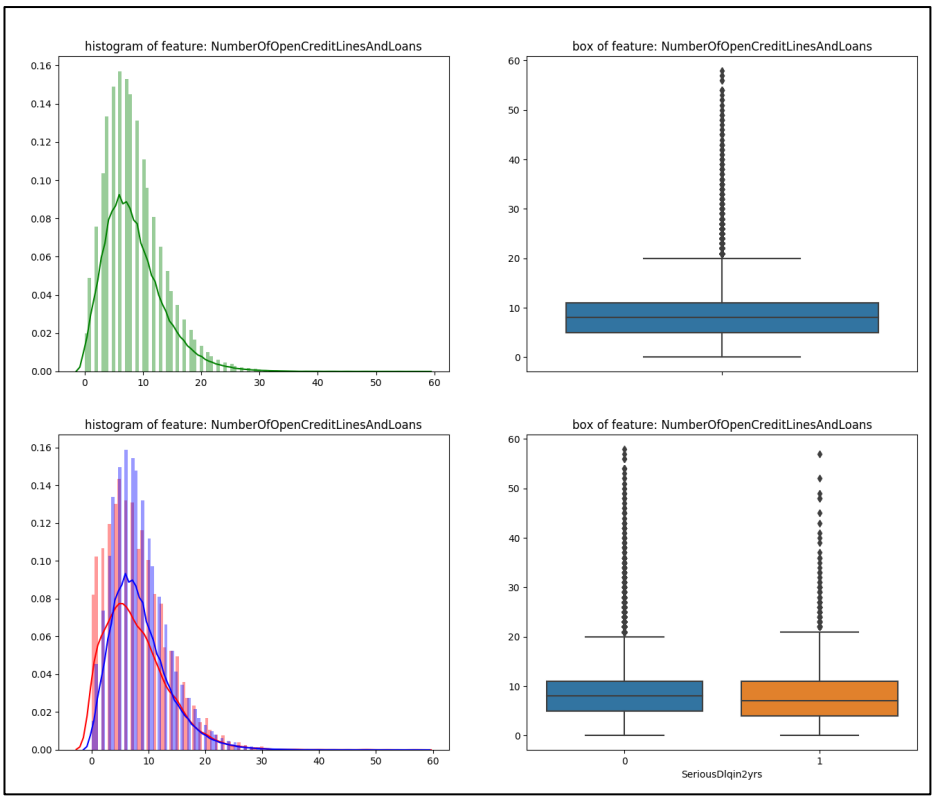

图 5 “信贷数量”特征分布

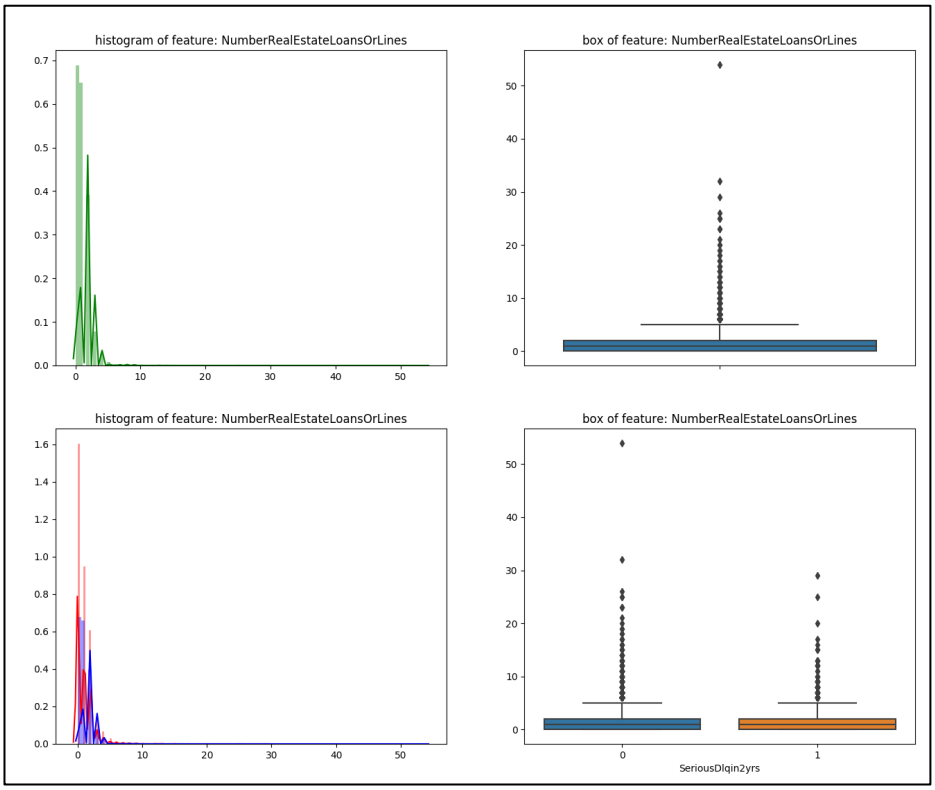

图 6 “固定资产贷款数量”特征分布

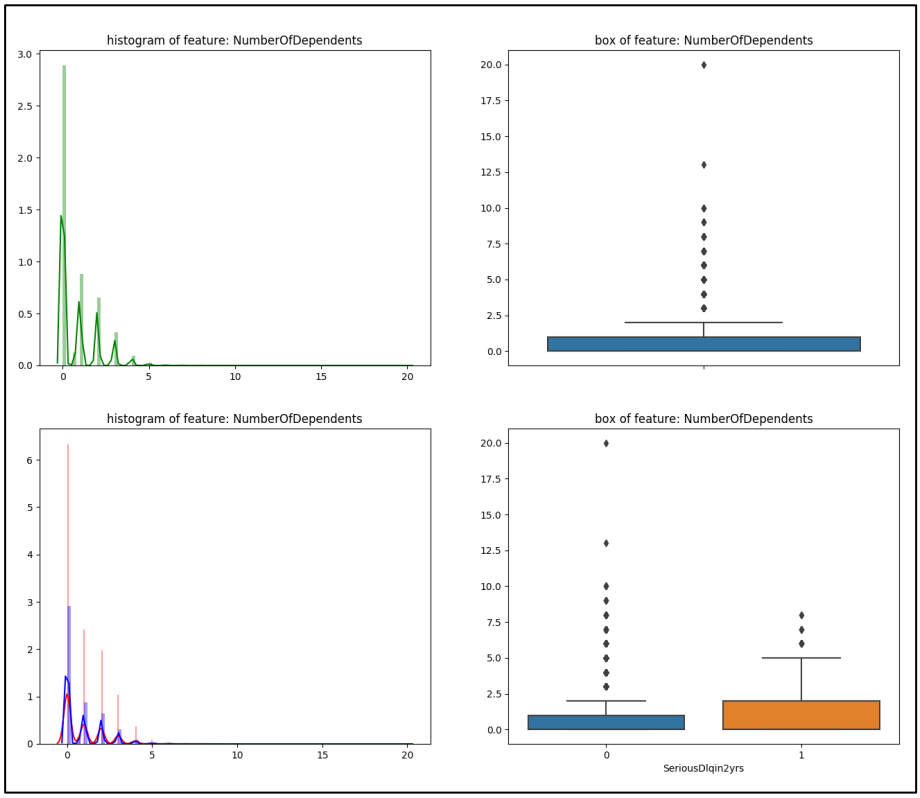

图 7 “家属数量”特征分布

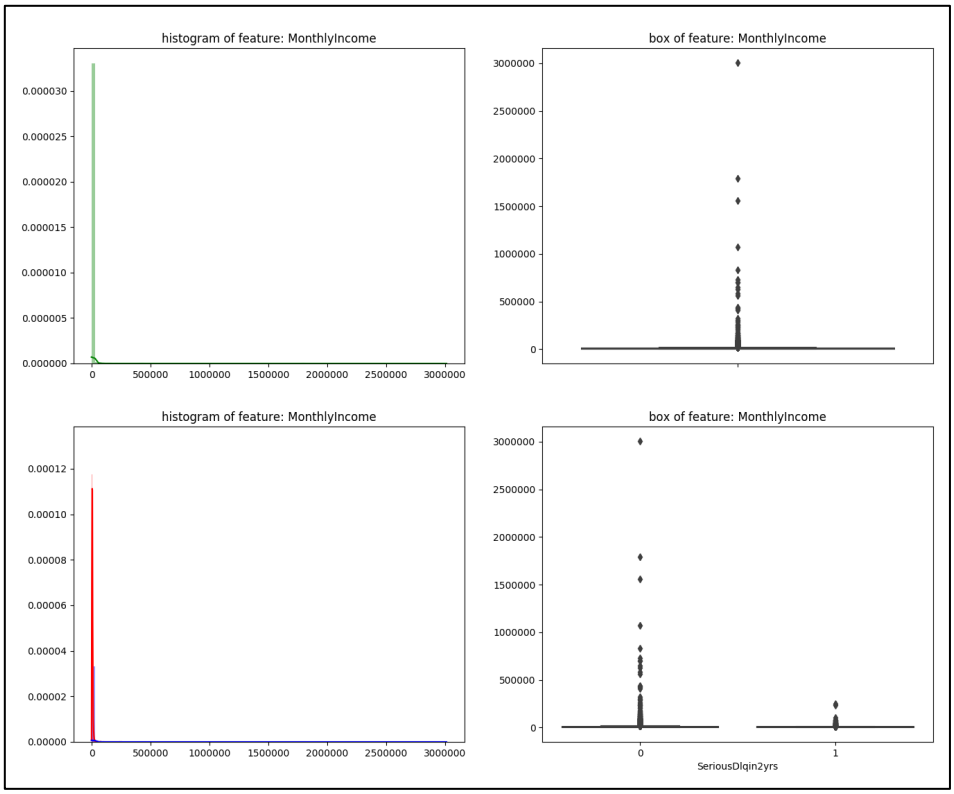

图 8 “月收入”特征分布

2、 IV 曲线

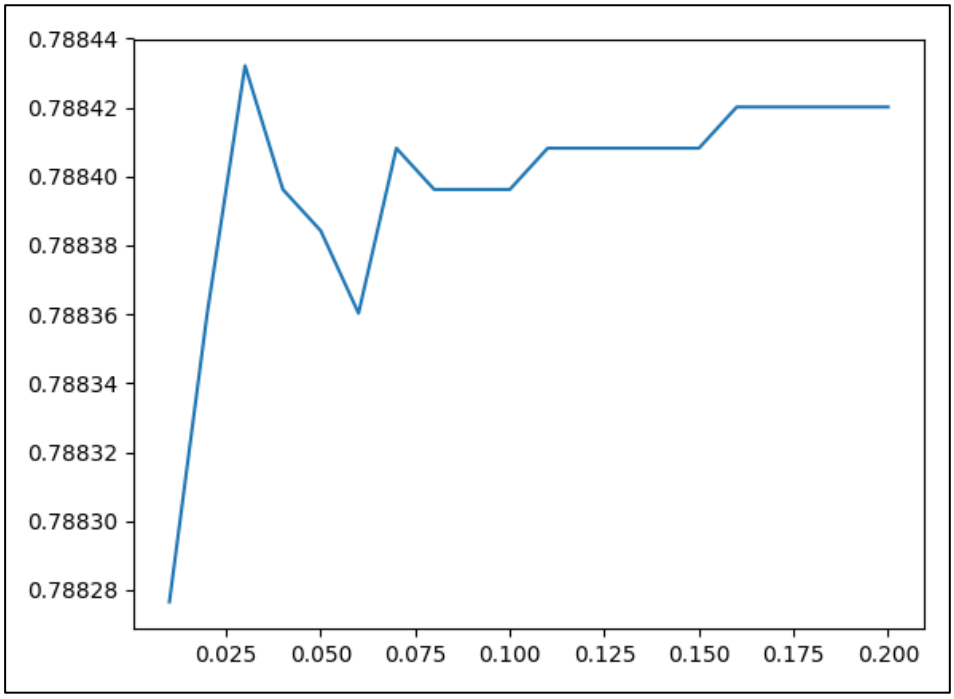

图 9　IV 曲线

3、 ROC 曲线

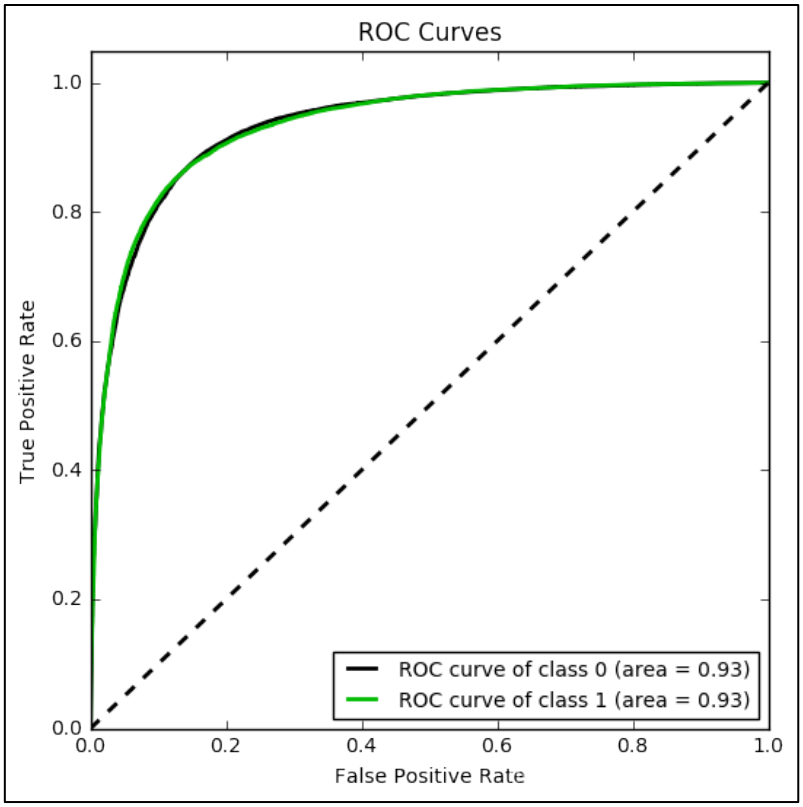

图 10　ROC 曲线

4、 评分表

base_score		[481.94380014]	
RevolvingUtilizationOfUnsecuredLines	**Score**	**NumberOfOpenCreditLinesAndLoans**	**Score**
(-inf, 0.0991]	48.05348668	(-inf, 1.0]	-13.21494743
(0.0991, 0.298]	14.59787756	(1.0, 3.0]	-5.308200853
(0.298, 0.465]	-2.743470154	(3.0, 5.0]	-0.814461429
(0.465, 0.983]	-23.44399694	(5.0, 17.0]	1.942689809
(0.983, 1.0]	-10.45329111	(17.0, inf]	7.36221216
(1.0, inf]	-44.39028056	**NumberOfTimes90DaysLate**	**Score**
age	**Score**	(-inf, 0.0]	3.891191119
(-inf, 36.0]	-4.390866932	(0.0, 1.0]	-28.94458677
(36.0, 54.0]	-2.347737938	(1.0, 2.0]	-37.22455044
(54.0, 61.0]	2.064923041	(2.0, inf]	-39.59109067
(61.0, 64.0]	6.497251785	**NumberRealEstateLoansOrLines**	**Score**
(64.0, 74.0]	9.724825722	(-inf, 0.0]	-9.984194527
(74.0, inf]	14.59644492	(0.0, 1.0]	4.997773612
NumberOfTime30-59DaysPastDueNotWorse	**Score**	(1.0, 2.0]	15.55993259
(-inf, 0.0]	7.198624683	(2.0, 4.0]	9.651750425
(0.0, 1.0]	-17.79234558	(4.0, inf]	-7.361546875
(1.0, 2.0]	-28.08110081	**NumberOfTime60-89DaysPastDueNotWorse**	**Score**
(2.0, inf]	-31.53149945	(-inf, 0.0]	0.991626713
DebtRatio	**Score**	(0.0, 1.0]	-11.05162419
(-inf, 0.0175]	23.75490051	(1.0, 2.0]	-14.06453689
(0.0175, 0.503]	-0.180562005	(2.0, inf]	-14.3584134
(0.503, 1.467]	-7.372258856	**NumberOfDependents**	**Score**
(1.467, inf]	2.74638772	(-inf, 0.0]	15.76124431
MonthlyIncome	**Score**	(0.0, 1.0]	-12.24999838
(-inf, 0.11]	8.389433846	(1.0, 2.0]	-12.82114305
(0.11, 5590.0]	-1.381689179	(2.0, inf]	-11.45981625
(5590.0, inf]	1.310777761		

图 11　评分卡

十、实验结论

本实验通过应用逻辑回归模型，结合 sklearn 库完成了评分卡预测信用风险的逻辑回归模型及其可视化。得到了的评分卡数据可用来自动对申请者进行信用风险评分。该模型逻辑回归准确率大致为 86.6%且 ROC 曲线吻合度较高。

十一、总结及心得体会

思考题：分析模型中用到的梯度下降算法，分析该算法的优略，并提出改进模型和实现代码。

解答：

回归问题的常规步骤为：

(1) 寻找 *h* 函数（预测函数）：

Logistic 回归实际上是一种分类方法，主要用于两分类问题（即输出只有两种，分别代表两个类别），所以利用了 Logistic 函数（或称为 Sigmoid 函数），函数形式为：

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

对于线性边界的情况，边界形式如下（n：特征个数；m：样本数）：

$$\theta_0+\theta_1x_1+\cdots+\theta_nx_n=\sum_{i=1}^{n}\theta_ix_i=\theta^Tx$$

构造预测函数为：

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

函数 $h_\theta(x)$ 的值有特殊的含义，它表示结果取 1 的概率，因此对于输入 x 分类结果为类别 1 和类别 0 的概率分别为：

$$\begin{aligned}P\left(y=1\middle|x;\theta\right)&=h_\theta(x)\\P\left(y=0\middle|x;\theta\right)&=1-h_\theta(x)\end{aligned}\qquad(1)$$

(2) 构造 *J* 函数（损失函数）：

J 函数如下，它是基于最大似然估计推导得到的。

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m}y_i\log h_\theta(x_i)+(1-y_i)\log(1-h_\theta(x_i))\right]$$

(1)式综合起来可以写成：

$$P\left(y\middle|x;\theta\right)=(h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

取似然函数为：

$$L(\theta)=\prod_{i=1}^{m}P\left(y_i\left|x_i;\theta\right.\right)=\prod_{i=1}^{m}(h_\theta(x_i))^{y_i}(1-h_\theta(x_i))^{1-y_i}$$

对数似然函数为：

$$l(\theta)=\log L(\theta)=\sum_{i=1}^{m}(y_i\log h_\theta(x_i)+(1-y_i)\log(1-h_\theta(x_i)))$$

最大似然估计就是求使$l(\theta)$取最大值时的θ，将$J(\theta)$取为下式，即：

$$J(\theta)=-\frac{1}{m}l(\theta)$$

因为乘了一个负的系数$-1/m$，所以取最小值时的θ为要求的最佳参数。

(3) 梯度下降法使 *J* 函数最小并求得回归参数（θ）：

θ更新过程：

$$\theta_j:=\theta_j-\alpha\frac{\delta}{\delta_{\theta_j}}J(\theta)$$

θ更新过程可以写成：

$$\theta_j:=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i)x_i^j$$

重复至收敛。

优点：仅需要很少的迭代次数即可收敛。

缺点：当样本数据量很大时，更新一次的时间较长。

改进模型：

随机梯度下降法：从样本中随机抽出一组，训练后按梯度更新一次，然后再抽取一组，再更新一次，在样本量及其大的情况下，可能不用训练完所有的样本就可以获得一个损失值在可接受范围之内的模型了。

实现代码：

```
#-*- coding: utf-8 -*-
import random
#用 y = 01*x1 + 02*x2 来拟合下面的输入和输出
#input1  1   2   5   4
#input2  4   5   1   2
#output  19  26  19  20
input_x = [[1,4], [2,5], [5,1], [4,2]]  #输入
y = [19,26,19,20]   #输出
theta = [1,1]       #θ 参数初始化
loss = 10           #loss 先定义一个数，为了进入循环迭代
step_size = 0.01    #步长
eps =0.0001         #精度要求
```

```
max_iters = 10000   #最大迭代次数
error =0            #损失值
iter_count = 0      #当前迭代次数

while( loss > eps and iter_count < max_iters):    #迭代条件
    loss = 0
    i = random.randint(0,3)  #每次迭代在 input_x 中随机选取一组样本进行权重的更新
    pred_y = theta[0]*input_x[i][0]+theta[1]*input_x[i][1] #预测值
    theta[0] = theta[0] - step_size * (pred_y - y[i]) * input_x[i][0]
    theta[1] = theta[1] - step_size * (pred_y - y[i]) * input_x[i][1]
    for i in range (3):
        pred_y = theta[0]*input_x[i][0]+theta[1]*input_x[i][1] #预测值
        error = 0.5*(pred_y - y[i])**2
        loss = loss + error
    iter_count += 1
    print ('iters_count', iter_count)
print ('theta: ',theta )
print ('final loss: ', loss)
print ('iters: ', iter_count)
```

十二、对本实验过程及方法、手段的改进建议

本实验对数据的处理上难度比较大，学生需要花费较多的课下时间来学习相关的数据处理知识并实现。可以考虑换一组数据处理难度较小的样本，使本实验着重实现逻辑回归部分。

报告评分：

指导教师签字：