Logistic回归

Logistic回归（Logit回归）是统计学、机器学习中非常常用的解决分类问题的统计方法。

我们首先介绍二分类问题。

二分类问题指被解释变量为0/1两个取值的情况。比如，我们可能需要判断一封电子邮件是否为垃圾邮件（垃圾邮件=1，否则=0），或者我们需要预测哪些个体会参与到某个项目中（参与=1）、是否会上大学（上大学=1）。

令$\left(y_i, x_i'\right)', i=1,\ldots,N, x_i \in \mathbb{R}^K$，而其中$y_i$为**二元变量**（**binary variable**），即$y_i \in \{0,1\}$，那么其条件期望：

$$\mathbb{E}\left(y_i|x_i\right) = 1 \cdot P\left(y_i|x_i\right) + 0 \cdot \left(1 - P\left(y_i|x_i\right)\right) = P\left(y_i|x_i\right)$$

即条件期望为给定x_i，$y_i=1$的条件概率。然而如果我们使用线性回归，线性函数$x_i'\beta$不能够保证一定在$(0,1)$区间范围以内，因而此时使用线性回归拟合上述条件期望就不再合适。

为了避免以上问题，我们可以将概率$P\left(y_i|x_i\right)$建模为一个概率分布，一个常用的假设是：

$$P\left(y_i=1|x_i,\beta\right) = F\left(x_i'\beta\right) = \frac{e^{x_i'\beta}}{1+e^{x_i'\beta}}$$

由于函数$F\left(x\right) = \frac{e^x}{1+e^x}$为一个Logistic分布的分布函数，因而其函数值一定是单调的且在$(0,1)$之间的。以上模型我们通常称为**逻辑斯蒂回归（Logistic regression）。**

我们将$y_i=1$的概率与$y_i=0$的概率的比值成为**几率**（**odds**），那么根据以上设定，该模型的几率为：

$$odds = \frac{P\left(y_i=1|x_i,\beta\right)}{P\left(y_i=0|x_i,\beta\right)} = \frac{\frac{e^{x_i'\beta}}{1+e^{x_i'\beta}}}{1-\frac{e^{x_i'\beta}}{1+e^{x_i'\beta}}} = e^{x_i'\beta}$$

而**对数几率**（**log odds**，也称为**logit**）为：

$$logit = \ln(odds) = x_i'\beta$$

因而以上模型被称为**对数几率回归**（**Logit regression**）。

为了估计上述模型中的β，我们可以使用条件极大似然法。以上模型的条件密度函数为：

$$f\left(y_i|x_i,\beta\right) = \left[\frac{e^{x_i'\beta}}{1+e^{x_i'\beta}}\right]^{1\{y_i=1\}} \left[\frac{1}{1+e^{x_i'\beta}}\right]^{1\{y_i=0\}}$$

因而极大似然函数为：

$$L(\beta|y,x)=\sum_{i=1}^{N}\left[y_i\ln\left(\frac{e^{x_i'\beta}}{1+e^{x_i'\beta}}\right)+(1-y_i)\ln\left(\frac{1}{1+e^{x_i'\beta}}\right)\right]$$

最大化以上似然函数，就可以得到β的一致估计$\hat{\beta}$。进而得到$p(x_i)=P(y_i=1|x_i)$，即给定x_i，$y_i=1$的概率的估计:

$$\hat{p}_i\triangleq P\widehat{(y_i=1}|x_i)=F\left(x_i'\hat{\beta}\right)$$

In [1]:
```
import pandas as pd

raw_data = pd.read_csv("csv/soep.csv")
raw_data.head()
```

Out[1]:

	persnr	year	employment	chld6	chld16	age	income	husworkhour	husemp
0	9401	2008	[1] Employed 1	0	0	48	50682	1923	
1	9401	2009	[1] Employed 1	0	0	49	45880	2078	
2	9401	2010	[1] Employed 1	0	0	50	48690	2078	
3	9401	2011	[1] Employed 1	0	0	51	52832	2494	
4	9401	2012	[1] Employed 1	0	0	52	55790	2078	

In [2]:
```
import numpy as np

data = raw_data.set_index(['persnr', 'year'])
data['log_income'] = np.log(data['income'])
data['age2'] = np.power(data['age'], 2)
data = data.drop('income', axis=1)
region_dummy = pd.get_dummies(data['region'])
data = pd.concat([data, region_dummy], axis=1)
data = data.drop(['region', '0'], axis=1)
data['employment'] = data['employment'] == data['employment'].iloc[0]
data.head()
```

Out[2]:

		employment	chld6	chld16	age	husworkhour	husemployment	ed
persnr	year							
9401	2008	True	0	0	48	1923	1	13
	2009	True	0	0	49	2078	1	13
	2010	True	0	0	50	2078	1	13
	2011	True	0	0	51	2494	1	13
	2012	True	0	0	52	2078	1	13

In [3]:

```
y = data['employment']
X = data.drop(['employment'], axis=1)
X = (X-X.mean())/X.std()
y.mean()
```

Out[3]: 0.8260999254287845

In [4]:

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(penalty=None, max_iter=1000)
LR.fit(X, y)  ## 训练模型
LR.coef_
```

Out[4]:

```
array([[-0.58690665, -0.29256672,  2.09002785,  0.035343  ,  0.11277509,
         0.38330784, -0.15366163, -0.14120281, -2.22864429,  0.21405959]])
```

值得注意的是我们特地加了一个选项「penalty='none'」，即没有惩罚项，做普通的Logistic回归。默认情况会使用L2正则化，也可以选择L1正则化，这里需要注意。

模型训练好之后，当然可以计算预测概率：

In [5]:

```
LR.predict_proba(X)
```

Out[5]:

```
array([[0.10672471, 0.89327529],
       [0.11223939, 0.88776061],
       [0.12215972, 0.87784028],
       ...,
       [0.17046975, 0.82953025],
       [0.15517853, 0.84482147],
       [0.14500921, 0.85499079]])
```

注意以上的概率有两列，分别是按照可能的结果排序的（False,True）的概率值，我们希望得到True的概率，所以：

In [6]:

```
data['prob'] = LR.predict_proba(X)[:, 1]
data
```

Out[6]:

		employment	chld6	chld16	age	husworkhour	husemployment
persnr	**year**						
9401	**2008**	True	0	0	48	1923	1
	2009	True	0	0	49	2078	1
	2010	True	0	0	50	2078	1
	2011	True	0	0	51	2494	1
	2012	True	0	0	52	2078	1
...	**...**	...	...	...	...	...	...
8270802	**2008**	True	0	0	23	2078	1
	2009	True	0	0	24	2078	1
	2010	True	0	0	25	2078	1
	2011	True	0	0	26	2078	1
	2012	True	0	0	27	2078	1

6705 rows × 12 columns

In [7]:

```
data['pred'] = LR.predict(X)
data
```

Out[7]:

		employment	chld6	chld16	age	husworkhour	husemployment
persnr	**year**						
9401	**2008**	True	0	0	48	1923	1
	2009	True	0	0	49	2078	1
	2010	True	0	0	50	2078	1
	2011	True	0	0	51	2494	1
	2012	True	0	0	52	2078	1
...	**...**	...	...	...	...	...	...
8270802	**2008**	True	0	0	23	2078	1
	2009	True	0	0	24	2078	1
	2010	True	0	0	25	2078	1
	2011	True	0	0	26	2078	1
	2012	True	0	0	27	2078	1

6705 rows × 13 columns

Logistic回归的模型评价

常用的模型评价指标

对于Logit回归：类比于线性回归中的R^2，McFadden(1974)建议使用Pseudo R^2:

$$R^2 = 1 - \frac{L\left(\hat{\beta}|y,x\right)}{L_0} = 1 - \frac{\sum_{i=1}^{N}\left[1\left\{y_i=1\right\}\ln(\hat{p}_i) + 1\left\{y_i=0\right\}\ln(1-\hat{p}_i)\right]}{N\left[\bar{y}\ln\bar{y} + (1-\bar{y})\ln(1-\bar{y})\right]}$$

此外，为了进行预测分类，我们可以选定一个临界值c，并令预测值$\hat{y}_i = 1\left\{\hat{p}_i > c\right\}$，之后将样本分为四类：

- 真正（True Positive， TP）：$y_i = 1, \hat{y}_i = 1$;
- 假正（False Positive， FP）：$y_i = 0, \hat{y}_i = 1$;
- 真反（True Negative， TN）：$y_i = 0, \hat{y}_i = 0$;
- 假反（False Negative， FN）：$y_i = 1, \hat{y}_i = 0$。

使用以上四个分类分别定义：

- 查准率（precision），即所有预测为正的样本中，正确的比例：

$$Precision = \frac{TP}{TP+FP}$$

- 查全率（或者召回率，recall），即所有正的样本中，正确的比例：

$$Recall = \frac{TP}{TP + FN}$$

- 精度（accuracy），即所有样本中预测正确的比例：

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- F1度量，即查准率和查全率的调和平均：

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{1}{\frac{1}{2}\left(\frac{1}{Precision} + \frac{1}{Recall}\right)}$$

查准率和查全率之间通常存在着权衡：

- 比如，如果我们希望提高查准率，需减少预测为正的比例，需要较大的c，从而降低查全率。
- 使得查准率等于查全率的点称为平衡点（break-event point, BEP）。

In [8]:

```
TP = np.sum(data['employment'] & data['pred'])
TN = np.sum((~data['employment'] & (~data['pred'])))
FP = np.sum((~data['employment'] & (data['pred'])))
FN = np.sum((data['employment']) & (~data['pred']))

print("TP=", TP)
print("TN=", TN)
print("FP=", FP)
print("FN=", FN)
print("查全率=敏感性=", TP / (TP + FN))
print("查准率=", TP / (TP + FP))
print("特异性=", TN / (TN + FP))
```

```
TP= 5507
TN= 49
FP= 1117
FN= 32
查全率=敏感性= 0.9942227838960102
查准率= 0.8313707729468599
特异性= 0.04202401372212693
```

此外，我们还可以使用ROC曲线：

对于任意的临界值c，我们都可以定义：

- 敏感性（sensitivity）：观察到的正的样本中，预测正确的比例，即

$$Sensitivity = Recall = \frac{TP}{TP + FN}$$

- 特异性（specificity）：观察到的反的样本中，预测正确的比例，即

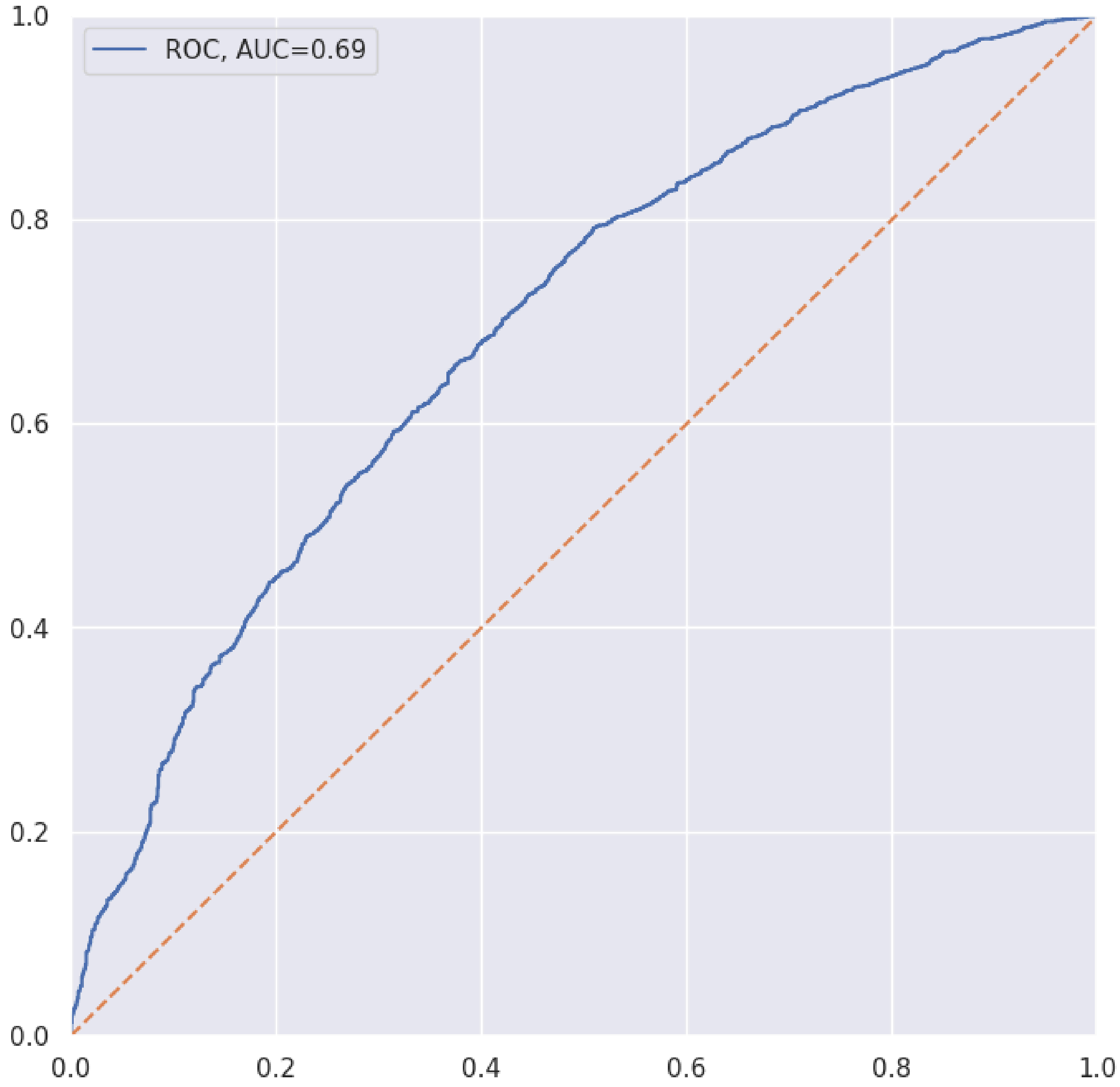
$$Specificity = \frac{TN}{TN + FP}$$

受试者工作特征曲线（receiver operating characteristic curve， **ROC curve**）：即当$c \in [0,1]$时，以$1 - Specificity$作为横坐标，以$Sensitivity$作为纵坐标所画出来的图。

当$c = 0$时，所有的$\hat{y}_i = 1$，因而$Sensitivity = 1$，$1 - Specificity = 1$；当$c = 1$时，所有的$\hat{y}_i = 0$，因而$Sensitivity = 0$，$1 - Specificity = 0$，因而ROC曲线从$(0,0)$出发，到$(1,1)$终止。一个理想的模型应该是$1 - Specificity$很小，同时$Sensitivity$很大，因而ROC曲线越向$(0,1)$弯曲，表明模型的预测能力越好。

为此，一个度量模型预测能力的指标即计算**ROC曲线的线下面积**（**area under ROC curve, AUC**），AUC越大，则模型的预测能力越强。

In [9]:
```
from sklearn.metrics import roc_curve, auc

fpr, tpr, threshold = roc_curve(data['employment'], data['prob'])
roc_auc = auc(fpr, tpr)
import matplotlib.pyplot as plt
import seaborn as sb

sb.set()
%matplotlib inline
plt.rcParams['figure.figsize'] = (8.0, 8.0)
plt.plot(fpr, tpr, label='ROC, AUC=%.2f' % roc_auc)
plt.legend(loc='upper left', frameon=True)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
```



样本外预测指标

同样的，在预测问题中，我们应该更加关注样本外预测能力，我们可以通过划分训练集、测试集的方式计算模型样本外预测能力，或者

In [10]:

```
X['employment'] = y
## 产生一个随机顺序，并排序，从而顺序是随机的
X['random_order'] = np.random.random(X.shape[0])
X = X.sort_values(['random_order'])
X = X.drop('random_order', axis=1)
X
```

Out[10]:

persnr	year	chld6	chld16	age	husworkhour	husemployment	
8015301	**2011**	-0.522012	-0.809605	1.067188	-3.089472	-4.948802	-0
22602	**2010**	-0.522012	-0.809605	0.607372	0.171100	0.202039	-0
3479101	**2009**	1.915378	-0.809605	-1.691705	0.388657	0.202039	-0
109702	**2012**	-0.522012	-0.809605	1.373732	-0.264015	0.202039	-0
13402	**2012**	-0.522012	-0.809605	1.373732	0.316138	0.202039	-0
...	**...**	...	...	...	...	...	
5504302	**2010**	-0.522012	0.350062	-0.618802	-0.046458	0.202039	-0
2937702	**2011**	-0.522012	-0.809605	0.913916	-1.640484	0.202039	-0
2673602	**2008**	-0.522012	-0.809605	-0.158987	0.193413	0.202039	-0
831602	**2012**	-0.522012	0.350062	0.147557	0.752648	0.202039	-0
2931401	**2010**	-0.522012	-0.809605	0.147557	0.126473	0.202039	-0

6705 rows × 11 columns

In [11]:

```python
from sklearn.model_selection import KFold
## 区分训练集和测试集
kf = KFold(n_splits=10)
## 使用训练集回归
CV_prob = np.array([])
for train, test in kf.split(X):
    X_train = X.iloc[train, :]
    X_test = X.iloc[test, :]
    # 先使用训练集训练模型
    logit = LogisticRegression(penalty=None, max_iter=1000).fit(
        X_train.drop('employment', axis=1), X_train['employment'])
    # 接下来在验证集上进行预测，得到预测概率
    pre_prob = logit.predict_proba(X_test.drop('employment', axis=1))[:, 1]
    CV_prob = np.concatenate([CV_prob, pre_prob])
CV_prob
```

Out[11]:

```
array([0.7699432 , 0.87403168, 0.77535298, ..., 0.9003344 , 0.88456438,
       0.91730594])
```

以$c = 0.6$进行预测：

In [12]:

```python
CV_pred = CV_prob >= 0.6
CV_pred
```

Out[12]:

```
array([ True,  True,  True, ...,  True,  True,  True])
```

计算指标：

In [13]:

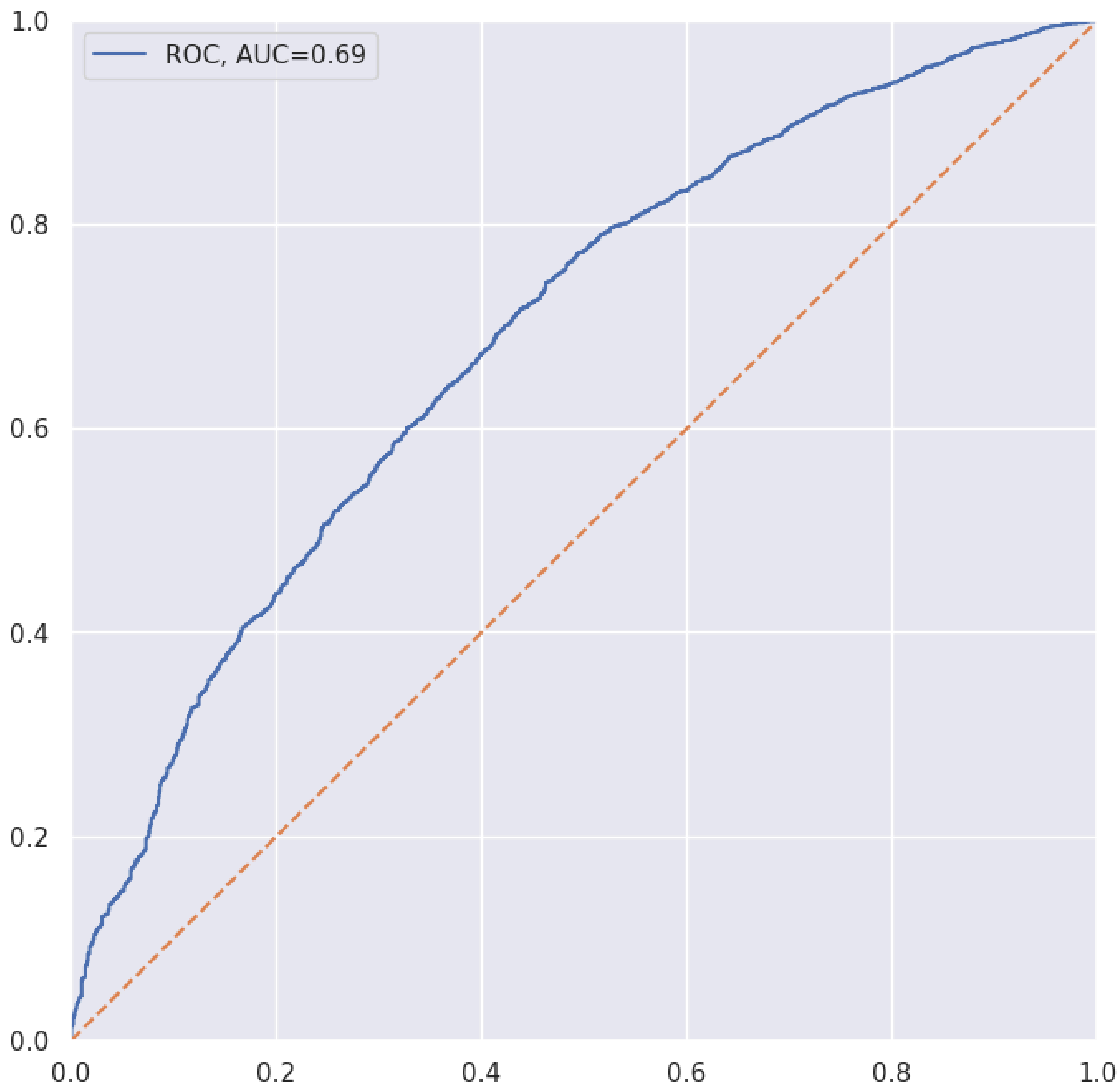
```python
TP = np.sum(X['employment'] & CV_pred)
TN = np.sum((~X['employment'] & (~CV_pred)))
```

```python
FP = np.sum((~X['employment'] & (CV_pred)))
FN = np.sum((X['employment']) & (~CV_pred))

print("TP=", TP)
print("TN=", TN)
print("FP=", FP)
print("FN=", FN)
print("查全率=敏感性=", TP / (TP + FN))
print("查准率=", TP / (TP + FP))
print("特异性=", TN / (TN + FP))
```

```
TP= 5350
TN= 157
FP= 1009
FN= 189
查全率=敏感性= 0.9658783173858098
查准率= 0.8413272527126907
特异性= 0.1346483704974271
```

以及ROC曲线:

In [14]:
```python
fpr, tpr, threshold = roc_curve(X['employment'], CV_prob)
roc_auc = auc(fpr, tpr)
import matplotlib.pyplot as plt
import seaborn as sb

sb.set()
%matplotlib inline
plt.rcParams['figure.figsize'] = (8.0, 8.0)
plt.plot(fpr, tpr, label='ROC, AUC=%.2f' % roc_auc)
plt.legend(loc='upper left', frameon=True)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
```



注意到我们的数据实际上是一个面板数据，上面的Cross-Validation过程中，我们把每年每个个人看成是一个独立的个人进行抽取的，这种方法并不是非常好。一个更好的方法是按照个体进行划分，我们可以使用unstack()方法先将面板数据变换成「宽」的格式，然后使用KFold进行划分，进行回归时再使用stack()方法将其变回「长」的格式，如此就可以保证在Cross-Validation时不存在不同分组中存在相同个体的不同年份的数据了。

In [15]:

```
X = X.sort_index()
X = X.unstack()
X['random_order'] = np.random.random(X.shape[0])
X = X.sort_values(['random_order'])
X = X.drop('random_order', axis=1)
X
```

Out[15]:

	chld6						
year	2008	2009	2010	2011	2012	2008	20(
persnr							
5173002	-0.522012	-0.522012	-0.522012	-0.522012	-0.522012	-0.809605	-0.8096(
1176402	1.915378	1.915378	1.915378	1.915378	1.915378	0.350062	0.3500(
7003702	-0.522012	-0.522012	-0.522012	-0.522012	-0.522012	1.509729	0.3500(
7255202	1.915378	1.915378	-0.522012	-0.522012	-0.522012	0.350062	0.3500(
2750501	-0.522012	1.915378	1.915378	1.915378	1.915378	-0.809605	-0.8096(
...	...	...	...	...	...	...	
5683402	1.915378	1.915378	1.915378	1.915378	-0.522012	-0.809605	0.3500(
2888802	-0.522012	-0.522012	-0.522012	-0.522012	-0.522012	-0.809605	-0.8096(
3256602	-0.522012	-0.522012	-0.522012	-0.522012	-0.522012	0.350062	0.3500(
652103	-0.522012	-0.522012	-0.522012	-0.522012	-0.522012	-0.809605	-0.8096(
3406302	1.915378	1.915378	-0.522012	-0.522012	-0.522012	0.350062	0.3500(

1341 rows × 55 columns

In [16]:

```
from sklearn.model_selection import KFold
## 区分训练集和测试集
kf = KFold(n_splits=10)
## 使用训练集回归
CV_prob = np.array([])
for train, test in kf.split(X):
    X_train = X.iloc[train, :].stack(1,future_stack=True)
    X_test = X.iloc[test, :].stack(1,future_stack=True)
    # 先使用训练集训练模型
    logit = LogisticRegression(penalty=None, max_iter=1000).fit(
        X_train.drop('employment', axis=1), X_train['employment'])
    # 接下来在验证集上进行预测，得到预测概率
    pre_prob = logit.predict_proba(X_test.drop('employment', axis=1))[:, 1]
    CV_prob = np.concatenate([CV_prob, pre_prob])
CV_prob
```

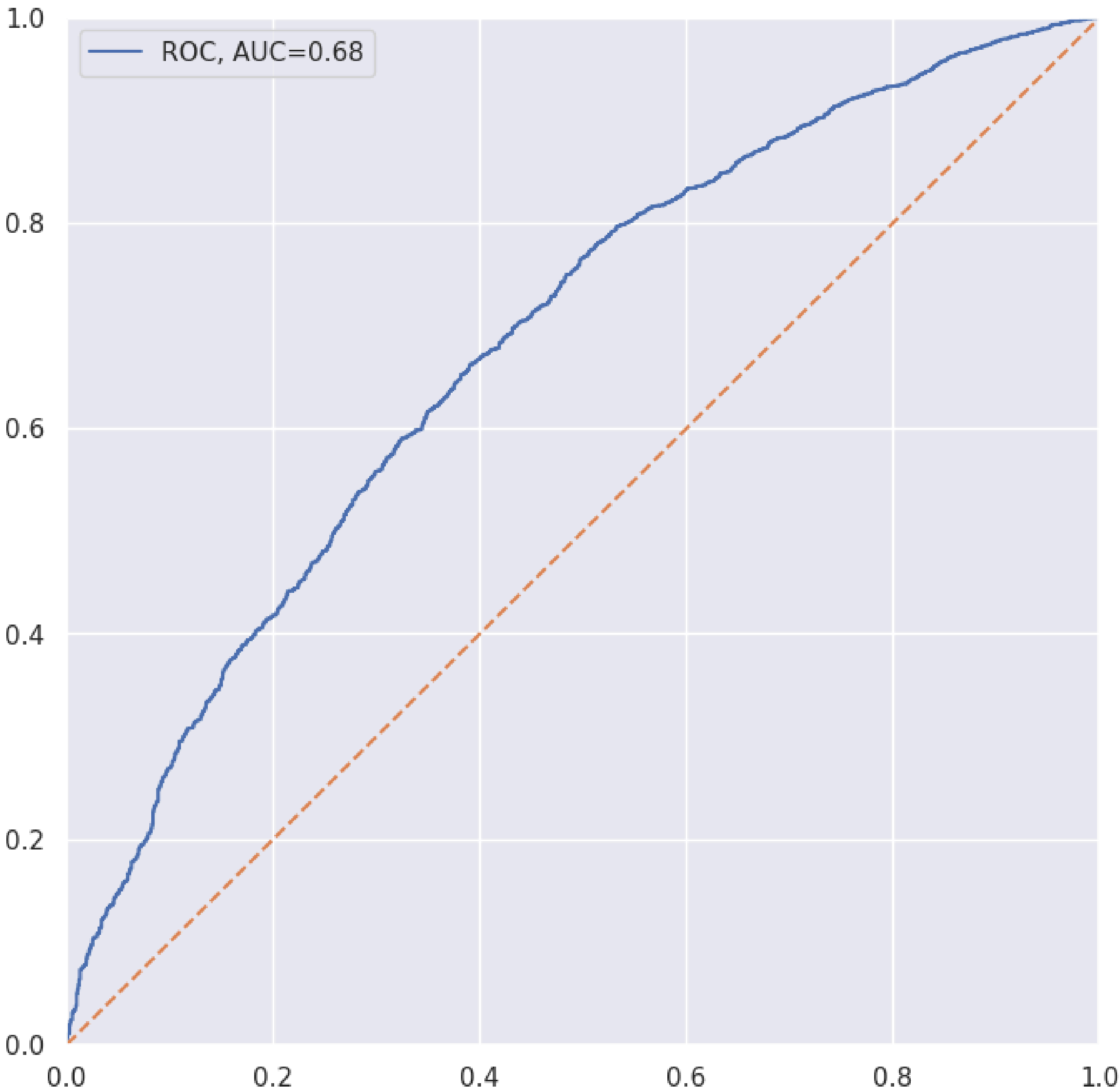
Out[16]:

```
array([0.95379662, 0.94459642, 0.939785  , ..., 0.80077777, 0.78926548,
       0.77315889])
```

In [17]:

```
CV_pred = CV_prob >= 0.5
X = X.stack(1,future_stack=True)
fpr, tpr, threshold = roc_curve(X['employment'], CV_prob)
roc_auc = auc(fpr, tpr)
import matplotlib.pyplot as plt
import seaborn as sb

sb.set()
%matplotlib inline
plt.rcParams['figure.figsize'] = (8.0, 8.0)
plt.plot(fpr, tpr, label='ROC, AUC=%.2f' % roc_auc)
plt.legend(loc='upper left', frameon=True)
```

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
```



多元Logistic

以上我们解决了预测目标为二分的情况（0/1），Logistic回归可以轻易的推广到多个选择的情况，即多元Logistic（multinomial Logistic）。

如果可供选择的选项有$J+1$种：$y \in \{0,1,2,\ldots,J\}$，选取$y=0$为基准选项

给定特征x，假定选择的概率为：

$$P(y=j|x)=\frac{e^{x'\beta_j}}{1+\sum_{j=1}^{J} e^{x'\beta_j}}, j=1,\ldots,J$$

$$P(y=0|x)=\frac{1}{1+\sum_{j=1}^{J} e^{x'\beta_j}}$$

此时，有J个β_j需要估计，我们同样可以使用极大似然估计对参数进行估计。

可以发现，以上介绍的Logistic回归实际上是$J=1$的一个特例，即一个简单的二元Logistic回归。

在Scikit-Learn中，使用多元Logistic回归的方法与二元的Logistic是一样的，只需要提供的y中有多于两个选项，就会自动帮我们做多元Logistic。

比如如下代码中我们使用教育情况、年龄、性别等特征预测每个人的雇主性质：

In [18]:

```
import pandas as pd

raw_data = pd.read_csv('csv/cfps_adult.csv')
raw_data = raw_data.loc[:, ['cfps_birthy', 'cfps_gender', 'te4', 'qg2']]
raw_data.head()
```

Out[18]:

	cfps_birthy	cfps_gender	te4	qg2
0	1969	0	-8	77
1	1966	1	-8	3
2	1981	1	-8	4
3	1990	0	-8	4
4	1988	0	-8	2

In [19]:

```
# 生成年龄
raw_data['age'] = 2014 - raw_data['cfps_birthy']
raw_data['age2'] = raw_data['age']**2
raw_data = raw_data.drop('cfps_birthy', axis=1)
# 产生教育的虚拟变量
edu_dummies = pd.get_dummies(raw_data['te4'], prefix='edu')
raw_data = pd.concat([raw_data, edu_dummies], axis=1)
raw_data = raw_data.drop('te4', axis=1)
# 只保留政府部门、事业单位、国有企业和私营企业
data = raw_data[(raw_data['qg2'] > 0) & (raw_data['qg2'] <= 4)]
data.head()
```

Out[19]:

	cfps_gender	qg2	age	age2	edu_-8	edu_-1	edu_1	edu_2	edu_3	edu_4	ed
1	1	3	48	2304	True	False	False	False	False	False	Fa
2	1	4	33	1089	True	False	False	False	False	False	Fa
3	0	4	24	576	True	False	False	False	False	False	Fa
4	0	2	26	676	True	False	False	False	False	False	Fa
5	0	4	27	729	True	False	False	False	False	False	Fa

接下来可以直接使用如上的LogisticRegression：

In [20]:

```
LR = LogisticRegression(penalty=None,
                        max_iter=1000).fit(data.drop('qg2', axis=1),
                                           data['qg2'])  ## 训练模型
LR.coef_
```

```
/opt/Anaconda/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[20]:

```
array([[ 2.21685347e-01, -4.34239750e-02,  6.49283216e-04,
         1.48964554e-01, -1.07812489e-02, -1.05379934e-01,
        -3.15502620e-01, -4.08068832e-01, -1.65478911e-01,
         9.99331454e-02,  1.42546279e-01,  1.85956047e-02],
       [-5.12363705e-01,  2.26653986e-02, -2.11787029e-04,
         8.77202585e-02, -2.18551320e-02, -1.32499018e-01,
        -3.90250870e-01, -5.48203670e-01, -2.73854211e-01,
         1.89331162e-01,  3.16935100e-01,  3.80582348e-02],
       [ 4.61027293e-01,  3.91066929e-02, -4.66341806e-04,
        -4.24380679e-01, -7.05291550e-03, -1.55736006e-01,
        -2.29087028e-01, -3.22680951e-01,  4.55982794e-02,
         2.28115938e-01,  1.12536819e-01,  2.93849835e-02],
       [-1.70348935e-01, -1.83481164e-02,  2.88456195e-05,
         1.87695866e-01,  3.96892965e-02,  3.93614959e-01,
         9.34840518e-01,  1.27895345e+00,  3.93734843e-01,
        -5.17380246e-01, -5.72018197e-01, -8.60388231e-02]])
```

以及预测概率，注意因为现在有4个选项，从而预测的概率有四列：

In [21]:

```
prob = LR.predict_proba(data.drop('qg2', axis=1))
prob
```

Out[21]:

```
array([[0.07850318, 0.10116812, 0.19875606, 0.62157264],
       [0.05966459, 0.08122662, 0.16989868, 0.68921011],
       [0.04153048, 0.10107933, 0.07849923, 0.77889097],
       ...,
       [0.05802513, 0.07381486, 0.15369223, 0.71446778],
       [0.04161228, 0.10824505, 0.08494107, 0.7652016 ],
       [0.05918625, 0.07976412, 0.1668571 , 0.69419253]])
```

可以使用如上概率计算各种指标，比如画ROC曲线：

In [22]:

```
fpr1, tpr1, threshold1 = roc_curve(data['qg2'] == 1, prob[:, 0])
roc_auc1 = auc(fpr1, tpr1)
fpr2, tpr2, threshold2 = roc_curve(data['qg2'] == 2, prob[:, 1])
roc_auc2 = auc(fpr2, tpr2)
fpr3, tpr3, threshold3 = roc_curve(data['qg2'] == 3, prob[:, 2])
roc_auc3 = auc(fpr3, tpr3)
fpr4, tpr4, threshold4 = roc_curve(data['qg2'] == 4, prob[:, 3])
roc_auc4 = auc(fpr4, tpr4)
import matplotlib.pyplot as plt
import seaborn as sb

sb.set()
%matplotlib inline
plt.rcParams['figure.figsize'] = (8.0, 8.0)
plt.plot(fpr1, tpr1, label='ROC1, AUC=%.2f' % roc_auc1)
plt.plot(fpr2, tpr2, label='ROC2, AUC=%.2f' % roc_auc2)
plt.plot(fpr3, tpr3, label='ROC3, AUC=%.2f' % roc_auc3)
plt.plot(fpr4, tpr4, label='ROC4, AUC=%.2f' % roc_auc4)
```

```
plt.legend(loc='upper left', frameon=True)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
```