Table of Contents

In [1]:

```
jupyter nbconvert to webpdf allow-chromium-download Python_Finaltest.ipynb
```

```
  File "C:\Users\17800\AppData\Local\Temp/ipykernel_2268/1449648151.py", line 1
    jupyter nbconvert to webpdf allow-chromium-download Python_Finaltest.ipynb
            ^
SyntaxError: invalid syntax
```

数据读取

In [164…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys
import warnings
import statsmodels.api as sm
import lightgbm as lgb
import xgboost
import tensorflow as tf
from sklearn.datasets import fetch_california_housing
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import learning_curve
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import *
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from statsmodels.base.model import GenericLikelihoodModel
from statsmodels.genmod.families import Binomial
from scipy.special import gammaln as lgamma
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from scipy.interpolate import make_interp_spline
import plotly.graph_objs as go
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings('ignore')
%matplotlib inline
%config InlineBackend.figure_format = 'retina' # 高清图
```

In [2]:

```
import plotly.io as pio
pio.renderers.default='notebook'
```

In [3]:

```
from bokeh.plotting import figure,show,output_notebook
from bokeh.models import ColumnDataSource
from bokeh.palettes import Spectral6
```

In [4]:

```
output_notebook()
```

BokehJS 2.4.1 successfully loaded.

In [5]:

```
# 比较好用的一个办法
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
#InteractiveShell.ast_node_interactivity = "last_expr"
from IPython.display import display
# 可选参数'all', 'last', 'last_expr', 'none', 'last_expr_or_assign'
```

In [6]:

```
# 使用mathjax输出公式
#init_printing( use_latex='mathjax' )
```

In [7]:

```
# 为了保证数据结构相同，我们需要在result中添加一列
def data_clean(df):
    int_ind = df.dtypes[df_train.dtypes!=object].index
    obj_ind = df.dtypes[df_train.dtypes==object].index #提取types为object的列名
    ndf = df[int_ind.sort_values()]
    temp2 = ndf.iloc[:,1:6].apply(lambda x:x**2,axis=0)
    temp3 = ndf.iloc[:,1:6].apply(lambda x:x**3,axis=0)
    temp2.columns = list(map(lambda x:x+'**2',list(temp2.columns)))
    temp3.columns = list(map(lambda x:x+'**3',list(temp3.columns)))
    ndf = ndf.join(temp2)
    ndf = ndf.join(temp3)
    for val in obj_ind:
        pp = pd.get_dummies(df[val],prefix=val,prefix_sep='_')
        ndf = ndf.join(pp)
    ndf.iloc[:,1:16] = ndf.iloc[:,1:16].apply(lambda x:(x-x.mean())/x.std(),axis=0)
    return ndf
```

In [8]:

```
df_train = pd.read_csv('Train.csv')
df_train
```

Out[8]:

	年龄	工作情况	教育	教育时间	婚姻状况	职业类型	家庭角色	民族	性别	投资收入	投资损失	工作天数	省份	Y
0	35	个体	初三	5	已婚平民配偶	其他职业	丈夫	民族D	男	0	0	40	省份22	0
1	37	中央部委	高中生	9	已婚平民配偶	保安	丈夫	民族D	男	0	0	40	省份8	0
2	19	个体	初三	5	未婚	手工艺维修	孩子	民族D	男	0	0	20	省份8	0
3	33	个体	大学生	13	已婚平民配偶	专业技术	丈夫	民族D	男	0	0	60	省份8	1
4	22	个体	大学未毕业	10	未婚	手工艺维修	离家	民族D	男	0	0	40	省份8	0

	年龄	工作情况	教育	教育时间	婚姻状况	职业类型	家庭角色	民族	性别	投资收入	投资损失	工作天数	省份	Y
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
38837	34	个体	大学生	13	已婚平民配偶	专业技术	妻子	民族A	女	0	0	35	省份8	0
38838	39	个体	高中生	9	已婚平民配偶	机械操作	丈夫	民族D	男	0	0	40	省份8	1
38839	51	个体	高中生	9	离婚	手工艺维修	离家	民族D	男	0	0	40	省份8	0
38840	25	个体	初三	5	未婚	管理文书	未婚	民族D	女	0	0	40	省份22	0
38841	34	个体	高中生	9	已婚平民配偶	技术支持	丈夫	民族D	男	0	0	40	省份8	1

38842 rows × 14 columns

数据清洗

数据类型检查

数据类型无误，但考虑到教育是**有序分类变量**，民族是**分类变量**，所以之后我们会对其进行额外的处理。

In [9]:
```
df_train.dtypes
```

Out[9]:
```
年龄        int64
工作情况     object
教育       object
教育时间      int64
婚姻状况     object
职业类型     object
家庭角色     object
民族       object
性别       object
投资收入      int64
投资损失      int64
工作天数      int64
省份       object
Y         int64
dtype: object
```

缺失值处理

无任何缺失值，因为题目中说了数据非常可信。

In [10]:
```
df_train.apply(lambda x:sum(x.isnull()),axis=0)
```

Out[10]:
```
年龄      0
工作情况     0
教育      0
教育时间     0
婚姻状况     0
职业类型     0
```

```
家庭角色      0
民族        0
性别        0
投资收入      0
投资损失      0
工作天数      0
省份        0
Y         0
dtype: int64
```

异常值检测

无任何异常值，因为题目中说了数据非常可信。

In [11]:

```
df_train.columns
```

Out[11]:

```
Index(['年龄', '工作情况', '教育', '教育时间', '婚姻状况', '职业类型', '家庭角色', '民族', '性别',
'投资收入',
       '投资损失', '工作天数', '省份', 'Y'],
      dtype='object')
```

连续变量的处理

关于连续变量的处理方式的讨论，始终集中在：

- 是否应当消除量纲？
- 如果选择消除量纲，那么是归一化还是标准化？这个问题将在 '构建新的数据集'探讨

In [ ]:

In [ ]:

分类变量的处理

- 对object类型的变量全部进行分类处理，即实现One-Hot编码
- 这样处理可能会产生问题，因为我们没有区分**有序变量**和**分类变量**

In [12]:

```
# 对object类型的变量全部进行分类处理，即实现One-Hot编码
int_ind = df_train.dtypes[df_train.dtypes!=object].index
obj_ind = df_train.dtypes[df_train.dtypes==object].index #提取types为object的列名
for val in obj_ind:
    globals()['df_train_{}'.format(val)] = pd.get_dummies(df_train[val],prefix=val,prefix_
```

In [13]:

```
df_train_婚姻状况
```

Out[13]:

	婚姻状况_丧偶	婚姻状况_分居	婚姻状况_已婚军属	婚姻状况_已婚平民配偶	婚姻状况_已婚配偶异地	婚姻状况_未婚	婚姻状况_离婚
0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
2	0	0	0	0	0	1	0

	婚姻状况_丧偶	婚姻状况_分居	婚姻状况_已婚军属	婚姻状况_已婚平民配偶	婚姻状况_已婚配偶异地	婚姻状况_未婚	婚姻状况_离婚
3	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...
38837	0	0	0	1	0	0	0
38838	0	0	0	1	0	0	0
38839	0	0	0	0	0	0	1
38840	0	0	0	0	0	1	0
38841	0	0	0	1	0	0	0

38842 rows × 7 columns

In [ ]:

探索性数据分析—描述统计

In [14]:

```
df_train.describe()
```

Out[14]:

	年龄	教育时间	投资收入	投资损失	工作天数	Y
count	38842.000000	38842.000000	38842.000000	38842.000000	38842.000000	38842.000000
mean	38.676613	10.089851	1096.261907	87.989470	40.388240	0.239921
std	13.732165	2.577300	7547.487571	403.268938	12.419557	0.427040
min	17.000000	1.000000	0.000000	0.000000	1.000000	0.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000	0.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000	0.000000
75%	48.000000	13.000000	0.000000	0.000000	45.000000	0.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000	1.000000

从5数表中可以看到数据离散程度非常高，这意味着我们的预测模型精度肯定会比较好。

关于分类变量的描述

In [15]:

```
df_train.loc[:,'婚姻状况'].value_counts()
```

Out[15]:

```
已婚平民配偶      17824
未婚            12809
离婚             5242
丧偶             1216
分居             1208
已婚配偶异地        512
```

```
已婚军属          31
Name: 婚姻状况, dtype: int64
```

In [16]:
```
df_train.loc[:,'工作情况',].value_counts()
```

Out[16]:
```
个体            26909
非有限责任公司       3095
地方政府          2512
未知            2244
中央部委          1568
有限责任公司        1357
省政府           1131
无收入             16
从未工作            10
Name: 工作情况, dtype: int64
```

In [17]:
```
i = 0
df_train.groupby(obj_ind[i])['Y'].mean()
```

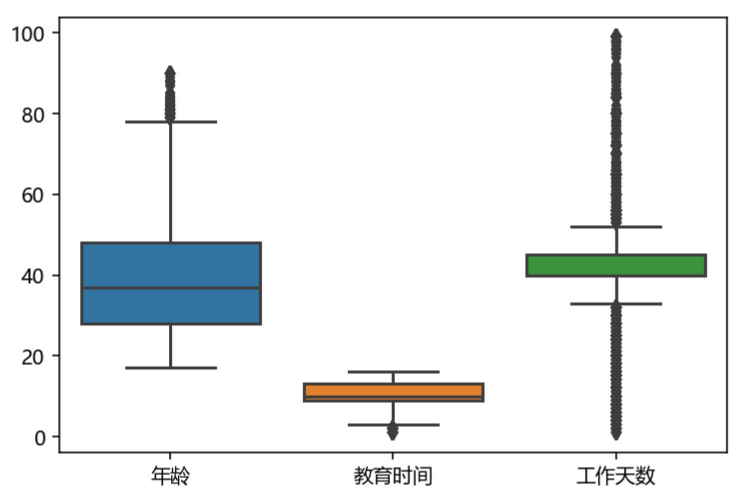
Out[17]:
```
工作情况
个体          0.218514
中央部委        0.260204
从未工作        0.000000
地方政府        0.298169
无收入         0.125000
有限责任公司      0.551216
未知          0.094029
省政府         0.396994
非有限责任公司     0.281745
Name: Y, dtype: float64
```

关于数值变量的描述

本节绘制了箱线图用于观察连续性数据的分布

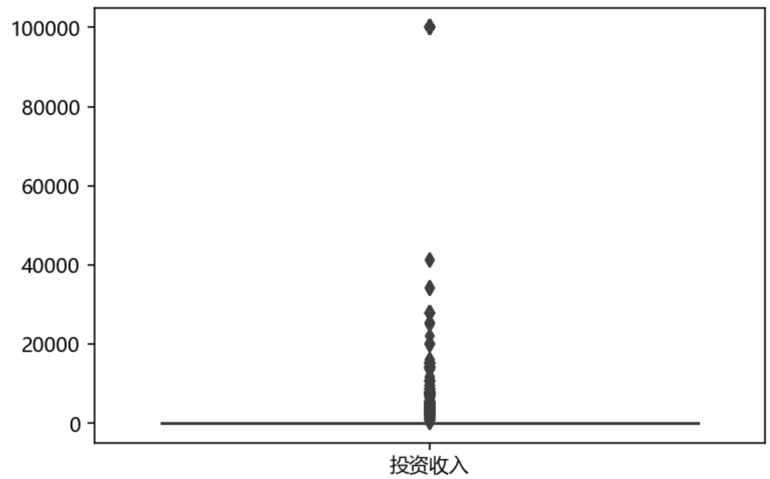
In [18]:
```
plt.rcParams['font.sans-serif'] =['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
sns.boxplot(data=df_train[[int_ind[0],int_ind[1],int_ind[4]]])
```

Out[18]:
```
&lt;AxesSubplot:&gt;
```



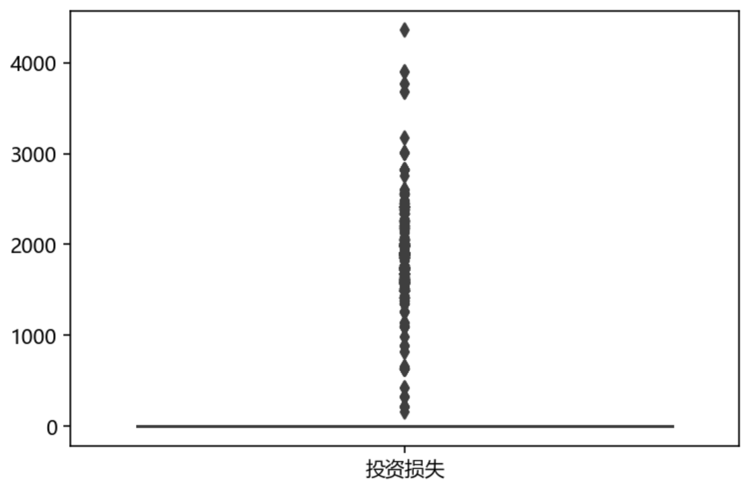
In [19]:
```
plt.rcParams['font.sans-serif'] =['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
sns.boxplot(data=df_train[[int_ind[2]]])
```

Out[19]: <AxesSubplot:>

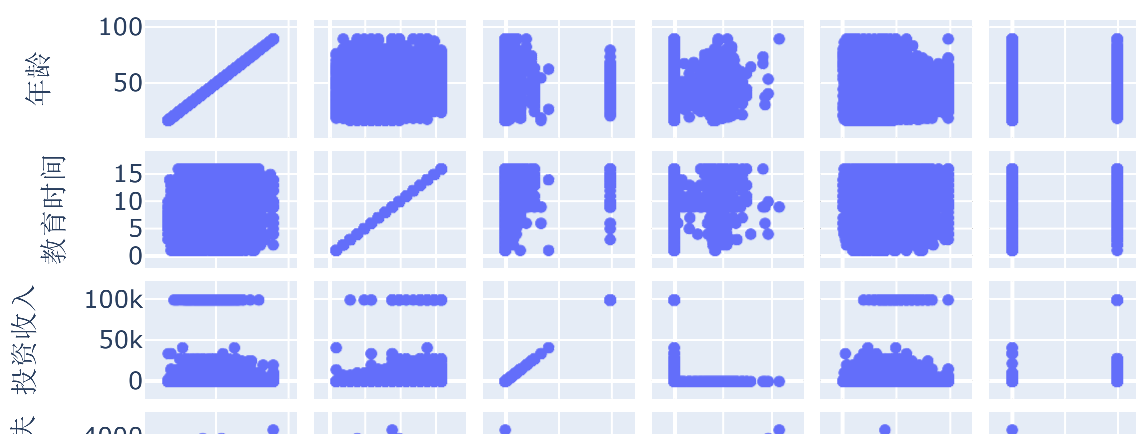


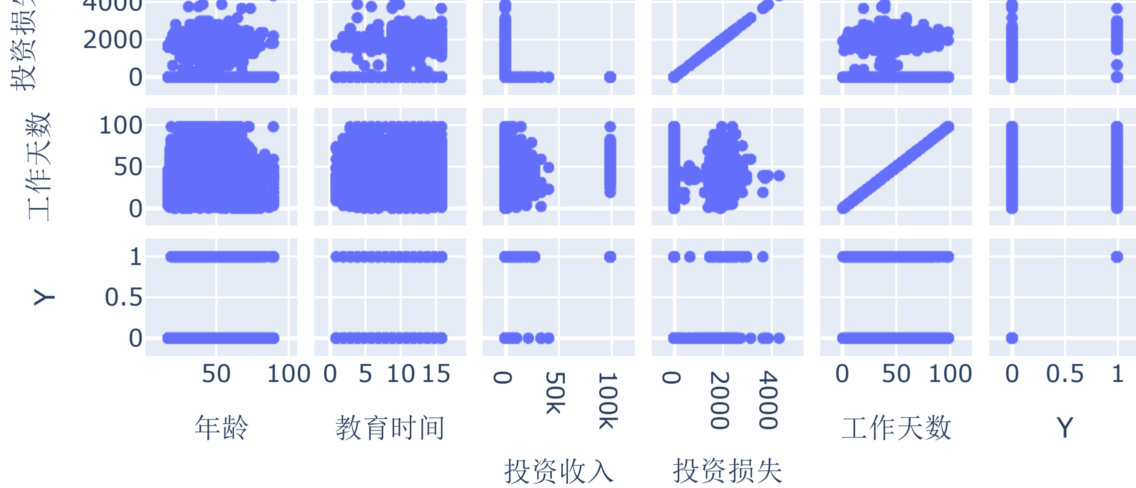
In [20]:
```
plt.rcParams['font.sans-serif'] =['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
sns.boxplot(data=df_train[[int_ind[3]]])
```

Out[20]: <AxesSubplot:>



In [21]:
```
import plotly.express as px
df = df_train
fig = px.scatter_matrix(df, dimensions=list(int_ind))

fig.show()
```






```
In [ ]:
```

高收入 v.s. 分类变量

In [22]:

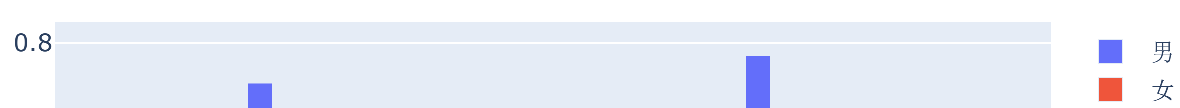
```python
i = 0
df_train.groupby(obj_ind[i])['Y'].mean()
```

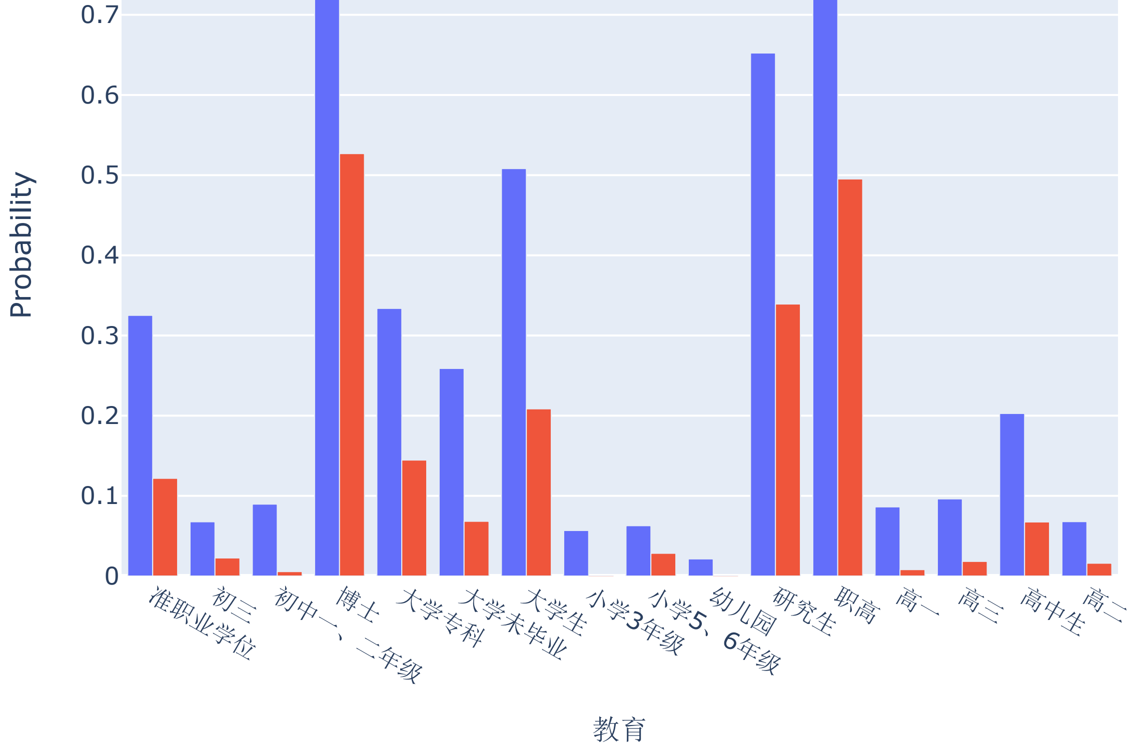
Out[22]:

```
工作情况
个体            0.218514
中央部委          0.260204
从未工作          0.000000
地方政府          0.298169
无收入           0.125000
有限责任公司        0.551216
未知            0.094029
省政府           0.396994
非有限责任公司       0.281745
Name: Y, dtype: float64
```

In [23]:

```python
import plotly.graph_objects as go
temp_df = pd.DataFrame(df_train.groupby([obj_ind[1],obj_ind[6]],as_index=False)['Y'].mean
classes=list(df_train.groupby(obj_ind[1])['Y'].mean().index)
tempy1 = temp_df.loc[temp_df['性别']=='男','Y']
tempy2 = temp_df.loc[temp_df['性别']=='女','Y']
fig = go.Figure(data=[
    go.Bar(name='男', x=classes, y=tempy1),
    go.Bar(name='女', x=classes, y=tempy2)])

fig.update_layout(
    title="",
    xaxis_title="教育",
    yaxis_title="Probability"
)
fig.show();
```

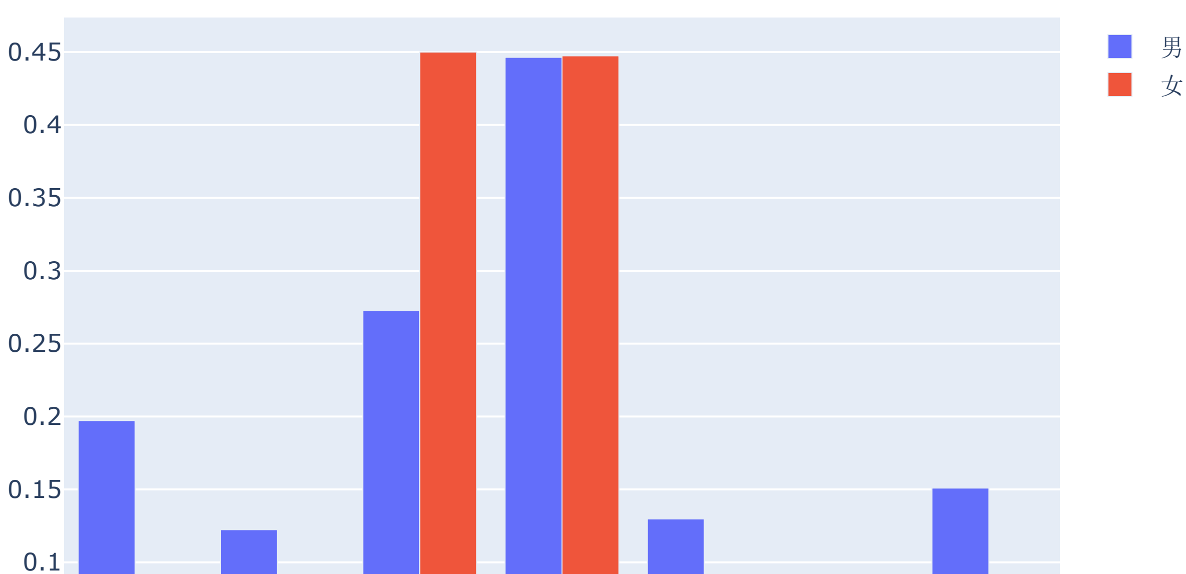


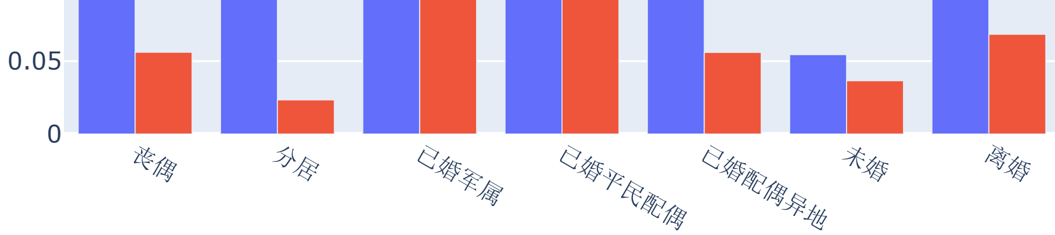


In [ ]:

In [24]:

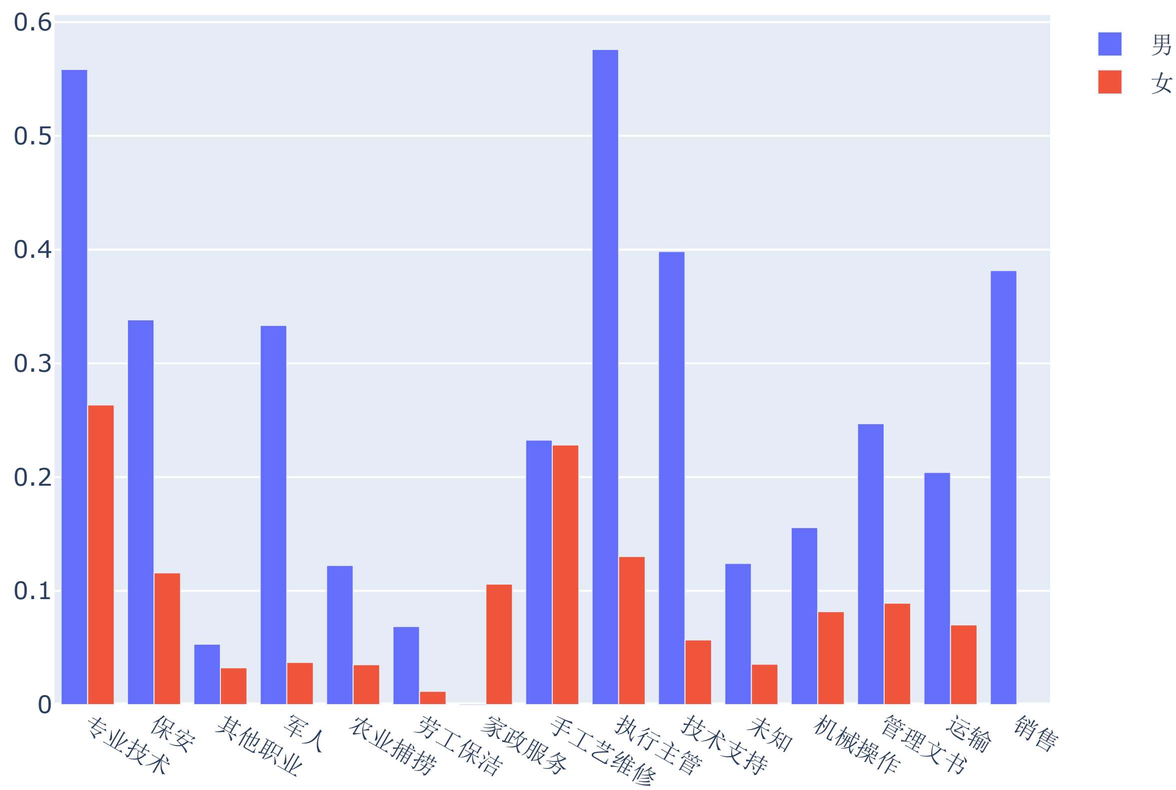
```
import plotly.graph_objects as go
temp_df = pd.DataFrame(df_train.groupby([obj_ind[2],obj_ind[6]],as_index=False)['Y'].mean
classes=list(df_train.groupby(obj_ind[2])['Y'].mean().index)
tempy1 = temp_df.loc[temp_df['性别']=='男','Y']
tempy2 = temp_df.loc[temp_df['性别']=='女','Y']
fig = go.Figure(data=[
    go.Bar(name='男', x=classes, y=tempy1),
    go.Bar(name='女', x=classes, y=tempy2)
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show();
```





In [25]:

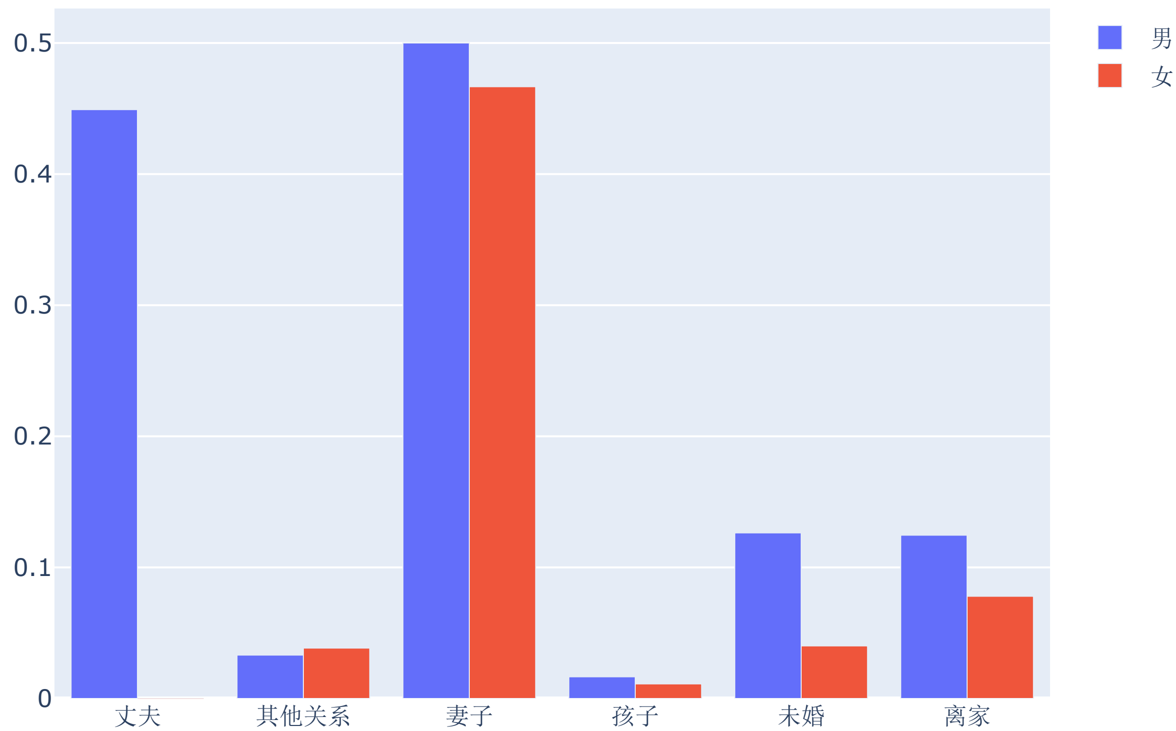
```
import plotly.graph_objects as go
temp_df = pd.DataFrame(df_train.groupby([obj_ind[3],obj_ind[6]],as_index=False)['Y'].mean
classes=list(df_train.groupby(obj_ind[3])['Y'].mean().index)
tempy1 = temp_df.loc[temp_df['性别']=='男','Y']
tempy2 = temp_df.loc[temp_df['性别']=='女','Y']
fig = go.Figure(data=[
    go.Bar(name='男', x=classes, y=tempy1),
    go.Bar(name='女', x=classes, y=tempy2)
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show();
```



In [26]:

```
import plotly.graph_objects as go
temp_df = pd.DataFrame(df_train.groupby([obj_ind[4],obj_ind[6]],as_index=False)['Y'].mean
classes=list(df_train.groupby(obj_ind[4])['Y'].mean().index)
tempy1 = temp_df.loc[temp_df['性别']=='男','Y']
tempy2 = temp_df.loc[temp_df['性别']=='女','Y']
fig = go.Figure(data=[
```

```python
    go.Bar(name='男', x=classes, y=tempy1),
    go.Bar(name='女', x=classes, y=tempy2)
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show();
```



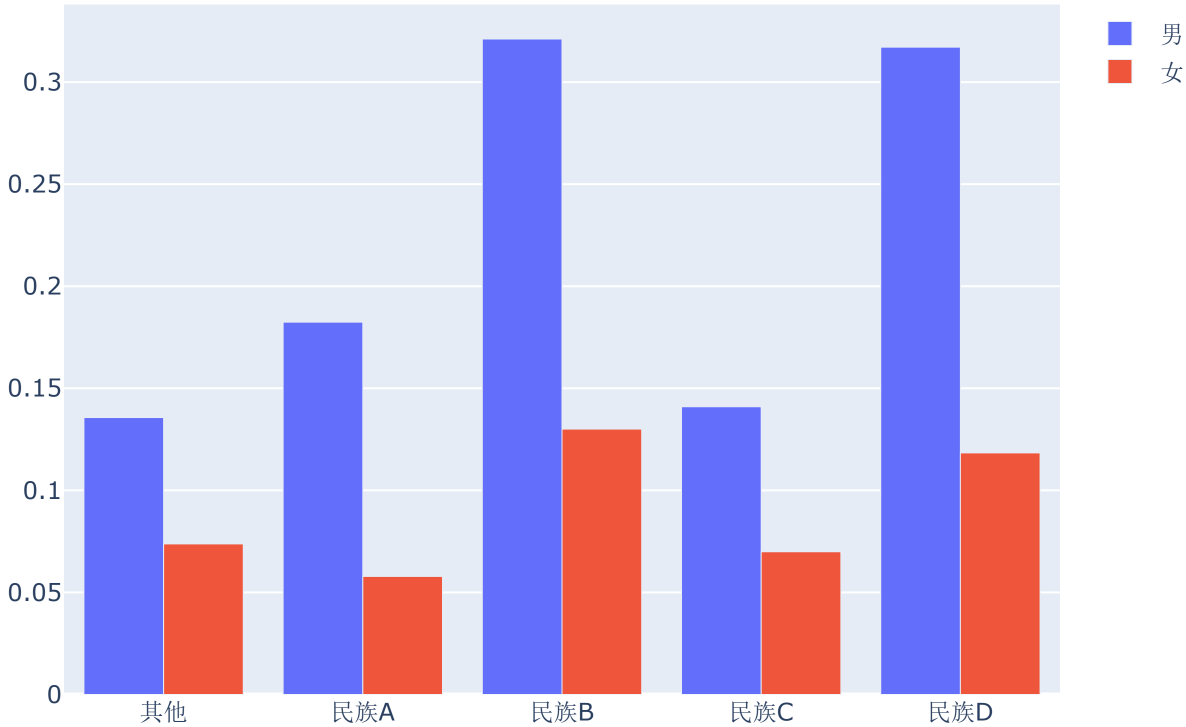
In [27]:

```python
list(df_train.groupby(obj_ind[5])['Y'].mean().index)
```

Out[27]: ['其他', '民族A', '民族B', '民族C', '民族D']

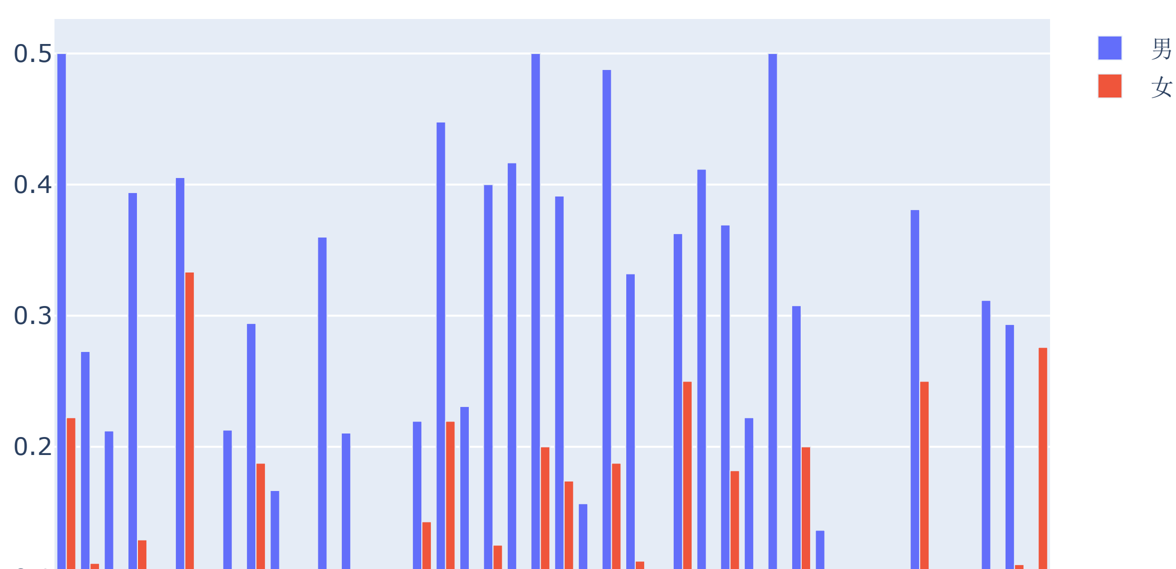
In [28]:

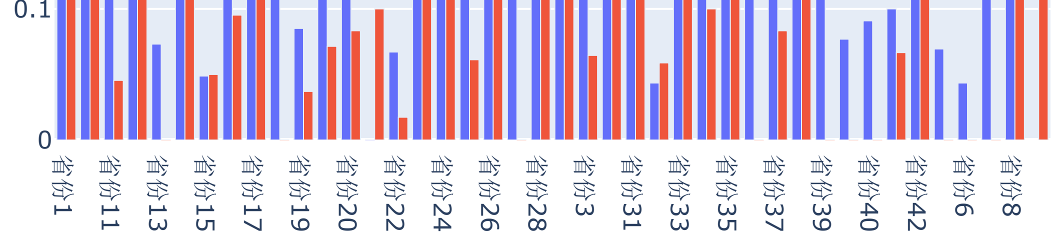
```python
import plotly.graph_objects as go
temp_df = pd.DataFrame(df_train.groupby([obj_ind[5],obj_ind[6]],as_index=False)['Y'].mean
classes=list(df_train.groupby(obj_ind[5])['Y'].mean().index)
tempy1 = temp_df.loc[temp_df['性别']=='男','Y']
tempy2 = temp_df.loc[temp_df['性别']=='女','Y']
fig = go.Figure(data=[
    go.Bar(name='男', x=classes, y=tempy1),
    go.Bar(name='女', x=classes, y=tempy2)
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show();
```



In [29]:

```
import plotly.graph_objects as go
temp_df = pd.DataFrame(df_train.groupby([obj_ind[7],obj_ind[6]],as_index=False)['Y'].mean
classes=list(df_train.groupby(obj_ind[7])['Y'].mean().index)
tempy1 = temp_df.loc[temp_df['性别']=='男','Y']
tempy2 = temp_df.loc[temp_df['性别']=='女','Y']
fig = go.Figure(data=[
    go.Bar(name='男', x=classes, y=tempy1),
    go.Bar(name='女', x=classes, y=tempy2)
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show();
```

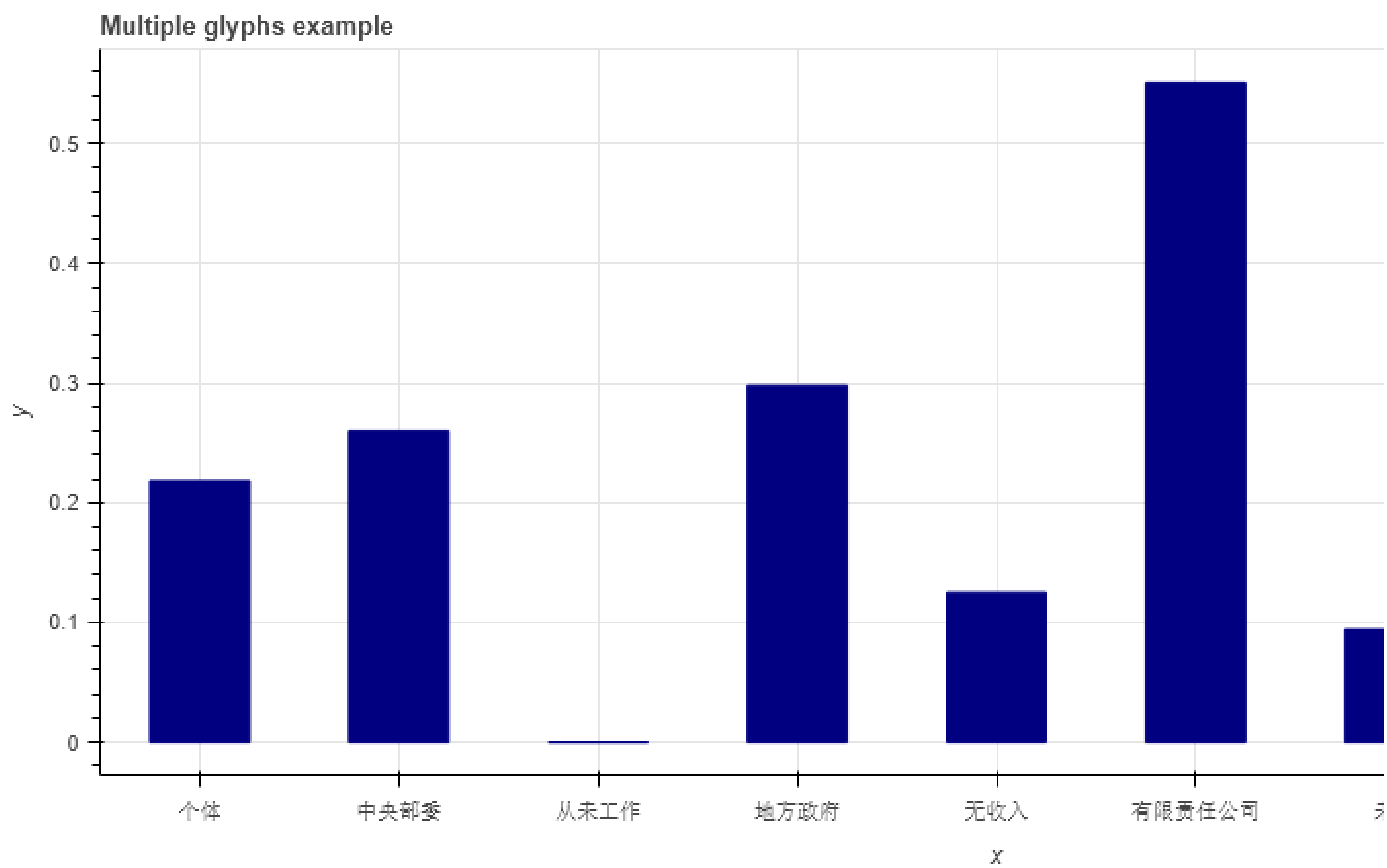




In [30]:

```
p = figure(x_range=list(df_train.groupby(obj_ind[i])['Y'].mean().index),title="Multiple gl
           x_axis_label="x", y_axis_label="y",plot_width=1000,plot_height=450,
          )
p.vbar(x=list(df_train.groupby(obj_ind[i])['Y'].mean().index), top=df_train.groupby(obj_ir
      )
show(p)
```

Out[30]: **GlyphRenderer**(id = '1039', ...)



In [31]:

```
list(df_train.groupby(obj_ind[i])['Y'].mean().index), df_train.groupby(obj_ind[i])['Y'].me
```

Out[31]:

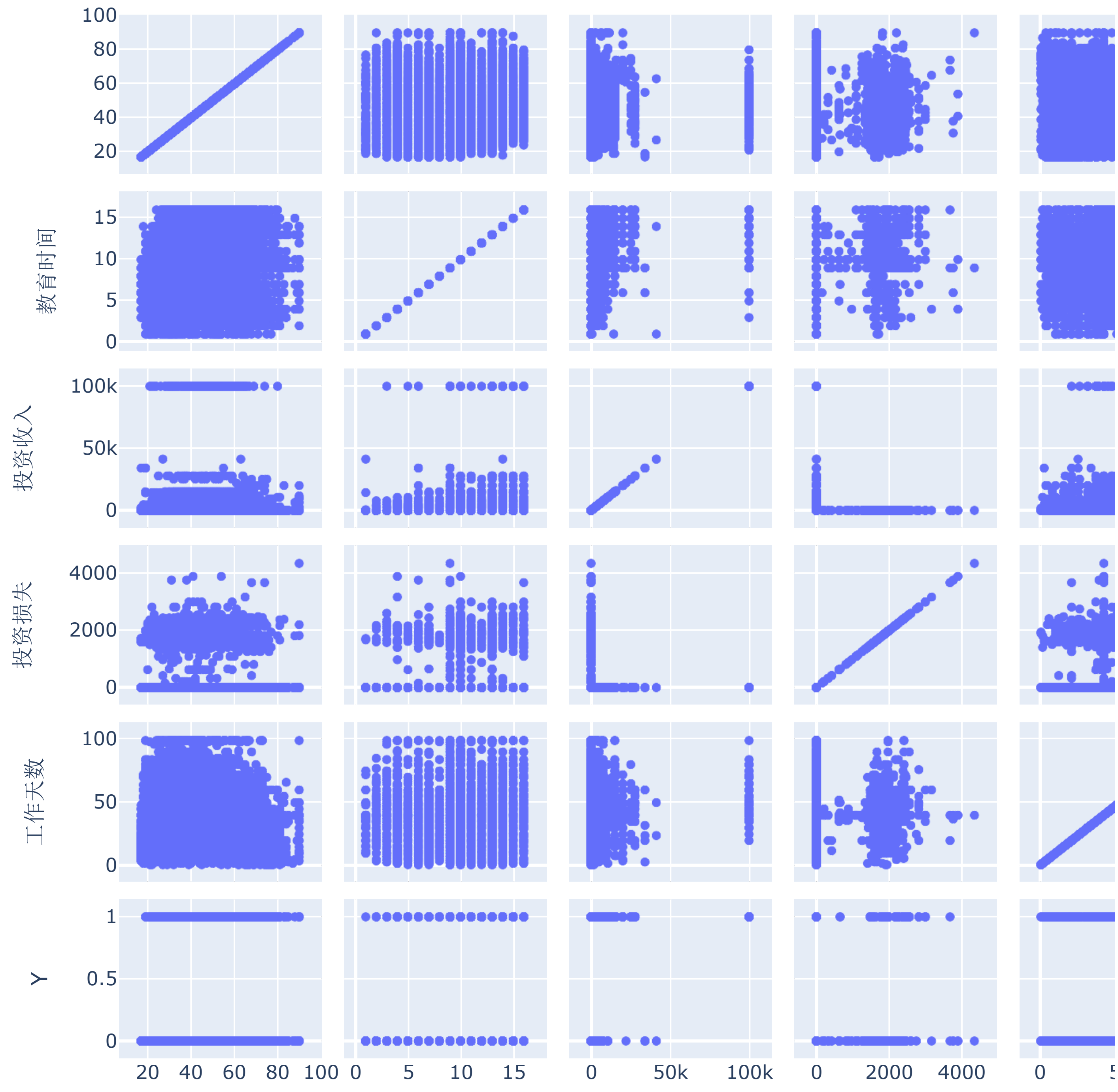
```
(['个体', '中央部委', '从未工作', '地方政府', '无收入', '有限责任公司', '未知', '省政府', '非有限责任
公司'],
 工作情况
 个体          0.218514
 中央部委        0.260204
 从未工作        0.000000
 地方政府        0.298169
 无收入         0.125000
 有限责任公司      0.551216
 未知          0.094029
 省政府         0.396994
```

```
非有限责任公司      0.281745
Name: Y, dtype: float64)
```

高收入 v.s. 数值变量

In [32]:

```
df = df_train
fig = px.scatter_matrix(df, dimensions=list(int_ind))
fig.update_layout(
    title="",
    xaxis_title="",
    yaxis_title="",
    autosize = False,
    width = 1000,
    height = 800
)
fig.show();
```



教育时间 投资收入 投资损失 工作

In [ ]:

构建新的数据集

直接按照下述方法构建数据集会产生问题，因为矩阵不是列满秩的(某些变量可由其余变量线性表出，这提供了重复信息)，不过现在的代码应该很智能，能自动处理这个问题

数据介绍

- ndf_train: One-Hot 编码后的数据
  - sndf_train: ndf_train 标准化后的数据
  - smdf_train: ndf_train 0-1化后的数据
- nonlinear_ndf_train: One-Hot 编码后,且引入非线性因素的数据
  - nonlinear_sndf_train: 标准化后的数据
  - nonlinear_smdf_train: 0-1化后的数据

强烈推荐使用nonlinear_sndf_train，因为我试过了

In [33]:

```
# 我们在
ndf_train = df_train[int_ind.sort_values()]
for val in obj_ind:
    ndf_train = ndf_train.join(globals()['df_train_{}'.format(val)])
```

In [34]:

```
## 如果将连续变量进行标准化处理处理
sndf_train = ndf_train.copy()
sndf_train.iloc[:,1:6] = sndf_train.iloc[:,1:6].apply(lambda x:(x-x.mean())/x.std(),axis=(
```

In [35]:

```
smndf_train = ndf_train.copy()
smndf_train.iloc[:,1:6] = smndf_train.iloc[:,1:6].apply(lambda x:(x-x.min())/(x.max()-x.mi
```

In [36]:

```
# 引入非线性的成分
nonlinear_ndf_train = df_train[int_ind.sort_values()]
temp2 = nonlinear_ndf_train.iloc[:,1:6].apply(lambda x:x**2,axis=0)
temp3 = nonlinear_ndf_train.iloc[:,1:6].apply(lambda x:x**3,axis=0)
temp2.columns = list(map(lambda x:x+'**2',list(temp2.columns)))
temp3.columns = list(map(lambda x:x+'**3',list(temp3.columns)))
nonlinear_ndf_train = nonlinear_ndf_train.join(temp2)
nonlinear_ndf_train = nonlinear_ndf_train.join(temp3)
for val in obj_ind:
    nonlinear_ndf_train = nonlinear_ndf_train.join(globals()['df_train_{}'.format(val)])
nonlinear_sndf_train = nonlinear_ndf_train.copy()
nonlinear_smndf_train = nonlinear_ndf_train.copy()
nonlinear_sndf_train.iloc[:,1:16] = nonlinear_sndf_train.iloc[:,1:16].apply(lambda x:(x-x.
nonlinear_smndf_train.iloc[:,1:16] = nonlinear_smndf_train.iloc[:,1:16].apply(lambda x:(x-
```

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [37]:

```
ndf_train['Y'].mean()
```

Out[37]:

```
0.23992070439215282
```

模型准备

函数定义

In [38]:

```
def learning_curve_plot(model,Xdata,Ydata):
    train_size,train_loss,test_loss = learning_curve(Logit.model_fitted,Xdata,Ydata,
                                                     scoring='neg_mean_squared_error', tra
                                                    n_jobs=-1)
    train_loss_mean = -np.mean(train_loss,axis=1)
    test_loss_mean = -np.mean(test_loss,axis=1)
    plt.figure()
    plt.plot(train_size,train_loss_mean,'r-+',linewidth=2, label="train")
    plt.plot(train_size,test_loss_mean,'b-',linewidth=3, label="validation")
    plt.xlabel('Training size')
    plt.ylabel('Mean squared error')
    plt.legend(['training','validation'])
    plt.show();
```

ROC曲线绘制

In [39]:

```
def roc_curve_plot(fpr,tpr,ruc):
    plt.plot(fpr, tpr, lw=2, alpha=.6)
    plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
    plt.xlim([0, 1])
    plt.ylim([0, 1.05])
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC curve")
    plt.legend(["(AUC {:.4f})".format(ruc)], fontsize=8, loc=2)
    plt.show();
```

并不是很推荐的调用方式

In [40]:

```
def Classifier(Xdata,Ydata,class_model,size_pct=0.2,model="Temp",rn=0,i=1,*args,**kwargs):
    globals()['Model_{}_{}'.format(model,i)] = class_model(*args,**kwargs)
    globals()['X_train_{}_{}'.format(model,i)],globals()['X_test_{}_{}'.format(model,i)],g
    globals()['X_trainabs_{}_{}'.format(model,i)],globals()['X_valid_{}_{}'.format(model,i
    globals()['Model_{}_{}'.format(model,i)].fit(globals()['X_train_{}_{}'.format(model,i)
    globals()['Y_train_predict_{}_{}'.format(model,i)] = globals()['Model_{}_{}'.format(mo
    # 在test集合上预测-----------------------------------------------------------------------
    globals()['Y_test_predict_{}_{}'.format(model,i)] = globals()['Model_{}_{}'.format(mod
    # 得到AUC -----------------------------------------------------------------------------
    globals()['Y_test_prob_{}_{}'.format(model,i)] = globals()['Model_{}_{}'.format(model,
    globals()['fpr_{}_{}'.format(model,i)], globals()['tpr_{}_{}'.format(model,i)], global
    globals()['auc_{}_{}'.format(model,i)] = auc(globals()['fpr_{}_{}'.format(model,i)], g
    return globals()['auc_{}_{}'.format(model,i)]
```

In [41]:

```
def Classifier_CV(Xdata,Ydata,class_model,size_pct=0.2,model="Temp",rn=0,i=1,*args,**kwarg
    globals()['Model_{}_{}'.format(model,i)] = class_model(*args,**kwargs)
    globals()['X_train_{}_{}'.format(model,i)],globals()['X_test_{}_{}'.format(model,i)],g
    globals()['X_trainabs_{}_{}'.format(model,i)],globals()['X_valid_{}_{}'.format(model,i
    globals()['Model_{}_{}'.format(model,i)].fit(globals()['X_train_{}_{}'.format(model,i)
    #在train上预测
    globals()['Y_train_predict_CV_{}_{}'.format(model,i)] = cross_val_predict(globals()['M
    # 在test集合上预测-----------------------------------------------------------------------
    globals()['Y_test_predict_CV_{}_{}'.format(model,i)] = cross_val_predict(globals()['Mo
    # 得到AUC --------------------------------------------------------------------------
    globals()['Y_test_prob_CV_{}_{}'.format(model,i)] = cross_val_predict(globals()['Model
    globals()['fpr_CV_{}_{}'.format(model,i)], globals()['tpr_CV_{}_{}'.format(model,i)],
    globals()['auc_CV_{}_{}'.format(model,i)] = auc(globals()['fpr_CV_{}_{}'.format(model,
    return globals()['auc_CV_{}_{}'.format(model,i)]
```

函数测试

In [42]:

```
i = 1 #模型编号
model = 'Logit' # 模型名称
size_pct = 0.2 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
```

In [43]:

```
Classifier(Xdata,Ydata,GradientBoostingClassifier,model='GTB',size_pct=0.3,rn=0,i=1)
```

Out[43]:

```
0.9179638236488398
```

In [44]:

```
Classifier(Xdata,Ydata,XGBClassifier,model='XGB',size_pct=0.3,rn=0,i=1)
```

```
[19:41:25] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[44]:

```
0.9269691772208188
```

定义分类器Class(强烈推荐)

参数字典

In [45]:

```
## 参数字典
Classifiers_dict = {'LogisticRegression':pd.DataFrame([['penalty','l1,l2','处罚项，可选'],['
 'XGBClassifier':pd.DataFrame([['eval_metric','''['logloss','auc','error']''','避免报错'],[
 'Log':pd.DataFrame([['n',10],['m',20]])}
```

分类器Classifiers

In [144…

```
class Classifiers():
    details = ['LogisticRegression','XGBClassifier','GradientBoostingClassifier','AdaBoost
               'RandomForestClassifier','KNeighborsClassifier','DecisionTreeClassifier','N

    def __init__(self,class_model):
        self.model_class = class_model()
        self.model_name = str(self.model_class).split('(')[0]
        #self.model_params = Classifiers_dict[self.model_name]
```

```
    # 构建常用参数字典
    def param_dict(self):
        print('1')
    def setmodelparam(self,class_model,*args,**kwargs):
        self.model_paramed = class_model(*args,**kwargs)

    def setdata(self,X,Y,test_size=0.3,random_state=0):
        self.Xdata = X
        self.Ydata = Y
        self.X_train, self.X_test, self.Y_train, self.Y_test =  train_test_split(X,Y, test
        self.X_train_abs, self.X_valid, self.Y_train_abs, self.Y_valid =  train_test_split

    def fit_model(self,X_train=[None],Y_train=[None],cv=0,*kargs,**kwargs):
        if list(Y_train)[0] == None:
            X_train, Y_train = self.X_train,self.Y_train
        self.model_fitted = self.model_paramed.fit(X_train,Y_train,*kargs,**kwargs)
        if cv > 0:
            self.Y_train_predict_CV = cross_val_predict(self.model_fitted,self.X_train,sel
            self.Y_test_predict_CV = cross_val_predict(self.model_fitted,self.X_test,self.
            self.Y_test_prob_CV = cross_val_predict(self.model_fitted,self.X_test,self.Y_t
            self.fpr_CV,self.tpr_CV,self.thresholds_CV = roc_curve(self.Y_test,self.Y_test
            self.auc_CV = auc(self.fpr_CV,self.tpr_CV)
        else:
            self.Y_train_predict = self.model_fitted.predict(self.X_train)
            self.Y_test_predict = self.model_fitted.predict(self.X_test)
            self.Y_test_prob = self.model_fitted.predict_proba(self.X_test)
            self.fpr,self.tpr,self.thresholds = roc_curve(self.Y_test,self.Y_test_prob[:,1
            self.auc = auc(self.fpr,self.tpr)

    def auc_plot(self,cv=0):
        if cv > 0:
            return roc_curve_plot(self.fpr_CV,self.tpr_CV,self.auc_CV)
        else:
            return roc_curve_plot(self.fpr,self.tpr,self.auc)
    def confusion_matrix_train(self,cv=0):
        if cv > 0:
            #print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(self.Y_t
            print("Classification report (training):\n {0}".format(classification_report(s
        else:
            #print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(self.Y_t
            print("Classification report (training):\n {0}".format(classification_report(s

    def confusion_matrix_test(self,cv=0):
        if cv > 0:
            #print("Confusion matrix (testing):\n {0}\n".format(confusion_matrix(self.Y_te
            print("Classification report (testing):\n {0}".format(classification_report(se
        else:
            #print("Confusion matrix (testing):\n {0}\n".format(confusion_matrix(self.Y_te
            print("Classification report (testing):\n {0}".format(classification_report(se

    def details_print(self):
        for i in self.details:
            print(i+'\n'+'----------------------------------')
    def learning_curve_plot(self,nmax=20):
        self.diagnose_train_size,self.diagnose_train_loss,self.diagnose_test_loss = learni
                                                 scoring='neg_mean_squared_error',
        self.diagnose_train_loss_mean = -np.mean(self.diagnose_train_loss,axis=1)
        self.diagnose_test_loss_mean = -np.mean(self.diagnose_test_loss,axis=1)
        plt.figure()
        plt.plot(self.diagnose_train_size,self.diagnose_train_loss_mean,'r-+',linewidth=2,
        plt.plot(self.diagnose_train_size,self.diagnose_test_loss_mean,'b-',linewidth=3, l
        plt.xlabel('Training_size')
        plt.ylabel('Mean_squared_error')
        plt.legend(['training','validation'])
        plt.show();
```

In [47]:
```
# class Classifiers_dict():
#     details = ['LogisticRegression','XGBClassifier','GradientBoostingClassifier','AdaBoo
#                'RandomForestClassifier','KNeighborsClassifier','DecisionTreeClassifier',
#     def __init__(self,class_model):

#     def model_parameters_dict(self):
#         self.params_LogisticRegression = pd.DataFrame([['penalty','l1,l2','处罚项, 可选'],
```

一些简单的测试

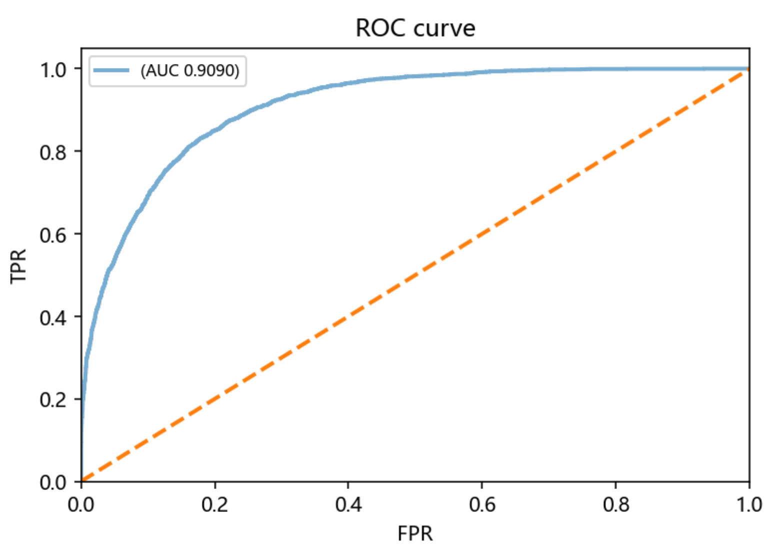
In [48]:
```
i = 1 #模型编号
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
```

In [49]:
```
Logit = Classifiers(LogisticRegression)
Logit.setmodelparam(LogisticRegression,penalty='l2')
Logit.setdata(Xdata,Ydata)
Logit.fit_model()
```

In [50]:
```
Logit.model_class
```

Out[50]:
```
LogisticRegression()
```

In [51]:
```
Logit.auc_plot()
```



In [52]:
```
XGB1 = Classifiers(XGBClassifier)
XGB1.setmodelparam(XGBClassifier,eval_metric=['logloss','auc','error'])
XGB1.setdata(Xdata,Ydata)
```

In [53]:
```
XGB1.fit_model()
#XGB1.fit_model(X_train=XGB1.X_train_abs,Y_train=XGB1.Y_train_abs,eval_set=[(XGB1.X_valid,
```

In [54]:
```
XGB1.model_params
```

Out[54]:

	Name	Options	Details
参数	eval_metric	['logloss','auc','error']	避免报错
	eta	0.3	学习率

In [ ]:

```

```

In [ ]:

```
## 调参例子
auc_lst = []
for n in range(30):
    Logit.setmodelparam(LogisticRegression,n)
    Logit.setdata(Xdata,Ydata)
    Logit.fit_model()
    auc_lst.append(Logit.auc())
```

模型选择

广义线性模型

关于广义线性模型的介绍，可参考：

- R语言教程第37节
- Beyond Multiple Linear Regression

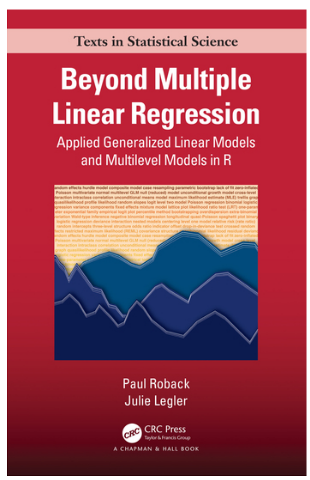


Figure 超越多元线性回归

个人认为在概率模型中，选择L_2惩罚比L_1惩罚更加合理，因为L_2惩罚算的是条件期望，而L_1惩罚算的是条件中位数

Binomial Regression (link = Canonical(Logit))

模型介绍

> 广义线性模型--二项回归，如果我们选择Logit函数作为连接Y与线性预测子的函数(链接函数)，此时的二项回归也称作Logistic回归：
> 但似乎没有在python里找到链接函数这个选项...

这里我选择把关于惩罚项的设定列出来，便于查阅:

''' penalty : {'l1', 'l2', 'elasticnet', 'none'}, default='l2' Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied. '''

In [101…

```
['年龄', '教育时间', '投资收入', '投资损失', '工作天数']
```

Out[101…

```
['年龄', '教育时间', '投资收入', '投资损失', '工作天数', 'Y']
```

In [55]:

```
i = 0 #模型编号
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = df_train[['年龄', '教育时间', '投资收入', '投资损失', '工作天数']]
Ydata = df_train['Y']
setname = 'Logit'
model_name = LogisticRegression
globals()['{}_{}'.format(setname,i)] = Classifiers(model_name)
globals()['{}_{}'.format(setname,i)].setmodelparam(model_name)
globals()['{}_{}'.format(setname,i)].setdata(Xdata,Ydata)
globals()['{}_{}'.format(setname,i)].fit_model()
```

In [ ]:

In [ ]:

In [56]:

```
i = 1 #模型编号
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = ndf_train.iloc[:,1:]
Ydata = ndf_train.iloc[:,0]
setname = 'Logit'
model_name = LogisticRegression
globals()['{}_{}'.format(setname,i)] = Classifiers(model_name)
globals()['{}_{}'.format(setname,i)].setmodelparam(model_name)
globals()['{}_{}'.format(setname,i)].setdata(Xdata,Ydata)
globals()['{}_{}'.format(setname,i)].fit_model()
```

In [57]:

```
i = 2 #模型编号
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = ndf_train.iloc[:,1:]
Ydata = ndf_train.iloc[:,0]
setname = 'Logit'
model_name = LogisticRegression
globals()['{}_{}'.format(setname,i)] = Classifiers(model_name)
globals()['{}_{}'.format(setname,i)].setmodelparam(model_name,penalty='l2',class_weight={(
globals()['{}_{}'.format(setname,i)].setdata(Xdata,Ydata)
globals()['{}_{}'.format(setname,i)].fit_model()
```

参数调整

In [58]:

```
i = 1 #模型编号
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
```

In [59]:

```
Logit = Classifiers(LogisticRegression)
Logit.setmodelparam(LogisticRegression,penalty='l2',class_weight={0:0.24,1:0.76})
```

```
Logit.setdata(Xdata,Ydata)
Logit.fit_model()
```

关于混淆矩阵，我们需要更多关注precision(1), 因为负样本占比较大，在某些极端情形，将预测值全部预测为负，此时也能做到precision(0)=1

$$precision(i) = Pr(X = i|\hat{X} = i)$$

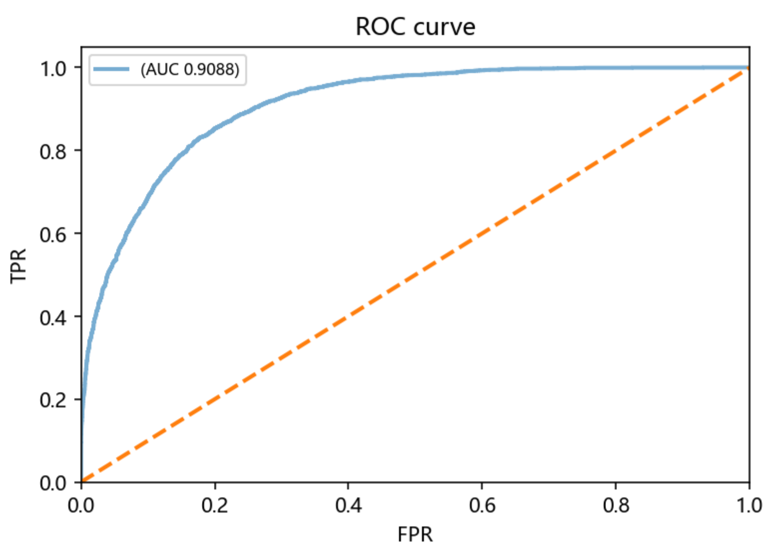
In [60]:
```
Logit.confusion_matrix_test()
```

```
Classification report (testing):
              precision    recall  f1-score   support

           0       0.94      0.80      0.87      8843
           1       0.58      0.85      0.69      2810

    accuracy                           0.81     11653
   macro avg       0.76      0.83      0.78     11653
weighted avg       0.86      0.81      0.82     11653
```

In [61]:
```
Logit.confusion_matrix_train()
```

```
Classification report (training):
              precision    recall  f1-score   support

           0       0.95      0.81      0.87     20680
           1       0.58      0.85      0.69      6509

    accuracy                           0.82     27189
   macro avg       0.76      0.83      0.78     27189
weighted avg       0.86      0.82      0.83     27189
```

In [62]:
```
Logit.auc_plot()
```



In [ ]:

In [ ]:

In [196…

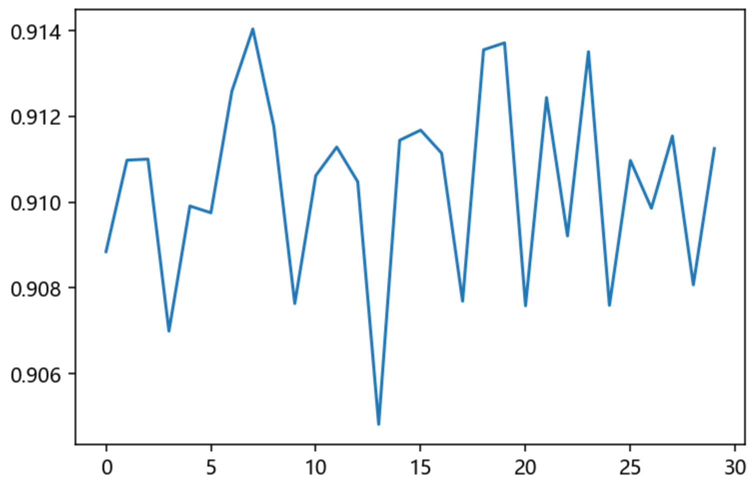
```python
# 验证样本的稳定型
auc_lst = []
for j in list(range(30)):
    rn = j
    Logit = Classifiers(LogisticRegression)
    Logit.setmodelparam(LogisticRegression,penalty='l2',class_weight={0:0.24,1:0.76})
    Logit.setdata(Xdata,Ydata)
    Logit.fit_model()
    auc_lst.append(Logit.auc)
```

In [198…

```python
plt.plot(pd.Series(auc_lst))
```

Out[198…

```
[<matplotlib.lines.Line2D at 0x126b609bca0>]
```



这说明结果相对稳定，平均auc在0.91

In [ ]:

In [ ]:

In [185…

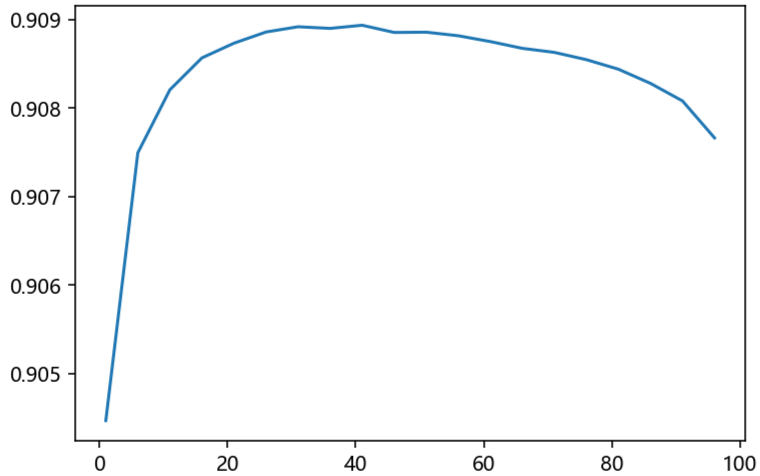
```python
# 验证参数的合理性
auc_lst = []
for x in list(range(1,100,5)):
    Logit = Classifiers(LogisticRegression)
    Logit.setmodelparam(LogisticRegression,penalty='l2',class_weight={0:x/100,1:1-x/100})
    Logit.setdata(Xdata,Ydata)
    Logit.fit_model()
    auc_lst.append(Logit.auc)
```

In [191…

```python
plt.plot(pd.Series(auc_lst,index=(range(1,100,5))))
```

Out[191…

```
[<matplotlib.lines.Line2D at 0x126b7473880>]
```



这说明我们的权重参数设置较为合理

In [148…

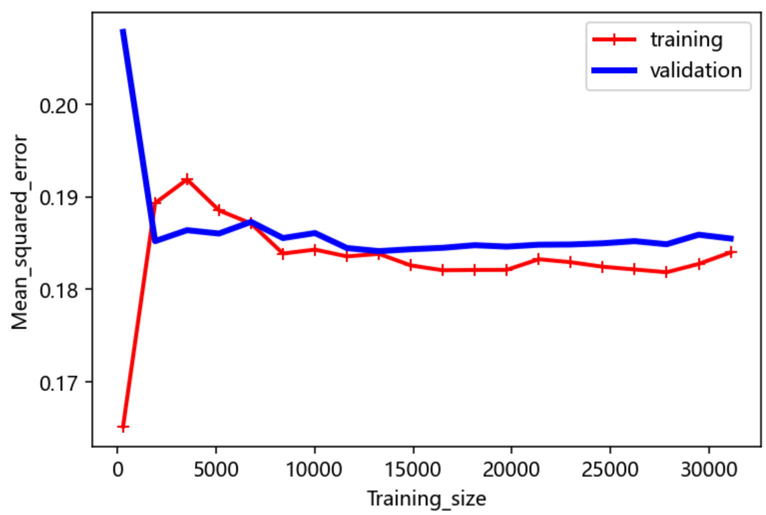
```
Classifiers_dict['LogisticRegression']
```

Out[148…

	Name	Options	Details
参数	penalty	l1,l2	处罚项，可选
	class_weight	{0:0.24,1:0.76}	计算损失函数时加权

In [370…

```
Logit.learning_curve_plot()
```



In [ ]:

```
learning_curve_plot(Logit.)
```

In [ ]:

In [ ]:

In [ ]:

In [174…

```
cross_val_score(Logit_Model_1,Logit_X_train_1,Logit_Y_train_1,scoring='accuracy',cv=10)
```

Out[174…

```
array([0.81094127, 0.80852776, 0.80008045, 0.80852776, 0.81053902,
       0.81617056, 0.807321  , 0.81898632, 0.80321932, 0.80482897])
```

In [178…

```
cross_val_score(Logit_Model_5,Logit_X_train_5,Logit_Y_train_5,scoring='accuracy',cv=5).mea
```

Out[178…

```
0.8558613732779647
```

In [179…

```
cross_val_score(XGB_Model_1,XGB_X_train_1,XGB_Y_train_1,scoring='accuracy',cv=5).mean()
```

Out[179…

```
0.8712686275490362
```

In [ ]:

```
XGB
```

Beta Regression (link = Canonical)

模型介绍

事实上，对概率建模，更优的选择是使用beta回归(虽然本次作业的目标是分类)。但还是那个问题，尽管一些模型能够进行统计推断，但它的预测能力有时不尽如人意。

这个实在没办法，python里似乎做不了beta回归。感兴趣的同学可以看看这个链接。

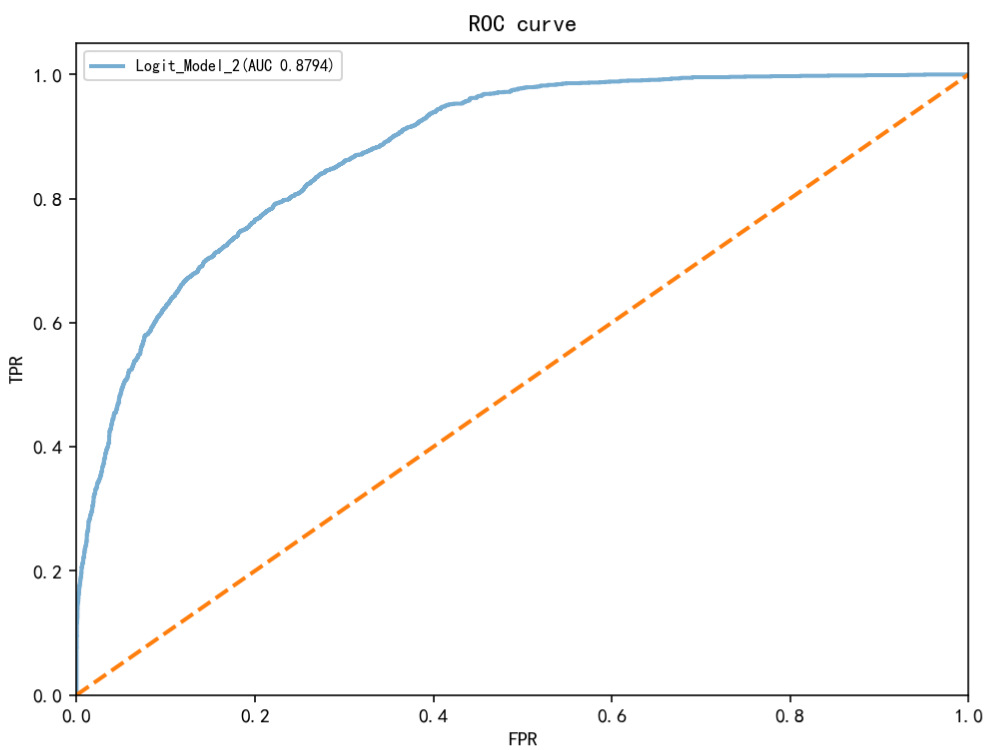
- 如何自行在python中定义Beta回归

In [239…

```
# 这里还是Logit回归------------------------
i = 2
model = 'Beta'
size_pct = 0.2
rn = 0
Xdata = ndf_train.iloc[:,1:]
Ydata = ndf_train.iloc[:,0]
globals()['{}_Model_{}'.format(model,i)] = LogisticRegression(penalty='l2',class_weight={(
#-----------------------------------------------------------------------------------------
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_test_{}'.format(model,i)],globa
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_valid_{}'.format(model,i)],glob
print(globals()['{}_X_train_{}'.format(model,i)].shape)
print(globals()['{}_X_valid_{}'.format(model,i)].shape)
print(globals()['{}_X_test_{}'.format(model,i)].shape)
# 拟合模型---------------------------------------------------------------------------------
globals()['{}_Model_{}'.format(model,i)].fit(globals()['{}_X_train_{}'.format(model,i)],gl
#在训练集上预测---------------------------------------------------------------------------
globals()['{}_Y_train_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,
print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_train
print("Classification report (training):\n {0}".format(classification_report(globals()['{}
# 在test集合上预测-------------------------------------------------------------------------
globals()['{}_Y_test_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i
print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_test_
print("Classification report (training):\n {0}".format(classification_report(globals()['{}
# 得到AUC ---------------------------------------------------------------------------------
globals()['{}_Y_test_score_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i)]
globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format(model,i)]
globals()['{}_Model_{}_roc_auc'.format(model,i)] = auc(globals()['{}_Model_{}_fpr'.format
plt.figure(figsize=(8,6))
plt.plot(globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
```

```
plt.legend(['{}_Model_{}'.format(model,i)+"(AUC {:.4f})".format(globals()['{}_Model_{}_roc
plt.show();
```

```
(24858, 107)
(6215, 107)
(7769, 107)
Confusion matrix (training):
 [[18706   171]
 [ 4548  1433]]

Classification report (training):
              precision    recall  f1-score   support

           0       0.80      0.99      0.89     18877
           1       0.89      0.24      0.38      5981

    accuracy                           0.81     24858
   macro avg       0.85      0.62      0.63     24858
weighted avg       0.83      0.81      0.77     24858

Confusion matrix (training):
 [[5859   59]
 [1410  441]]

Classification report (training):
              precision    recall  f1-score   support

           0       0.81      0.99      0.89      5918
           1       0.88      0.24      0.38      1851

    accuracy                           0.81      7769
   macro avg       0.84      0.61      0.63      7769
weighted avg       0.82      0.81      0.77      7769
```



In [ ]:

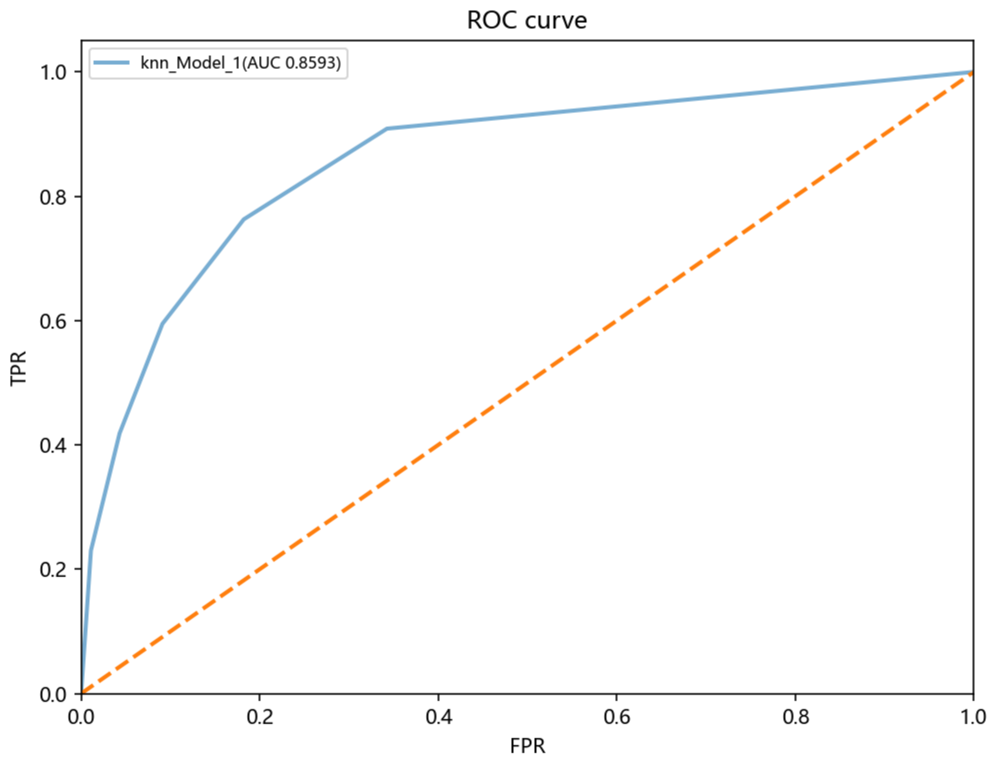
非参数模型

In [ ]:

KNN

In [144…

```
# 这里还是Logit回归------------------------
i = 1
model = 'knn'
size_pct = 0.2
rn = 0
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
globals()['{}_Model_{}'.format(model,i)] = KNeighborsClassifier()
#--------------------------------------------------------------------------------------------
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_test_{}'.format(model,i)],globa
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_valid_{}'.format(model,i)],glob
#print(globals()['{}_X_train_{}'.format(model,i)].shape)
#print(globals()['{}_X_valid_{}'.format(model,i)].shape)
#print(globals()['{}_X_test_{}'.format(model,i)].shape)
# 拟合模型--------------------------------------------------------------------------------
globals()['{}_Model_{}'.format(model,i)].fit(globals()['{}_X_train_{}'.format(model,i)],gl
#在训练集上预测--------------------------------------------------------------------------
globals()['{}_Y_train_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,
#print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_tra
#print("Classification report (training):\n {0}".format(classification_report(globals()['
# 在test集合上预测-------------------------------------------------------------------------
globals()['{}_Y_test_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i
#print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_tes
#print("Classification report (training):\n {0}".format(classification_report(globals()['
# 得到AUC ----------------------------------------------------------------------------------
globals()['{}_Y_test_score_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i)]
globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format(model,i)]
globals()['{}_Model_{}_roc_auc'.format(model,i)] = auc(globals()['{}_Model_{}_fpr'.format
plt.figure(figsize=(8,6))
plt.plot(globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.legend(['{}_Model_{}'.format(model,i)+"(AUC {:.4f})".format(globals()['{}_Model_{}_roc
plt.show();
```



In [ ]:

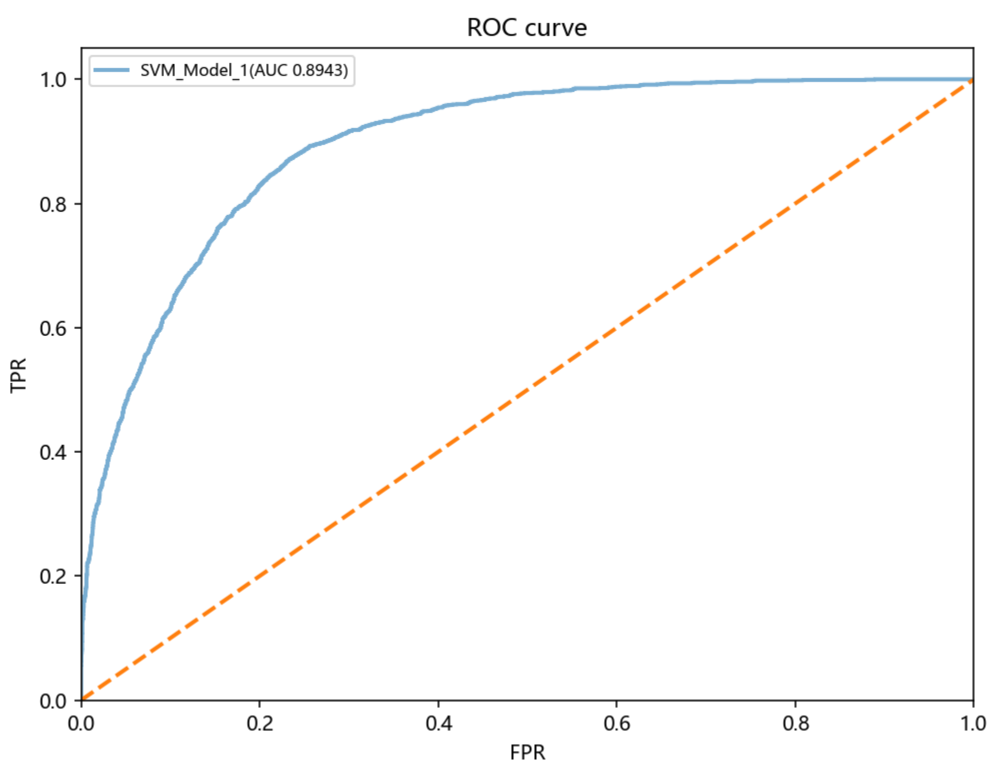
Support Vector Machine

In [163…

```
# SVM 1号------------------------
i = 1
model = 'SVM'
size_pct = 0.2
rn = 0
Xdata = nonlinear_smndf_train.iloc[:,1:]
Ydata = nonlinear_smndf_train.iloc[:,0]
globals()['{}_Model_{}'.format(model,i)] = SVC(probability=True)
#------------------------------------------------------------------------------------------
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_test_{}'.format(model,i)],globa
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_valid_{}'.format(model,i)],glob
print(globals()['{}_X_train_{}'.format(model,i)].shape)
print(globals()['{}_X_valid_{}'.format(model,i)].shape)
print(globals()['{}_X_test_{}'.format(model,i)].shape)
# 拟合模型----------------------------------------------------------------------------------
globals()['{}_Model_{}'.format(model,i)].fit(globals()['{}_X_train_{}'.format(model,i)],gl
#在训练集上预测-------------------------------------------------------------------------
globals()['{}_Y_train_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,
print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_train
print("Classification report (training):\n {0}".format(classification_report(globals()['{}
# 在test集合上预测---------------------------------------------------------------------
globals()['{}_Y_test_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i
print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_test_
print("Classification report (training):\n {0}".format(classification_report(globals()['{}
# 得到AUC --------------------------------------------------------------------------------
globals()['{}_Y_test_score_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i)]
globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format(model,i)]
globals()['{}_Model_{}_roc_auc'.format(model,i)] = auc(globals()['{}_Model_{}_fpr'.format
plt.figure(figsize=(8,6))
plt.plot(globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
```

```
plt.ylabel("TPR")
plt.title("ROC curve")
plt.legend(['{}_Model_{}'.format(model,i)+"(AUC {:.4f})".format(globals()['{}_Model_{}_roc
plt.show();
```

```
(24858, 117)
(6215, 117)
(7769, 117)
Confusion matrix (training):
 [[17688  1189]
 [ 2599  3382]]

Classification report (training):
              precision    recall  f1-score   support

           0       0.87      0.94      0.90     18877
           1       0.74      0.57      0.64      5981

    accuracy                           0.85     24858
   macro avg       0.81      0.75      0.77     24858
weighted avg       0.84      0.85      0.84     24858

Confusion matrix (training):
 [[5528  390]
 [ 866  985]]

Classification report (training):
              precision    recall  f1-score   support

           0       0.86      0.93      0.90      5918
           1       0.72      0.53      0.61      1851

    accuracy                           0.84      7769
   macro avg       0.79      0.73      0.75      7769
weighted avg       0.83      0.84      0.83      7769
```



XGBoost

- Xgboost实战

模型介绍

In [63]:
```
Classifiers_dict['XGBClassifier']
```

Out[63]:

	Name	Options	Details
参数	eval_metric	['logloss','auc','error']	避免报错
	eta	0.3	学习率

In [64]:
```
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
model_name = XGBClassifier
```

In [65]:
```
XGBoost = Classifiers(model_name)
XGBoost.setmodelparam(model_name,eval_metric=['logloss','auc','error'])
XGBoost.setdata(Xdata,Ydata)
XGBoost.fit_model()
```

与Logit分类器不同的是,XGBoost的precision(0)更小，precision(1)更高，这意味着他能更好的判断正样本。

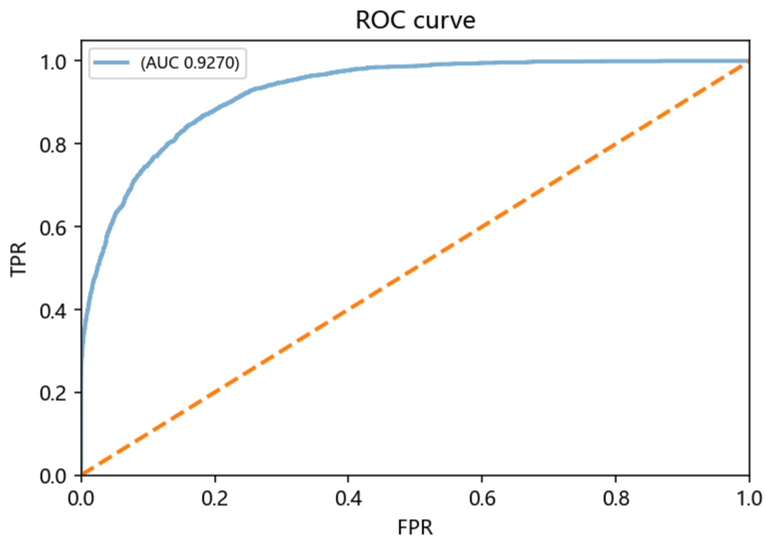
In [66]:
```
XGBoost.confusion_matrix_train()
```

```
Classification report (training):
              precision    recall  f1-score   support

           0       0.91      0.95      0.93     20680
           1       0.83      0.70      0.76      6509

    accuracy                           0.89     27189
   macro avg       0.87      0.83      0.85     27189
weighted avg       0.89      0.89      0.89     27189
```

In [67]:
```
XGBoost.confusion_matrix_test()
```

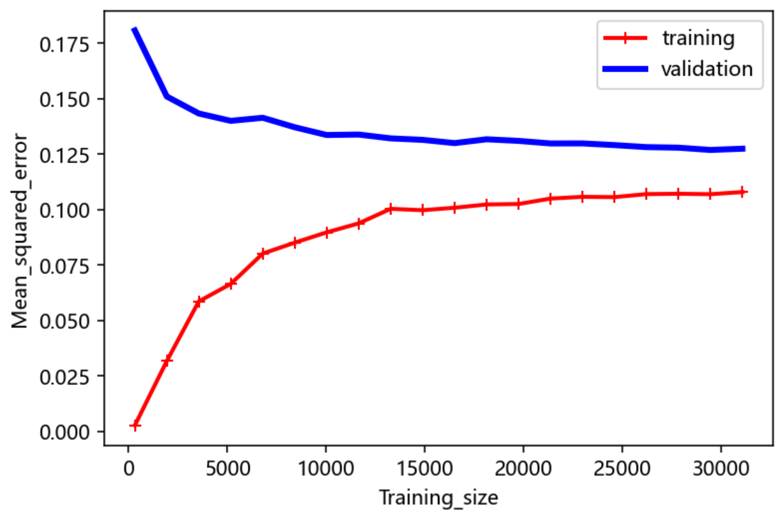
```
Classification report (testing):
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      8843
           1       0.77      0.65      0.71      2810

    accuracy                           0.87     11653
   macro avg       0.83      0.79      0.81     11653
weighted avg       0.86      0.87      0.87     11653
```

In [68]:
```
XGBoost.auc_plot()
```



In [386…

```
# 看图就行了，不要重新运行，很费时间。
XGBoost.learning_curve_plot(nmax=20)
```



相较于Logit回归,模型的均方误差降低了，但Valid与train之间的差异更大，这说明可能存在一些过拟合的情况。

参数调整

In [388…

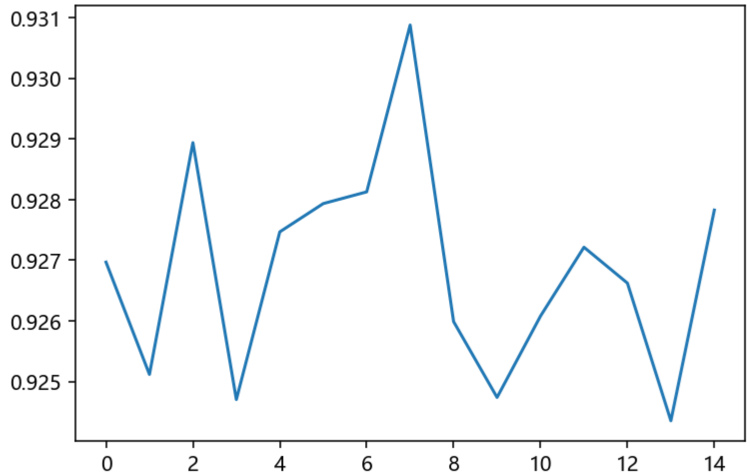
```
# 验证样本的稳定型
auc_lst = []
for j in list(range(15)):
    rn = j
    model_name = XGBClassifier
    temp = Classifiers(model_name)
    temp.setmodelparam(model_name,eval_metric=['logloss','auc','error'])
    temp.setdata(Xdata,Ydata)
    temp.fit_model()
    auc_lst.append(temp.auc)
```

In [389…

```
plt.plot(pd.Series(auc_lst))
```

Out[389…

```
[<matplotlib.lines.Line2D at 0x126bf5a1be0>]
```



AreaUnderRoc稳定在0.927上下

In [220…

```
XGBoost.model_fitted
```

Out[220…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              eval_metric=['logloss', 'auc', 'error'], gamma=0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

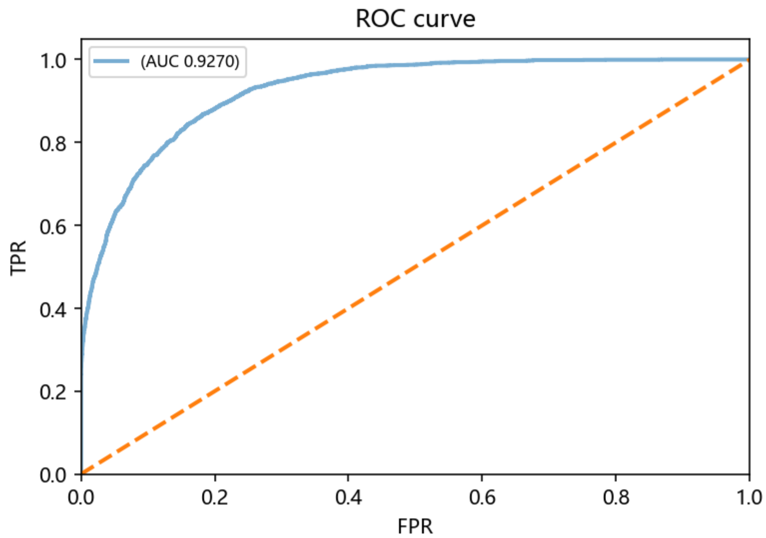
```
'booster': 'gbtree',
'objective': 'multi:softmax',  # 多分类的问题
'num_class': 10,               # 类别数，与 multisoftmax 并用
'gamma': 0.1,                  # 用于控制是否后剪枝的参数,越大越保守，一般0.1、0.2这样
子。
'max_depth': 12,               # 构建树的深度，越大越容易过拟合
'lambda': 2,                   # 控制模型复杂度的权重值的L2正则化项参数，参数越大，模型
越不容易过拟合。
'subsample': 0.7,              # 随机采样训练样本
'colsample_bytree': 0.7,       # 生成树时进行的列采样
'min_child_weight': 3,
'silent': 1,                   # 设置成1则没有运行信息输出，最好是设置为0.
'eta': 0.007,                  # 如同学习率
'seed': 1000,
'nthread': 4,                  # cpu 线程
```

In [289…

```
i = 1 #模型编号
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
model_name = XGBClassifier
globals()['XGBoost_{}'.format(i)] = Classifiers(model_name)
globals()['XGBoost_{}'.format(i)].setmodelparam(model_name,eval_metric=['logloss','auc','e
globals()['XGBoost_{}'.format(i)].setdata(Xdata,Ydata)
globals()['XGBoost_{}'.format(i)].fit_model()
```

In [269…

```
globals()['XGBoost_{}'.format(1)].auc_plot()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

LightGBM

模型介绍

In [230…

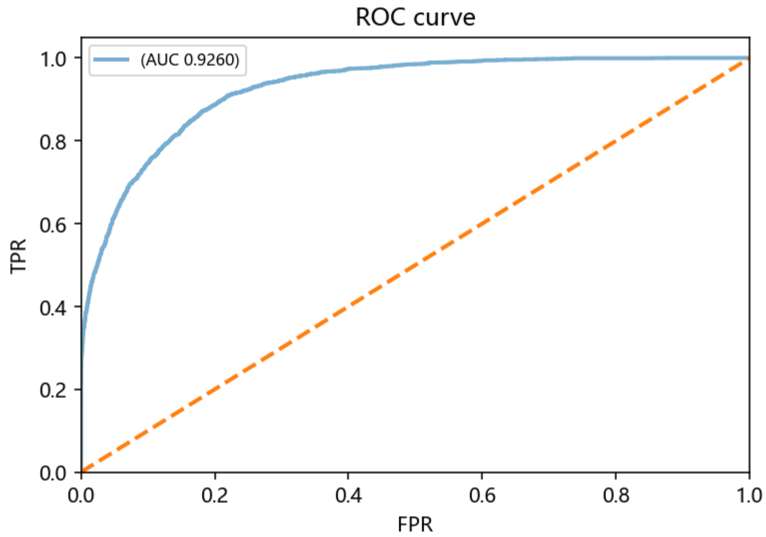
```
size_pct = 0.3 # 训练集的比例
rn = 1 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
model_name = LGBMClassifier
```

In [236…

```
Ltgbm = Classifiers(model_name)
# Ltgbm.setmodelparam(model_name,max_depth=6,num_leaves=35,bagging_fraction=0.6,feature_fi
#                                                    lambda_l1=0.7,lambda_l2=0)
Ltgbm.setmodelparam(model_name)
Ltgbm.setdata(Xdata,Ydata)
Ltgbm.fit_model()
```

In [237…

```
Ltgbm.auc_plot()
```



In [ ]:

In [ ]:

In [ ]:

In [206…

```
params={
    'max_depth':range(3,8,1),
    'num_leaves':range(5,100,5)
}
params1={
    'feature_fraction':[0.6,0.7,0.8,0.9,1],
    'bagging_fraction':[0.6,0.7,0.8,0.9,1]
}
params2={
    'lambda_l1':[0,0.1,0.3,0.5,0.7,0.8,0.9,1],
    'lambda_l2':[0,0.1,0.3,0.5,0.7,0.8,0.9,1]
}
```

In [207…

```
gsearch1 = GridSearchCV(estimator=LGBMClassifier(max_depth=6,num_leaves=35,bagging_fractic
                                                 lambda_l1=0.7,lambda_l2=0),
                        param_grid=params2,scoring='roc_auc',cv=5,n_jobs=-1)
gsearch1.fit(Ltgbm.X_train,Ltgbm.Y_train)
gsearch1.best_params_
```

Out[207…

```
GridSearchCV(cv=5,
             estimator=LGBMClassifier(bagging_fraction=0.6, feature_fraction=1,
                                      max_depth=6, num_leaves=35),
             n_jobs=-1,
             param_grid={'lambda_l1': [0, 0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 1],
                         'lambda_l2': [0, 0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 1]},
             scoring='roc_auc')
```

Out[207…

```
{'lambda_l1': 0.7, 'lambda_l2': 0}
```

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

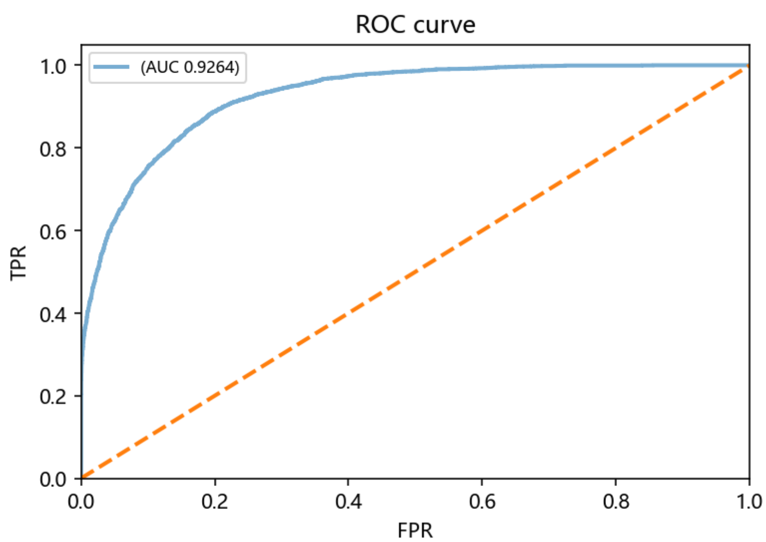
```

```

In [ ]:

```
Ltgbm.model_paramed.
```

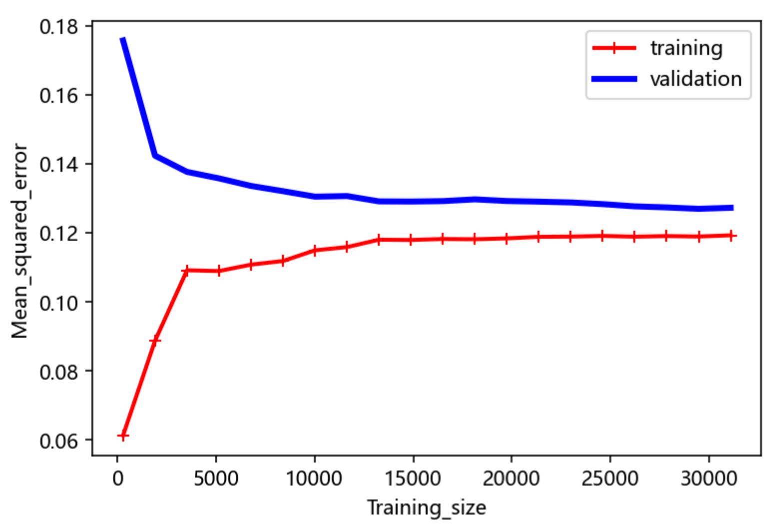
In [137…

```
Ltgbm.auc_plot()
```



In [210…

```
Ltgbm.learning_curve_plot(nmax=20)
```



LightGBM在test上的表现与XGBoost差不多，但训练集上LightGBM表现较差，但这说明过拟合问题并不严重。

参数调整

In [76]:

```
# 验证样本的稳定型
auc_lst = []
```

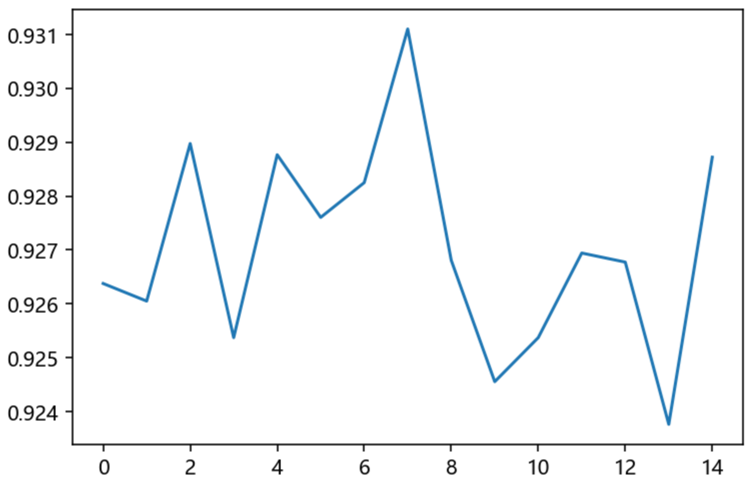
```
for j in list(range(15)):
    rn = j
    model_name = LGBMClassifier
    temp = Classifiers(model_name)
    temp.setmodelparam(model_name)
    temp.setdata(Xdata,Ydata)
    temp.fit_model()
    auc_lst.append(temp.auc)
```

In [77]:

```
plt.plot(auc_lst)
```

Out[77]:

```
[<matplotlib.lines.Line2D at 0x239d5c3daf0>]
```



In [ ]:

In [ ]:

In [ ]:

决策树

In [86]:

```
size_pct = 0.3 #  训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
model_name = DecisionTreeClassifier
```

In [87]:

```
DCTree = Classifiers(model_name)
DCTree.setmodelparam(model_name)
DCTree.setdata(Xdata,Ydata)
DCTree.fit_model()
```

In [89]:

```
DCTree.confusion_matrix_train()
```

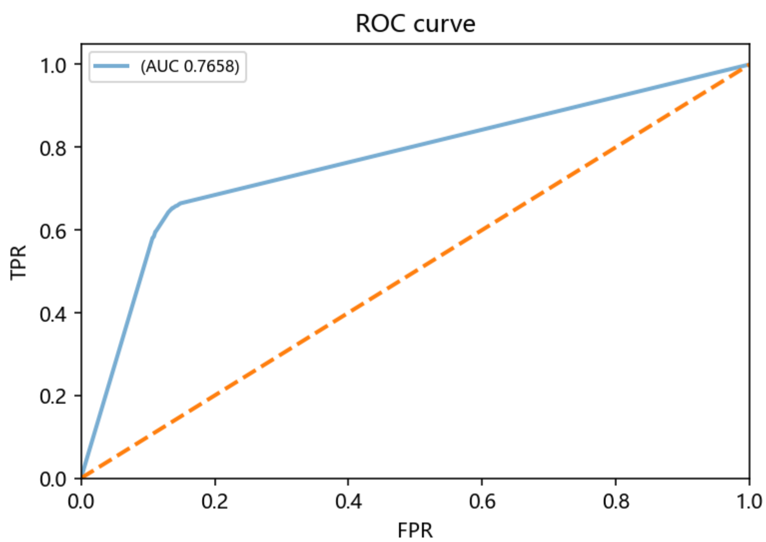
```
Classification report (training):
               precision    recall  f1-score   support

           0       0.98      1.00      0.99     20680
           1       0.98      0.92      0.95      6509
```

```
    accuracy                           0.98     27189
   macro avg       0.98      0.96      0.97     27189
weighted avg       0.98      0.98      0.98     27189
```

In [90]:
```
DCTree.confusion_matrix_test()
```

```
Classification report (testing):
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      8843
           1       0.63      0.60      0.61      2810

    accuracy                           0.82     11653
   macro avg       0.75      0.74      0.75     11653
weighted avg       0.81      0.82      0.82     11653
```

In [91]:
```
DCTree.auc_plot()
```



In [ ]:

Random Forest

In [92]:
```
size_pct = 0.3 #  训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
model_name = RandomForestClassifier
```

In [93]:
```
RDForest = Classifiers(model_name)
RDForest.setmodelparam(model_name)
RDForest.setdata(Xdata,Ydata)
RDForest.fit_model()
```

In [94]:
```
RDForest.confusion_matrix_train()
```

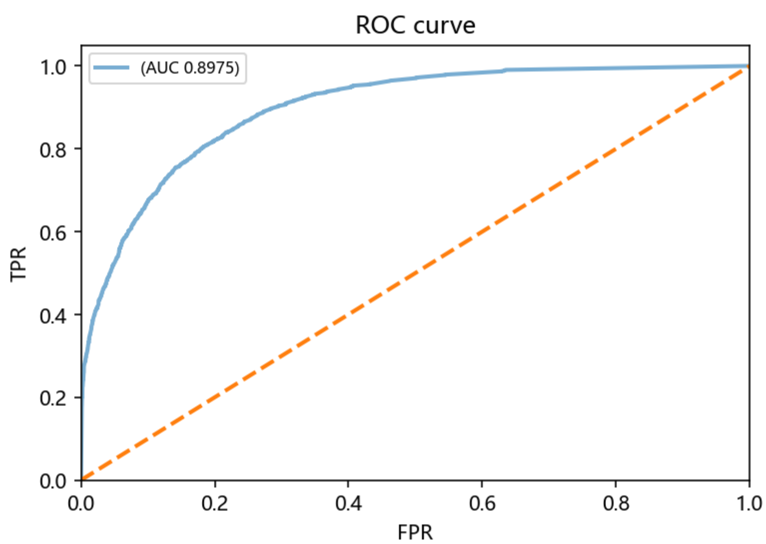
```
Classification report (training):
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     20680
```

```
           1       0.96      0.94      0.95      6509

    accuracy                           0.98     27189
   macro avg       0.97      0.97      0.97     27189
weighted avg       0.98      0.98      0.98     27189
```

In [95]:
```
RDForest.confusion_matrix_test()
```

```
Classification report (testing):
               precision    recall  f1-score   support

           0       0.88      0.92      0.90      8843
           1       0.71      0.62      0.66      2810

    accuracy                           0.85     11653
   macro avg       0.80      0.77      0.78     11653
weighted avg       0.84      0.85      0.84     11653
```

In [96]:
```
RDForest.auc_plot()
```



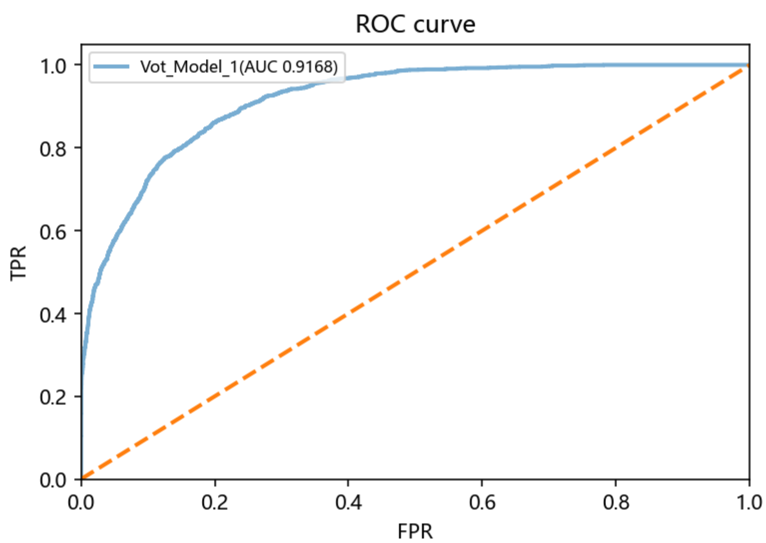
In [ ]:

In [ ]:

VotingClassifier

In [182…
```
i = 1 #模型编号
model = 'Vot' # 模型名称
size_pct = 0.2 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]

# 模型定义
globals()['{}_Model_{}'.format(model,i)] = VotingClassifier(estimators=[('lr',Logit_Model_
#-----------------------------------------------------------------------------------------
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_test_{}'.format(model,i)],globa
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_valid_{}'.format(model,i)],glob
#print(globals()['{}_X_train_{}'.format(model,i)].shape)
#print(globals()['{}_X_valid_{}'.format(model,i)].shape)
```

```
#print(globals()['{}_X_test_{}'.format(model,i)].shape)
# 拟合模型-----------------------------------------------------------------------
globals()['{}_Model_{}'.format(model,i)].fit(globals()['{}_X_train_{}'.format(model,i)],gl
#在训练集上预测----------------------------------------------------------------------
globals()['{}_Y_train_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,
#print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_tra
#print("Classification report (training):\n {0}".format(classification_report(globals()['
# 在test集合上预测-------------------------------------------------------------------
globals()['{}_Y_test_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i
#print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_test
#print("Classification report (training):\n {0}".format(classification_report(globals()['
# 得到AUC ------------------------------------------------------------------------
globals()['{}_Y_test_score_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i)]
globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format(model,i)]
globals()['{}_Model_{}_roc_auc'.format(model,i)] = auc(globals()['{}_Model_{}_fpr'.format
#plt.figure(figsize=(8,6))
plt.plot(globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.legend(['{}_Model_{}'.format(model,i)+"(AUC {:.4f})".format(globals()['{}_Model_{}_roc
plt.show();
```



In [ ]:

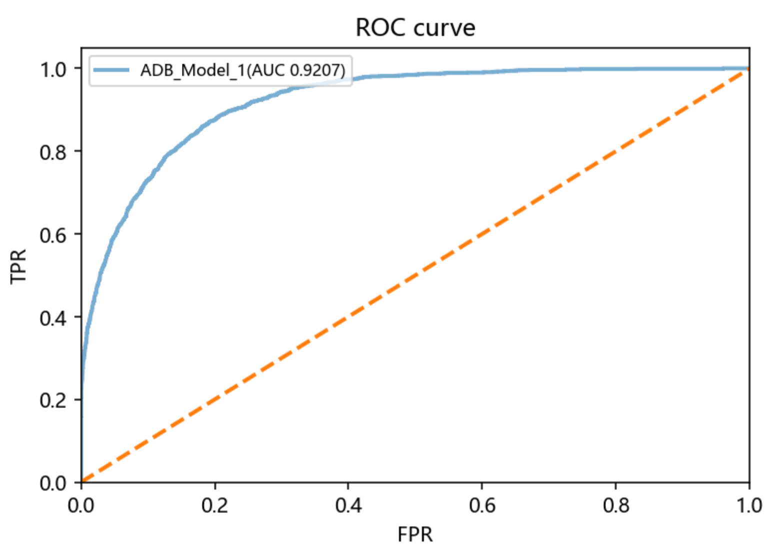
In [ ]:

ADB

In [189…

```
i = 1 #模型编号
model = 'ADB' # 模型名称
size_pct = 0.2 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]

# 模型定义
globals()['{}_Model_{}'.format(model,i)] = AdaBoostClassifier(DecisionTreeClassifier(max_d
algorithm="SAMME.R", learning_rate=0.5)
#-------------------------------------------------------------------------------
```

```
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_test_{}'.format(model,i)],globa
globals()['{}_X_train_{}'.format(model,i)],globals()['{}_X_valid_{}'.format(model,i)],glob
#print(globals()['{}_X_train_{}'.format(model,i)].shape)
#print(globals()['{}_X_valid_{}'.format(model,i)].shape)
#print(globals()['{}_X_test_{}'.format(model,i)].shape)
# 拟合模型-----------------------------------------------------------------------------------
globals()['{}_Model_{}'.format(model,i)].fit(globals()['{}_X_train_{}'.format(model,i)],gl
#在训练集上预测-------------------------------------------------------------------------------
globals()['{}_Y_train_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,
#print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_tra
#print("Classification report (training):\n {0}".format(classification_report(globals()['
# 在test集合上预测---------------------------------------------------------------------------
globals()['{}_Y_test_predict_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i
#print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(globals()['{}_Y_tes
#print("Classification report (training):\n {0}".format(classification_report(globals()['
# 得到AUC -----------------------------------------------------------------------------------
globals()['{}_Y_test_score_{}'.format(model,i)] = globals()['{}_Model_{}'.format(model,i)]
globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format(model,i)]
globals()['{}_Model_{}_roc_auc'.format(model,i)] = auc(globals()['{}_Model_{}_fpr'.format
#plt.figure(figsize=(8,6))
plt.plot(globals()['{}_Model_{}_fpr'.format(model,i)], globals()['{}_Model_{}_tpr'.format
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.legend(['{}_Model_{}'.format(model,i)+"(AUC {:.4f})".format(globals()['{}_Model_{}_roc
plt.show();
```



In [ ]:

GTB

In [80]:

```
size_pct = 0.3 # 训练集的比例
rn = 0 #随机种子号
Xdata = nonlinear_sndf_train.iloc[:,1:]
Ydata = nonlinear_sndf_train.iloc[:,0]
model_name = GradientBoostingClassifier
```

In [81]:

```
GTBoost = Classifiers(model_name)
GTBoost.setmodelparam(model_name)
```

```
GTBoost.setdata(Xdata,Ydata)
GTBoost.fit_model()
```

In [84]:

```
GTBoost.confusion_matrix_train()
```

```
Classification report (training):
              precision    recall  f1-score   support

           0       0.89      0.95      0.92     20680
           1       0.80      0.61      0.69      6509

    accuracy                           0.87     27189
   macro avg       0.84      0.78      0.81     27189
weighted avg       0.87      0.87      0.86     27189
```

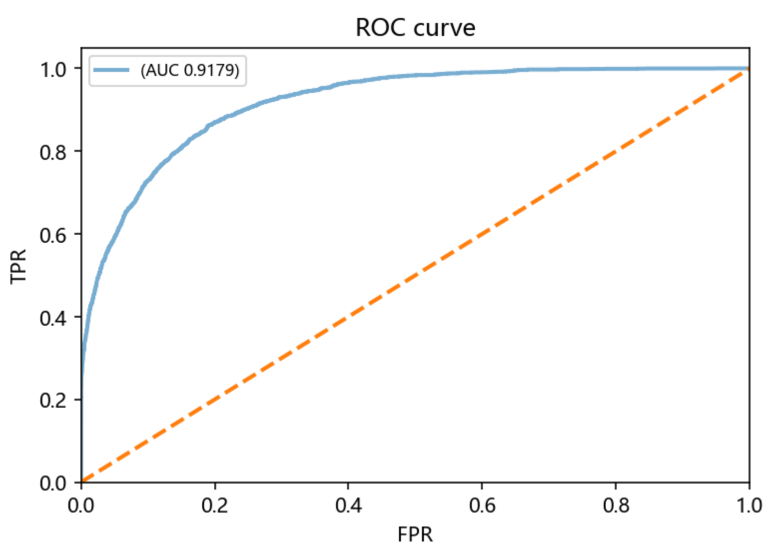
In [82]:

```
GTBoost.confusion_matrix_test()
```

```
Classification report (testing):
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      8843
           1       0.79      0.59      0.68      2810

    accuracy                           0.86     11653
   macro avg       0.83      0.77      0.79     11653
weighted avg       0.86      0.86      0.86     11653
```

In [85]:

```
GTBoost.auc_plot()
```



In [ ]:

```
```

超出能力范围的模型

123

In [222…

```
# housing = fetch_california_housing()
# X_train_full, X_test, y_train_full, y_test = train_test_split(
# housing.data, housing.target)
# X_train, X_valid, y_train, y_valid = train_test_split(
```

```
# X_train_full, y_train_full)
# scaler = StandardScaler()
# X_train_scaled = scaler.fit_transform(X_train)
# X_valid_scaled = scaler.transform(X_valid)
# X_test_scaled = scaler.transform(X_test)
```

In [224…

```
# model = keras.models.Sequential([
# keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
# keras.layers.Dense(1)
# ])
# model.compile(loss="mean_squared_error", optimizer="sgd")
# history = model.fit(X_train, y_train, epochs=20,
# validation_data=(X_valid, y_valid))
# mse_test = model.evaluate(X_test, y_test)
# X_new = X_test[:3] # pretend these are new instances
# y_pred = model.predict(X_new)
```

```
Epoch 1/20
363/363 [==============================] - 1s 1ms/step - loss: nan - val_loss: nan
Epoch 2/20
363/363 [==============================] - 0s 867us/step - loss: nan - val_loss: nan
Epoch 3/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 4/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 5/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 6/20
363/363 [==============================] - 1s 1ms/step - loss: nan - val_loss: nan
Epoch 7/20
363/363 [==============================] - 1s 1ms/step - loss: nan - val_loss: nan
Epoch 8/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
Epoch 9/20
363/363 [==============================] - 1s 1ms/step - loss: nan - val_loss: nan
Epoch 10/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
Epoch 11/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 12/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 13/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 14/20
363/363 [==============================] - 0s 1ms/step - loss: nan - val_loss: nan
Epoch 15/20
363/363 [==============================] - 1s 1ms/step - loss: nan - val_loss: nan
Epoch 16/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
Epoch 17/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
Epoch 18/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
Epoch 19/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
Epoch 20/20
363/363 [==============================] - 1s 2ms/step - loss: nan - val_loss: nan
162/162 [==============================] - 0s 1ms/step - loss: nan
1/1 [==============================] - 0s 54ms/step
```

In [ ]:

```
#######################
```

In [422…

```
X_valid = Logit_X_valid_5
X_train = Logit_X_train_5
X_test = Logit_X_test_5
y_valid = Logit_Y_valid_5
y_train = Logit_Y_train_5
y_test = Logit_Y_test_5
```

In [425…

```
model = keras.models.Sequential([
keras.layers.Flatten(input_shape=[117]),
keras.layers.Dense(600, activation="relu"),
keras.layers.Dense(200, activation="relu"),
keras.layers.Dense(2, activation="softmax")
])
```

In [428…

```
model.compile(loss="sparse_categorical_crossentropy",
optimizer="sgd",
metrics=["accuracy"])
```

In [429…

```
history = model.fit(X_train, y_train, epochs=30,validation_data=(X_valid, y_valid))
```

```
Epoch 1/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3766 - accuracy: 0.8252 -
val_loss: 0.3420 - val_accuracy: 0.8415
Epoch 2/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3255 - accuracy: 0.8462 -
val_loss: 0.3274 - val_accuracy: 0.8508
Epoch 3/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3145 - accuracy: 0.8530 -
val_loss: 0.3205 - val_accuracy: 0.8549
Epoch 4/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3092 - accuracy: 0.8563 -
val_loss: 0.3193 - val_accuracy: 0.8555
Epoch 5/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3073 - accuracy: 0.8574 -
val_loss: 0.3177 - val_accuracy: 0.8553
Epoch 6/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3056 - accuracy: 0.8582 -
val_loss: 0.3174 - val_accuracy: 0.8550
Epoch 7/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3040 - accuracy: 0.8600 -
val_loss: 0.3160 - val_accuracy: 0.8553
Epoch 8/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3036 - accuracy: 0.8590 -
val_loss: 0.3163 - val_accuracy: 0.8558
Epoch 9/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3025 - accuracy: 0.8600 -
val_loss: 0.3155 - val_accuracy: 0.8555
Epoch 10/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3015 - accuracy: 0.8606 -
val_loss: 0.3206 - val_accuracy: 0.8512
Epoch 11/30
777/777 [==============================] - 2s 3ms/step - loss: 0.3011 - accuracy: 0.8601 -
val_loss: 0.3153 - val_accuracy: 0.8545
Epoch 12/30
777/777 [==============================] - 2s 2ms/step - loss: 0.3000 - accuracy: 0.8613 -
val_loss: 0.3157 - val_accuracy: 0.8537
Epoch 13/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2990 - accuracy: 0.8628 -
val_loss: 0.3196 - val_accuracy: 0.8520
Epoch 14/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2987 - accuracy: 0.8613 -
```

```
val_loss: 0.3145 - val_accuracy: 0.8541
Epoch 15/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2983 - accuracy: 0.8622 -
val_loss: 0.3151 - val_accuracy: 0.8549
Epoch 16/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2976 - accuracy: 0.8615 -
val_loss: 0.3154 - val_accuracy: 0.8560
Epoch 17/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2969 - accuracy: 0.8623 -
val_loss: 0.3216 - val_accuracy: 0.8468
Epoch 18/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2967 - accuracy: 0.8631 -
val_loss: 0.3171 - val_accuracy: 0.8574
Epoch 19/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2960 - accuracy: 0.8639 -
val_loss: 0.3142 - val_accuracy: 0.8534
Epoch 20/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2951 - accuracy: 0.8640 -
val_loss: 0.3168 - val_accuracy: 0.8523
Epoch 21/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2949 - accuracy: 0.8627 -
val_loss: 0.3150 - val_accuracy: 0.8550
Epoch 22/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2947 - accuracy: 0.8633 -
val_loss: 0.3145 - val_accuracy: 0.8549
Epoch 23/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2938 - accuracy: 0.8639 -
val_loss: 0.3174 - val_accuracy: 0.8555
Epoch 24/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2933 - accuracy: 0.8641 -
val_loss: 0.3143 - val_accuracy: 0.8541
Epoch 25/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2928 - accuracy: 0.8658 -
val_loss: 0.3143 - val_accuracy: 0.8558
Epoch 26/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2919 - accuracy: 0.8660 -
val_loss: 0.3203 - val_accuracy: 0.8531
Epoch 27/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2917 - accuracy: 0.8643 -
val_loss: 0.3168 - val_accuracy: 0.8542
Epoch 28/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2916 - accuracy: 0.8648 -
val_loss: 0.3162 - val_accuracy: 0.8526
Epoch 29/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2909 - accuracy: 0.8660 -
val_loss: 0.3150 - val_accuracy: 0.8536
Epoch 30/30
777/777 [==============================] - 2s 3ms/step - loss: 0.2903 - accuracy: 0.8658 -
val_loss: 0.3189 - val_accuracy: 0.8499
```

In [430…

```
model.evaluate(X_test, y_test)
```

```
243/243 [==============================] - 0s 1ms/step - loss: 0.3055 - accuracy: 0.8547
```

Out[430…

$[0.305548638105392, 0.854678869247437]$

In [431…

```
y_proba = model.predict(X_test)
y_proba
```

```
243/243 [==============================] - 0s 1ms/step
```

Out[431…

```
array([[9.9960917e-01, 3.9089634e-04],
       [1.2657326e-01, 8.7342674e-01],
       [9.9479383e-01, 5.2061444e-03],
       ...,
```

```
       [9.0555042e-01, 9.4449580e-02],
       [9.9907184e-01, 9.2815049e-04],
       [2.1499975e-01, 7.8500021e-01]], dtype=float32)
```

In [432…

```
model.predict_generator(X_test)
```

Out[432…

```
array([[9.9960917e-01, 3.9089634e-04],
       [1.2657326e-01, 8.7342674e-01],
       [9.9479383e-01, 5.2061444e-03],
       ...,
       [9.0555042e-01, 9.4449580e-02],
       [9.9907184e-01, 9.2815049e-04],
       [2.1499975e-01, 7.8500021e-01]], dtype=float32)
```

In [433…

```
y_test_pred = np.argmax(model.predict(X_test), axis=-1)
y_train_pred = np.argmax(model.predict(X_train), axis=-1)
```

```
243/243 [==============================] - 0s 1ms/step
777/777 [==============================] - 1s 2ms/step
```

In [ ]:

In [434…

```
print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(y_train, y_train_prec
print("Classification report (training):\n {0}".format(classification_report(y_train, y_tı
```

```
Confusion matrix (training):
 [[17255  1622]
 [ 1776  4205]]

Classification report (training):
               precision    recall  f1-score   support

           0       0.91      0.91      0.91     18877
           1       0.72      0.70      0.71      5981

    accuracy                           0.86     24858
   macro avg       0.81      0.81      0.81     24858
weighted avg       0.86      0.86      0.86     24858
```

In [435…

```
print("Confusion matrix (training):\n {0}\n".format(confusion_matrix(y_test, y_test_pred))
print("Classification report (training):\n {0}".format(classification_report(y_test, y_tes
```

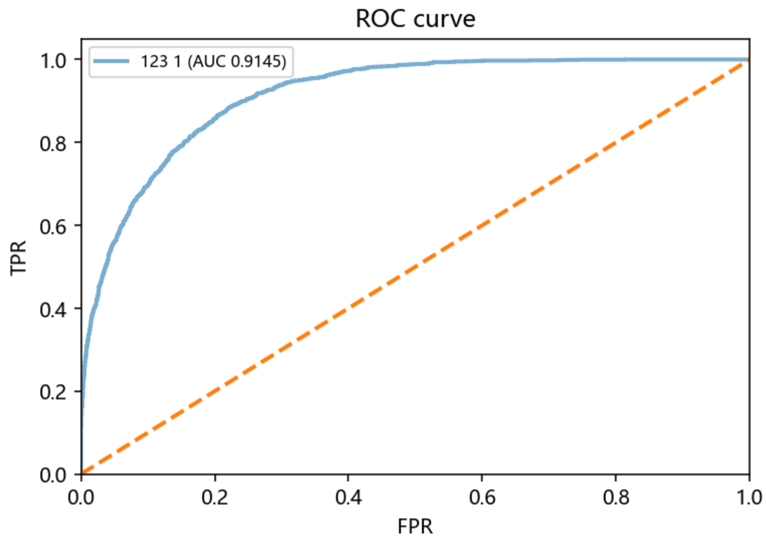
```
Confusion matrix (training):
 [[5390  528]
 [ 601 1250]]

Classification report (training):
               precision    recall  f1-score   support

           0       0.90      0.91      0.91      5918
           1       0.70      0.68      0.69      1851

    accuracy                           0.85      7769
   macro avg       0.80      0.79      0.80      7769
weighted avg       0.85      0.85      0.85      7769
```

In [436…

```
y1_valid_score_lr1 = model.predict(X_test)
fpr_lr1, tpr_lr1, thresholds_lr1 = roc_curve(y_test, y1_valid_score_lr1[:, 1])
roc_auc_lr1 = auc(fpr_lr1, tpr_lr1)
```

```
plt.plot(fpr_lr1, tpr_lr1, lw=2, alpha=.6)
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.legend(["123 1 (AUC {:.4f})".format(roc_auc_lr1)], fontsize=8, loc=2)
plt.show();
```

```
243/243 [==============================] - 0s 1ms/step
```



In [ ]:

Multi-Layer Perceptrons

123

In [ ]:

In [ ]:

In [ ]:

模型对比

In [492…

```
??pd.DataFrame.drop_duplicates
```

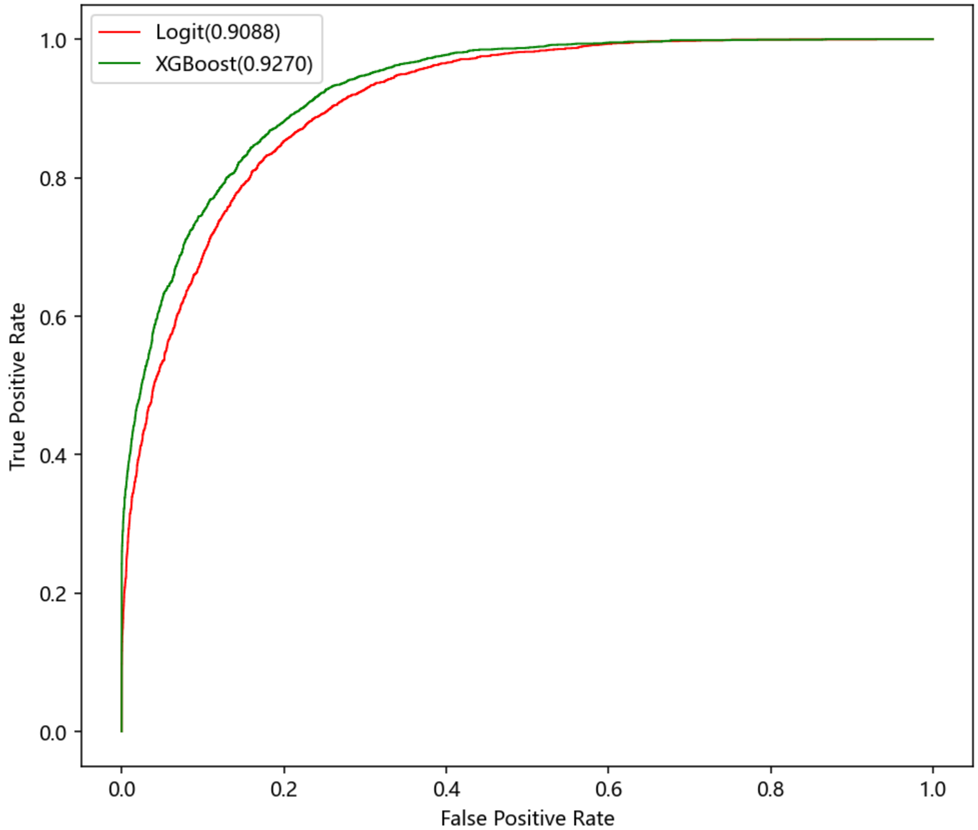
In [519…

```
A = pd.DataFrame([Logit.fpr,Logit.tpr]).transpose()
A.columns = ['fpr','tpr']
A = A.drop_duplicates('fpr')
```

In [70]:

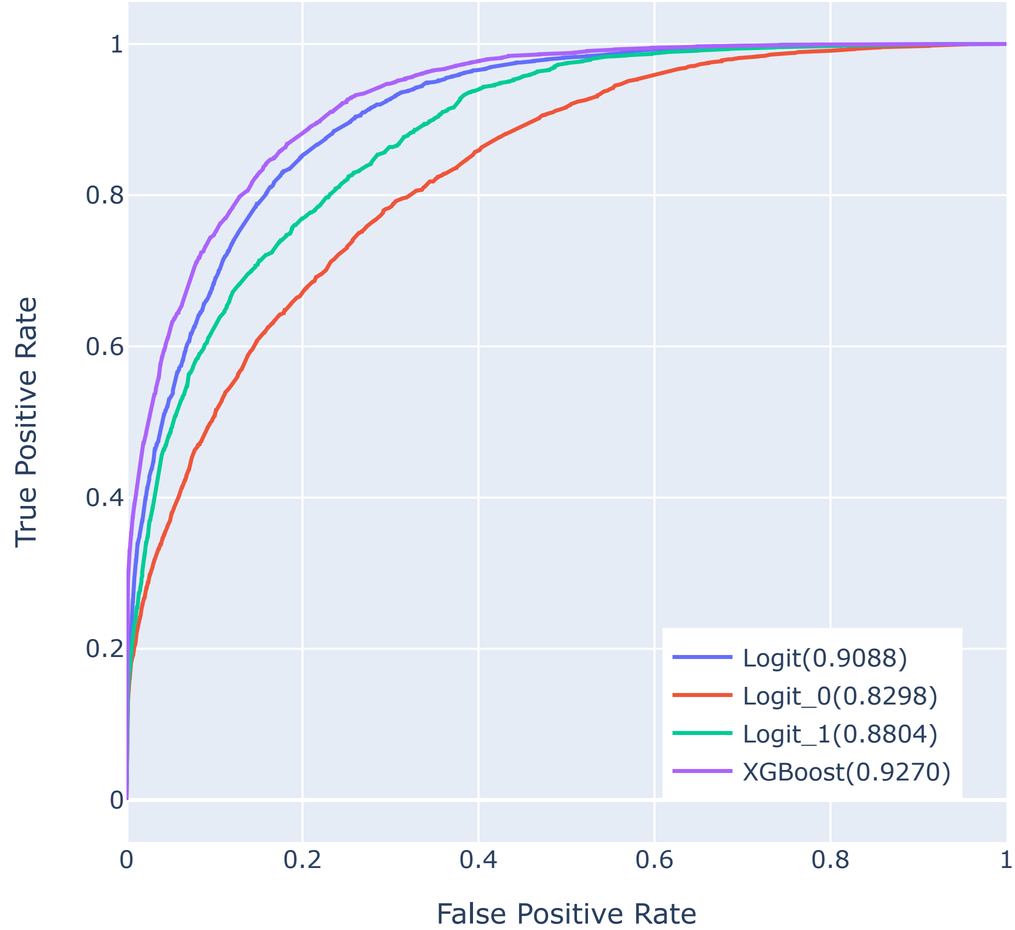
```
plt.figure(figsize=(8,7))
plt.plot(Logit.fpr,Logit.tpr,color='r',linewidth=1, label="Logit({:.4f})".format(Logit.auc
plt.plot(XGBoost.fpr,XGBoost.tpr,color='g',linewidth=1, label="XGBoost({:.4f})".format(XGB
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.legend()
plt.savefig(sys.path[0]+'test.svg', dpi=5000, bbox_inches='tight',format="svg") #设置输出位
plt.show();
```



In [103…

```
import plotly.graph_objects as go
# Create traces
fig = go.Figure()
fig.add_trace(go.Scatter(x=Logit.fpr, y=Logit.tpr,
                    mode='lines',
                    name="Logit({:.4f})".format(Logit.auc)))
fig.add_trace(go.Scatter(x=Logit_0.fpr, y=Logit_0.tpr,
                    mode='lines',
                    name="Logit_0({:.4f})".format(Logit_0.auc)))
fig.add_trace(go.Scatter(x=Logit_1.fpr, y=Logit_1.tpr,
                    mode='lines',
                    name="Logit_1({:.4f})".format(Logit_1.auc)))
fig.add_trace(go.Scatter(x=XGBoost.fpr, y=XGBoost.tpr,
                    mode='lines',
                    name="XGBoost({:.4f})".format(XGBoost.auc)))
fig.update_layout(
    #title="",
    xaxis_title="False Positive Rate",
    yaxis_title="True Positive Rate",
    autosize = False,
    width = 600,
    height = 600,
    legend=dict(traceorder="normal",yanchor="auto",xanchor="right",x=0.95,y=0.05,
               orientation="v")
)
fig.show();
```



In [ ]:

test

In [101…

```
df_train
```

Out[101…

	年龄	工作情况	教育	教育时间	婚姻状况	职业类型	家庭角色	民族	性别	投资收入	投资损失	工作天数	省份	Y
0	35	个体	初三	5	已婚平民配偶	其他职业	丈夫	民族D	男	0	0	40	省份22	0
1	37	中央部委	高中生	9	已婚平民配偶	保安	丈夫	民族D	男	0	0	40	省份8	0
2	19	个体	初三	5	未婚	手工艺维修	孩子	民族D	男	0	0	20	省份8	0
3	33	个体	大学生	13	已婚平民配偶	专业技术	丈夫	民族D	男	0	0	60	省份8	1
4	22	个体	大学未毕业	10	未婚	手工艺维修	离家	民族D	男	0	0	40	省份8	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
38837	34	个体	大学生	13	已婚平民配偶	专业技术	妻子	民族A	女	0	0	35	省份8	0
38838	39	个体	高中生	9	已婚平民配偶	机械操作	丈夫	民族D	男	0	0	40	省份8	1

	年龄	工作情况	教育	教育时间	婚姻状况	职业类型	家庭角色	民族	性别	投资收入	投资损失	工作天数	省份	Y
38839	51	个体	高中生	9	离婚	手工艺维修	离家	民族D	男	0	0	40	省份8	0
38840	25	个体	初三	5	未婚	管理文书	未婚	民族D	女	0	0	40	省份22	0
38841	34	个体	高中生	9	已婚平民配偶	技术支持	丈夫	民族D	男	0	0	40	省份8	1

38842 rows × 14 columns

In [136…

```
df_test = pd.read_csv('Test.csv')
df_test = df_test.reindex(list(df_train),axis=1,fill_value=0)
df_test = data_clean(df_test)
nonlinear_sndf_test = df_test.reindex(list(nonlinear_ndf_train.columns),axis=1,fill_value=
```

In [137…

```
nonlinear_ndf_test
```

Out[137…

	Y	工作天数	年龄	投资损失	投资收入	教育时间	工作天数**2	年龄**2	投资损失**2	投资收入**2	...	省份_省份39	省份_省份4
0	0	-0.045190	-0.918487	-0.212965	-0.143201	-0.405521	-0.177469	-0.837661	-0.196427	-0.076878	...	0	0
1	0	-1.266554	-1.358822	-0.212965	-0.143201	-0.012727	-1.062434	-1.070467	-0.196427	-0.076878	...	0	0
2	0	-0.045190	-0.184596	-0.212965	-0.143201	1.165658	-0.177469	-0.314691	-0.196427	-0.076878	...	0	0
3	0	-0.045190	0.475907	-0.212965	-0.143201	0.380068	-0.177469	0.300221	-0.196427	-0.076878	...	0	0
4	0	-0.045190	0.182350	-0.212965	-0.143201	-0.405521	-0.177469	0.010057	-0.196427	-0.076878	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0	-0.452312	1.063020	-0.212965	-0.143201	-0.405521	-0.517840	0.961525	-0.196427	-0.076878	...	0	0
9996	0	-0.045190	1.576744	-0.212965	-0.143201	-2.369495	-0.177469	1.628734	-0.196427	-0.076878	...	0	0
9997	0	1.583294	-1.212044	-0.212965	-0.143201	-0.405521	1.637843	-0.999613	-0.196427	-0.076878	...	0	0
9998	0	-0.452312	-0.184596	-0.212965	-0.143201	-2.369495	-0.517840	-0.314691	-0.196427	-0.076878	...	0	0
9999	0	3.211779	-0.331374	-0.212965	-0.143201	-0.405521	4.179280	-0.432781	-0.196427	-0.076878	...	0	0

10000 rows × 118 columns

In [135…

```
nonlinear_sndf_train
```

Out[135…

	Y	工作天数	年龄	投资损失	投资收入	教育时间	工作天数**2	年龄**2	投资损失**2	投资收入**2	...	省份_省份39
0	0	-0.031260	-0.267737	-0.218191	-0.145249	-1.974877	-0.170105	-0.386918	-0.206955	-0.081162	...	0

	Y	工作天数	年龄	投资损失	投资收入	教育时间	工作天数**2	年龄**2	投资损失**2	投资收入**2	...	省份_省份39
1	0	-0.031260	-0.122094	-0.218191	-0.145249	-0.422865	-0.170105	-0.265650	-0.206955	-0.081162	...	0
2	0	-1.641624	-1.432885	-0.218191	-0.145249	-1.974877	-1.270804	-1.114525	-0.206955	-0.081162	...	0
3	1	1.579103	-0.413381	-0.218191	-0.145249	1.129146	1.664393	-0.501449	-0.206955	-0.081162	...	0
4	0	-0.031260	-1.214420	-0.218191	-0.145249	-0.034863	-0.170105	-1.010942	-0.206955	-0.081162	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
38837	0	-0.433851	-0.340559	-0.218191	-0.145249	1.129146	-0.514073	-0.445026	-0.206955	-0.081162	...	0
38838	1	-0.031260	0.023550	-0.218191	-0.145249	-0.422865	-0.170105	-0.137646	-0.206955	-0.081162	...	0
38839	0	-0.031260	0.897410	-0.218191	-0.145249	-0.422865	-0.170105	0.771863	-0.206955	-0.081162	...	0
38840	0	-0.031260	-0.995955	-0.218191	-0.145249	-1.974877	-0.170105	-0.892201	-0.206955	-0.081162	...	0
38841	1	-0.031260	-0.340559	-0.218191	-0.145249	-0.422865	-0.170105	-0.445026	-0.206955	-0.081162	...	0

38842 rows × 118 columns

```
In [ ]:
```