# 1.变量绘图需求

# 分析离职和受教育程度的分析sns.countplot(x="Attrition", hue="Education", data=talent\_data)

# 1.离职和年龄的关系sns.boxplot(x="Attrition", y="Age", data=talent\_data)  
plt.subplot(222)  
# 2.离职和家庭和距离之间的关系sns.boxplot(x="Attrition", y="DistanceFromHome", data=talent\_data)  
plt.subplot(223)  
# 3.离职和月收入的关系sns.boxplot(x="Attrition", y="MonthlyIncome", data=talent\_data)  
plt.subplot(224)  
# 4.离职和曾经工作公司的关系sns.boxplot(x="Attrition", y="NumCompaniesWorked", data=talent\_data)  
# 离职和婚姻状况分析sns.countplot(x="Attrition", hue="MaritalStatus", data=talent\_data)  
plt.subplot(212)  
# 离职和性别关系sns.countplot(x="Attrition", hue="Gender", data=talent\_data)

# 多变量之间关系

def show\_pairplot():  
 sns.pairplot(data=talent\_data, hue="Attrition",  
 vars=["Age", "DistanceFromHome", "JobLevel", "PerformanceRating", "WorkLifeBalance"])  
 plt.show()

# 2.数据说明

\* Attrition：员工是否已经离职，1表示已经离职，0表示未离职，这是目标预测值；

\* Age：员工年龄

\* BusinessTravel：商务差旅频率，Non-Travel表示不出差，Travel\_Rarely表示不经常出差，Travel\_Frequently表示经常出差；

\* Department：员工所在部门，Sales表示销售部，Research & Development表示研发部，Human Resources表示人力资源部；

\* DistanceFromHome：公司跟家庭住址的距离，从1到29，1表示最近，29表示最远；

\* Education：员工的教育程度，从1到5，5表示教育程度最高；

\* EducationField：员工所学习的专业领域，Life Sciences表示生命科学，Medical表示医疗，Marketing表示市场营销，Technical Degree表示技术学位，Human Resources表示人力资源，Other表示其他；

~~\* EmployeeNumber：员工号码；~~

\* EnvironmentSatisfaction：员工对于工作环境的满意程度，从1到4，1的满意程度最低，4的满意程度最高；

\* Gender：员工性别，Male表示男性，Female表示女性；

\* JobInvolvement：员工工作投入度，从1到4，1为投入度最低，4为投入度最高；

\* JobLevel：职业级别，从1到5，1为最低级别，5为最高级别；

\* JobRole：工作角色：Sales Executive是销售主管，Research Scientist是科学研究员，Laboratory Technician实验室技术员，Manufacturing Director是制造总监，Healthcare Representative是医疗代表，Manager是经理，Sales Representative是销售代表，Research Director是研究总监，Human Resources是人力资源；

\* JobSatisfaction：工作满意度，从1到4，1代表满意程度最低，4代表满意程度最高；

\* MaritalStatus：员工婚姻状况，Single代表单身，Married代表已婚，Divorced代表离婚；

\* MonthlyIncome：员工月收入，范围在1009到19999之间；

\* NumCompaniesWorked：员工曾经工作过的公司数；

\* Over18：年龄是否超过18岁；

\* OverTime：是否加班，Yes表示加班，No表示不加班；

\* PercentSalaryHike：工资提高的百分比；

\* PerformanceRating：绩效评估；

\* RelationshipSatisfaction：关系满意度，从1到4，1表示满意度最低，4表示满意度最高；

\* StandardHours：标准工时；

\* StockOptionLevel：股票期权水平；

\* TotalWorkingYears：总工龄；

\* TrainingTimesLastYear：上一年的培训时长，从0到6，0表示没有培训，6表示培训时间最长；

\* WorkLifeBalance：工作与生活平衡程度，从1到4，1表示平衡程度最低，4表示平衡程度最高；

\* YearsAtCompany：在目前公司工作年数；

\* YearsInCurrentRole：在目前工作职责的工作年数

\* YearsSinceLastPromotion：距离上次升职时长

\* YearsWithCurrManager：跟目前的管理者共事年数；

# 3.对特征数据进行分类:

为什么进行特征分类？

一般在机器学习中对数值型数据、类别型数据、**有序性数据处理方式**和方法是不同：

数值型数据：进行标准化和归一化处理，均值插补(处理缺失值)

类别型数据：label-encode和one-hot encode

有序数据：数值型+顺序

将上述数据进行分类：

类别标签：Attrition 0 1-1表示已经离职，0表示未离职，这是目标预测值

数值型数据：Age、MonthlyIncome、NumCompaniesWorked、PercentSalaryHike、StandardHours、TotalWorkingYears、YearsAtCompany、YearsInCurrentRole、YearsSinceLastPromotion、YearsWithCurrManager

类别型数据：BusinessTravel、Department、EducationField、Gender、JobRole、MaritalStatus、Over18、OverTime

有序数据：DistanceFromHome、Education、EnvironmentSatisfaction、JobInvolvement、JobLevel、JobSatisfaction、RelationshipSatisfaction、StockOptionLevel、TrainingTimesLastYear、WorkLifeBalance、

进一步查看数据类型：

# MaritalStatus 1100 non-null object

# BusinessTravel 1100 non-null object

# Department 1100 non-null object

# EducationField 1100 non-null object

# Gender 1100 non-null object

# JobRole 1100 non-null object

# Over18 1100 non-null object

# OverTime 1100 non-null object

# 处理过程

## 4.1数据准备

import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns #pip install seaborn安装  
talent\_data=pd.read\_csv("./train.csv")  
def show\_edu\_attri():  
 # 分析离职和受教育程度的分析  
 sns.countplot(x="Attrition", hue="Education", data=talent\_data)  
 plt.show()  
  
def show\_attri\_var():  
 plt.subplot(221)  
 # 1.离职和年龄的关系  
 sns.boxplot(x="Attrition", y="Age", data=talent\_data)  
 plt.subplot(222)  
 # 2.离职和家庭和距离之间的关系  
 sns.boxplot(x="Attrition", y="DistanceFromHome", data=talent\_data)  
 plt.subplot(223)  
 # 3.离职和月收入的关系  
 sns.boxplot(x="Attrition", y="MonthlyIncome", data=talent\_data)  
 plt.subplot(224)  
 # 4.离职和曾经工作公司的关系  
 sns.boxplot(x="Attrition", y="NumCompaniesWorked", data=talent\_data)  
 plt.show()  
  
  
def show\_attri\_gender\_marry():  
 plt.subplot(211)  
 # 离职和婚姻状况分析  
 sns.countplot(x="Attrition", hue="MaritalStatus", data=talent\_data)  
 plt.subplot(212)  
 # 离职和性别关系  
 sns.countplot(x="Attrition", hue="Gender", data=talent\_data)  
 plt.show()  
  
  
def show\_pairplot():  
 sns.pairplot(data=talent\_data, hue="Attrition",  
 vars=["Age", "DistanceFromHome", "JobLevel", "PerformanceRating", "WorkLifeBalance"])  
 plt.show()  
  
if \_\_name\_\_=="\_\_main\_\_":  
 # show\_edu\_attri()  
 # show\_attri\_var()  
 # show\_attri\_gender\_marry()  
 show\_pairplot()  
 pass

4.2处理数据

import pandas as pd  
  
talent\_data = pd.read\_csv("./train.csv")  
# 数值型数据  
num\_cols = ["Age", "MonthlyIncome", "NumCompaniesWorked", "PercentSalaryHike", "PerformanceRating",  
 "StandardHours", "TotalWorkingYears", "YearsAtCompany",  
 "YearsInCurrentRole", "YearsSinceLastPromotion"]  
# \* Age：员工年龄  
# \* MonthlyIncome：员工月收入  
# \* NumCompaniesWorked：员工曾经工作过的公司数；  
# \* PercentSalaryHike：工资提高的百分比；  
# \* PerformanceRating：绩效评估；  
# \* StandardHours：标准工时；  
# \* TotalWorkingYears：总工龄；  
# \* YearsAtCompany：在目前公司工作年数；  
# \* YearsInCurrentRole：在目前工作职责的工作年数  
# \* YearsSinceLastPromotion：距离上次升职时长  
**# 类别型数据**  
cat\_cols=["BusinessTravel","Department","EducationField",  
 "Gender","JobRole","MaritalStatus","Over18","OverTime"]  
# \* BusinessTravel：商务差旅频率  
# \* Department：员工所在部门  
# \* EducationField：员工所学习的专业领域  
# \* Gender：员工性别  
# \* JobRole：工作角色  
# \* MaritalStatus：员工婚姻状况  
# \* Over18：年龄是否超过18岁；  
# \* OverTime：是否加班，Yes表示加班，No表示不加班；  
#  
**# 有序型数据**  
ord\_cols=["DistanceFromHome","Education","EnvironmentSatisfaction","JobInvolvement",  
 "JobLevel","JobSatisfaction","RelationshipSatisfaction","StockOptionLevel",  
 "TrainingTimesLastYear","WorkLifeBalance"]  
# \* DistanceFromHome：公司跟家庭住址的距离  
# \* Education：员工的教育程度  
# \* EnvironmentSatisfaction：员工对于工作环境的满意程度  
# \* JobInvolvement：员工工作投入度  
# \* JobLevel：职业级别  
# \* JobSatisfaction：工作满意度  
# \* RelationshipSatisfaction：关系满意度  
# \* StockOptionLevel：股票期权水平；  
# \* TrainingTimesLastYear：上一年的培训时长  
# \* WorkLifeBalance：工作与生活平衡程度，

## 4.3特征工程

import pandas as pd  
  
talent\_data = pd.read\_csv("./train.csv")  
#1.不同类型数据的整理  
  
# 数值型数据  
num\_cols = ["Age", "MonthlyIncome", "NumCompaniesWorked", "PercentSalaryHike", "PerformanceRating",  
 "StandardHours", "TotalWorkingYears", "YearsAtCompany",  
 "YearsInCurrentRole", "YearsSinceLastPromotion"]  
# 类别型数据  
cat\_cols=["Gender","MaritalStatus","OverTime"]  
# 有序型数据  
ord\_cols=["DistanceFromHome","Education","EnvironmentSatisfaction","JobInvolvement",  
 "JobLevel","JobSatisfaction","RelationshipSatisfaction","StockOptionLevel",  
 "TrainingTimesLastYear","WorkLifeBalance"]  
#类别标签  
target\_col=["Attrition"]  
total\_data=num\_cols+ord\_cols+cat\_cols  
#将所有的特征数据和类别标签进行整合  
use\_data=talent\_data[total\_data+target\_col]  
  
#2.正负样本的比例  
#正负样本的不均衡问题  
neg\_data = use\_data[use\_data["Attrition"] == 0] #未离职  
pos\_data = use\_data[use\_data["Attrition"] == 1] #离职  
print("正负样本比例：", len(pos\_data)/len(neg\_data))  
print("离职:",len(pos\_data))  
print("未离职:",len(neg\_data))  
  
#3.数据集的切分  
# X=talent\_data[total\_data]  
# y=talent\_data["Attrition"]  
# from sklearn.cross\_validation import train\_test\_split  
# X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=22)  
print(len(neg\_data)\*0.8)  
train\_neg\_data=neg\_data.iloc[:int(len(neg\_data)\*0.8)].copy()  
test\_neg\_data=neg\_data.iloc[int(len(neg\_data)\*0.8):].copy()  
print("train\_neg\_data:",len(train\_neg\_data))  
print("test\_neg\_data:",len(test\_neg\_data))  
print(len(pos\_data)\*0.8)  
train\_pos\_data = pos\_data.iloc[:int(len(pos\_data) \* 0.8)].copy()  
test\_pos\_data = pos\_data.iloc[int(len(pos\_data) \* 0.8):].copy()  
print("train\_pos\_data:",len(train\_pos\_data))  
print("test\_pos\_data:",len(test\_pos\_data))  
#合并  
train\_data=pd.concat([train\_neg\_data,train\_pos\_data])  
test\_data=pd.concat([test\_neg\_data,test\_neg\_data])  
print("正负样本比例：",len(pos\_data)/len(neg\_data))  
print("训练集的个数：",len(train\_data))  
print("训练集中正负样本比例",len(train\_pos\_data)/len(train\_neg\_data))  
print("测试集中正负样本比例",len(test\_pos\_data)/len(test\_neg\_data))  
  
**#4.类别型特征的处理**#onehotencoder 独热编码---必须要求数据是整数类型  
#labelencoder 标签编码---0-(claass-1)  
from sklearn.preprocessing import LabelEncoder  
# "Gender","MaritalStatus","OverTime"  
gender\_label\_enc=LabelEncoder()  
train\_data["Gender\_enc"]=gender\_label\_enc.fit\_transform(train\_data["Gender"])  
marital\_label\_enc=LabelEncoder()  
train\_data["MaritalStatus\_enc"]=marital\_label\_enc.fit\_transform(train\_data["MaritalStatus"])  
OT\_label\_enc=LabelEncoder()  
train\_data["OT\_enc"]=OT\_label\_enc.fit\_transform(train\_data["OverTime"])  
print("=="\*100)  
print(train\_data.groupby("Gender\_enc").size())  
print(train\_data.groupby("MaritalStatus\_enc").size())  
print(train\_data.groupby("OT\_enc").size())  
print("=="\*100)  
#独热编码  
from sklearn.preprocessing import OneHotEncoder  
ohe\_enc=OneHotEncoder()  
train\_cat\_feats=ohe\_enc.fit\_transform(train\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]).toarray()  
print(type(train\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]))  
print(type(train\_cat\_feats))  
print(train\_cat\_feats[:5,:])  
# 测试集部分  
test\_data["Gender\_enc"]=gender\_label\_enc.transform(test\_data["Gender"])  
test\_data["MaritalStatus\_enc"]=marital\_label\_enc.transform(test\_data["MaritalStatus"])  
test\_data["OT\_enc"]=OT\_label\_enc.transform(test\_data["OverTime"])  
test\_cat\_feats=ohe\_enc.fit\_transform(test\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]).toarray()  
  
#整合所有的特征  
print(type(train\_data[num\_cols]))  
print(type(train\_data[num\_cols].values))  
train\_num\_feats=train\_data[num\_cols].values  
train\_col\_feats=train\_data[ord\_cols].values  
import numpy as np  
train\_feats=np.hstack([train\_num\_feats,train\_col\_feats,train\_cat\_feats])  
train\_target=train\_data[target\_col].values  
print(len(train\_feats))  
print(len(train\_target))  
# 879  
# 879  
#测试数据  
test\_num\_feats=test\_data[num\_cols].values  
test\_ord\_feats=test\_data[ord\_cols].values  
test\_feats=np.hstack([test\_num\_feats,test\_ord\_feats,test\_cat\_feats])  
test\_target=test\_data[target\_col].values  
print(len(test\_feats))  
print(len(test\_target))  
# 370  
# 370

**容易出错的地方**

**ohe=OneHotEncoder()  
train\_cat\_feat\_data=ohe.fit\_transform(train\_data[["Gender\_enc","MaritalStatus\_enc","OverTime\_enc"]]).toarray()  
test\_cat\_feat\_data=ohe.transform(test\_data[["Gender\_enc","MaritalStatus\_enc","OverTime\_enc"]]).toarray()**

**# <class 'scipy.sparse.csr.csr\_matrix'>  
#通过toarray()方法转化为ndarray  
# <class 'numpy.ndarray'>**

## 4.4字典处理

import pandas as pd  
  
talent\_data = pd.read\_csv("./train.csv")  
#1.不同类型数据的整理  
  
# 数值型数据  
num\_cols = ["Age", "MonthlyIncome", "NumCompaniesWorked", "PercentSalaryHike", "PerformanceRating",  
 "StandardHours", "TotalWorkingYears", "YearsAtCompany",  
 "YearsInCurrentRole", "YearsSinceLastPromotion"]  
# 类别型数据  
cat\_cols=["Gender","MaritalStatus","OverTime"]  
# 有序型数据  
ord\_cols=["DistanceFromHome","Education","EnvironmentSatisfaction","JobInvolvement",  
 "JobLevel","JobSatisfaction","RelationshipSatisfaction","StockOptionLevel",  
 "TrainingTimesLastYear","WorkLifeBalance"]  
#类别标签  
target\_col=["Attrition"]  
total\_data=num\_cols+ord\_cols+cat\_cols  
#将所有的特征数据和类别标签进行整合  
use\_data=talent\_data[total\_data+target\_col]  
  
#2.正负样本的比例  
#正负样本的不均衡问题  
neg\_data = use\_data[use\_data["Attrition"] == 0] #未离职  
pos\_data = use\_data[use\_data["Attrition"] == 1] #离职  
print("正负样本比例：", len(pos\_data)/len(neg\_data))  
print("离职:",len(pos\_data))  
print("未离职:",len(neg\_data))  
  
#3.数据集的切分  
# X=talent\_data[total\_data]  
# y=talent\_data["Attrition"]  
# from sklearn.cross\_validation import train\_test\_split  
# X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=22)  
print(len(neg\_data)\*0.8)  
train\_neg\_data=neg\_data.iloc[:int(len(neg\_data)\*0.8)].copy()  
test\_neg\_data=neg\_data.iloc[int(len(neg\_data)\*0.8):].copy()  
print("train\_neg\_data:",len(train\_neg\_data))  
print("test\_neg\_data:",len(test\_neg\_data))  
print(len(pos\_data)\*0.8)  
train\_pos\_data = pos\_data.iloc[:int(len(pos\_data) \* 0.8)].copy()  
test\_pos\_data = pos\_data.iloc[int(len(pos\_data) \* 0.8):].copy()  
print("train\_pos\_data:",len(train\_pos\_data))  
print("test\_pos\_data:",len(test\_pos\_data))  
#合并  
train\_data=pd.concat([train\_neg\_data,train\_pos\_data])  
test\_data=pd.concat([test\_neg\_data,test\_neg\_data])  
print("正负样本比例：",len(pos\_data)/len(neg\_data))  
print("训练集的个数：",len(train\_data))  
print("训练集中正负样本比例",len(train\_pos\_data)/len(train\_neg\_data))  
print("测试集中正负样本比例",len(test\_pos\_data)/len(test\_neg\_data))  
  
#4.类别型特征的处理  
#onehotencoder 独热编码---必须要求数据是整数类型  
#labelencoder 标签编码---0-(claass-1)  
from sklearn.feature\_extraction import DictVectorizer  
dtc=DictVectorizer(sparse=False)  
# arr1=dtc.fit\_transform([{'foo': "1st", 'bar': "apple"}, {'foo': "2nd", 'bar': "pear"},{'foo': "3rd", 'bar': "apple"}])  
# print(arr1)  
# print(dtc.feature\_names\_)  
train\_feats\_data=dtc.fit\_transform(train\_data[["Gender","MaritalStatus","OverTime"]].to\_dict(orient="records"))  
# print(train\_feats\_data)  
# print(dtc.feature\_names\_)  
test\_feats\_data=dtc.transform(test\_data[["Gender","MaritalStatus","OverTime"]].to\_dict(orient="records"))  
print(test\_feats\_data)  
print(dtc.feature\_names\_)

## 4.5建模

import pandas as pd  
  
talent\_data = pd.read\_csv("./train.csv")  
#1.不同类型数据的整理  
  
# 数值型数据  
num\_cols = ["Age", "MonthlyIncome", "NumCompaniesWorked", "PercentSalaryHike", "PerformanceRating",  
 "StandardHours", "TotalWorkingYears", "YearsAtCompany",  
 "YearsInCurrentRole", "YearsSinceLastPromotion"]  
# 类别型数据  
cat\_cols=["Gender","MaritalStatus","OverTime"]  
# 有序型数据  
ord\_cols=["DistanceFromHome","Education","EnvironmentSatisfaction","JobInvolvement",  
 "JobLevel","JobSatisfaction","RelationshipSatisfaction","StockOptionLevel",  
 "TrainingTimesLastYear","WorkLifeBalance"]  
#类别标签  
target\_col=["Attrition"]  
total\_data=num\_cols+ord\_cols+cat\_cols  
#将所有的特征数据和类别标签进行整合  
use\_data=talent\_data[total\_data+target\_col]  
  
#2.正负样本的比例  
#正负样本的不均衡问题  
neg\_data = use\_data[use\_data["Attrition"] == 0] #未离职  
pos\_data = use\_data[use\_data["Attrition"] == 1] #离职  
print("正负样本比例：", len(pos\_data)/len(neg\_data))  
print("离职:",len(pos\_data))  
print("未离职:",len(neg\_data))  
  
#3.数据集的切分  
# X=talent\_data[total\_data]  
# y=talent\_data["Attrition"]  
# from sklearn.cross\_validation import train\_test\_split  
# X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=22)  
print(len(neg\_data)\*0.8)  
train\_neg\_data=neg\_data.iloc[:int(len(neg\_data)\*0.8)].copy()  
test\_neg\_data=neg\_data.iloc[int(len(neg\_data)\*0.8):].copy()  
print("train\_neg\_data:",len(train\_neg\_data))  
print("test\_neg\_data:",len(test\_neg\_data))  
print(len(pos\_data)\*0.8)  
train\_pos\_data = pos\_data.iloc[:int(len(pos\_data) \* 0.8)].copy()  
test\_pos\_data = pos\_data.iloc[int(len(pos\_data) \* 0.8):].copy()  
print("train\_pos\_data:",len(train\_pos\_data))  
print("test\_pos\_data:",len(test\_pos\_data))  
#合并  
train\_data=pd.concat([train\_neg\_data,train\_pos\_data])  
test\_data=pd.concat([test\_neg\_data,test\_neg\_data])  
print("正负样本比例：",len(pos\_data)/len(neg\_data))  
print("训练集的个数：",len(train\_data))  
print("训练集中正负样本比例",len(train\_pos\_data)/len(train\_neg\_data))  
print("测试集中正负样本比例",len(test\_pos\_data)/len(test\_neg\_data))  
  
#4.类别型特征的处理  
#onehotencoder 独热编码---必须要求数据是整数类型  
#labelencoder 标签编码---0-(claass-1)  
from sklearn.preprocessing import LabelEncoder  
# "Gender","MaritalStatus","OverTime"  
gender\_label\_enc=LabelEncoder()  
train\_data["Gender\_enc"]=gender\_label\_enc.fit\_transform(train\_data["Gender"])  
marital\_label\_enc=LabelEncoder()  
train\_data["MaritalStatus\_enc"]=marital\_label\_enc.fit\_transform(train\_data["MaritalStatus"])  
OT\_label\_enc=LabelEncoder()  
train\_data["OT\_enc"]=OT\_label\_enc.fit\_transform(train\_data["OverTime"])  
print("=="\*100)  
print(train\_data.groupby("Gender\_enc").size())  
print(train\_data.groupby("MaritalStatus\_enc").size())  
print(train\_data.groupby("OT\_enc").size())  
print("=="\*100)  
#独热编码  
from sklearn.preprocessing import OneHotEncoder  
ohe\_enc=OneHotEncoder()  
train\_cat\_feats=ohe\_enc.fit\_transform(train\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]).toarray()  
print(type(train\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]))  
print(type(train\_cat\_feats))  
print(train\_cat\_feats[:5,:])  
# 测试集部分  
test\_data["Gender\_enc"]=gender\_label\_enc.transform(test\_data["Gender"])  
test\_data["MaritalStatus\_enc"]=marital\_label\_enc.transform(test\_data["MaritalStatus"])  
test\_data["OT\_enc"]=OT\_label\_enc.transform(test\_data["OverTime"])  
test\_cat\_feats=ohe\_enc.fit\_transform(test\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]).toarray()  
  
#整合所有的特征  
print(type(train\_data[num\_cols]))  
print(type(train\_data[num\_cols].values))  
train\_num\_feats=train\_data[num\_cols].values  
train\_col\_feats=train\_data[ord\_cols].values  
import numpy as np  
train\_feats=np.hstack([train\_num\_feats,train\_col\_feats,train\_cat\_feats])  
train\_target=train\_data[target\_col].values  
print(len(train\_feats))  
print(len(train\_target))  
# 879  
# 879  
#测试数据  
test\_num\_feats=test\_data[num\_cols].values  
test\_ord\_feats=test\_data[ord\_cols].values  
test\_feats=np.hstack([test\_num\_feats,test\_ord\_feats,test\_cat\_feats])  
test\_target=test\_data[target\_col].values  
print(len(test\_feats))  
print(len(test\_target))  
# 370  
# 370  
  
#使用随机森林建立模型  
from sklearn.ensemble import RandomForestClassifier  
rf=RandomForestClassifier(n\_estimators=100,criterion="gini")  
rf.fit(train\_feats,train\_target)  
#预测  
y\_pred=rf.predict(test\_feats)  
print("model in trainset score is:",rf.score(train\_feats,train\_target))  
print("model in testsize score is:",rf.score(test\_feats,test\_target))  
# model in trainset score is: 1.0  
# model in testsize score is: 0.972972972972973  
#混淆矩阵  
from sklearn.metrics import confusion\_matrix,classification\_report  
print("混淆矩阵：\n",confusion\_matrix(test\_target,y\_pred))  
print(classification\_report(test\_target,y\_pred))  
#使用逻辑斯特回归模型--分类模型  
from sklearn.linear\_model import LogisticRegression  
lr=LogisticRegression(penalty="l2")  
lr.fit(train\_feats,train\_target)  
#预测  
y\_pred1=lr.predict(test\_feats)  
print("model in trainset score is:",lr.score(train\_feats,train\_target))  
print("model in testsize score is:",lr.score(test\_feats,test\_target))  
# model in trainset score is: 1.0  
# model in testsize score is: 0.972972972972973  
#混淆矩阵  
from sklearn.metrics import confusion\_matrix,classification\_report  
print("混淆矩阵：\n",confusion\_matrix(test\_target,y\_pred1))  
print(classification\_report(test\_target,y\_pred1))

## 4.6SMOTE处理

import pandas as pd  
  
talent\_data = pd.read\_csv("./train.csv")  
#1.不同类型数据的整理  
  
# 数值型数据  
num\_cols = ["Age", "MonthlyIncome", "NumCompaniesWorked", "PercentSalaryHike", "PerformanceRating",  
 "StandardHours", "TotalWorkingYears", "YearsAtCompany",  
 "YearsInCurrentRole", "YearsSinceLastPromotion"]  
# 类别型数据  
cat\_cols=["Gender","MaritalStatus","OverTime"]  
# 有序型数据  
ord\_cols=["DistanceFromHome","Education","EnvironmentSatisfaction","JobInvolvement",  
 "JobLevel","JobSatisfaction","RelationshipSatisfaction","StockOptionLevel",  
 "TrainingTimesLastYear","WorkLifeBalance"]  
#类别标签  
target\_col=["Attrition"]  
total\_data=num\_cols+ord\_cols+cat\_cols  
#将所有的特征数据和类别标签进行整合  
use\_data=talent\_data[total\_data+target\_col]  
  
#2.正负样本的比例  
#正负样本的不均衡问题  
neg\_data = use\_data[use\_data["Attrition"] == 0] #未离职  
pos\_data = use\_data[use\_data["Attrition"] == 1] #离职  
print("正负样本比例：", len(pos\_data)/len(neg\_data))  
print("离职:",len(pos\_data))  
print("未离职:",len(neg\_data))  
  
#3.数据集的切分  
# X=talent\_data[total\_data]  
# y=talent\_data["Attrition"]  
# from sklearn.cross\_validation import train\_test\_split  
# X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=22)  
print(len(neg\_data)\*0.8)  
train\_neg\_data=neg\_data.iloc[:int(len(neg\_data)\*0.8)].copy()  
test\_neg\_data=neg\_data.iloc[int(len(neg\_data)\*0.8):].copy()  
print("train\_neg\_data:",len(train\_neg\_data))  
print("test\_neg\_data:",len(test\_neg\_data))  
print(len(pos\_data)\*0.8)  
train\_pos\_data = pos\_data.iloc[:int(len(pos\_data) \* 0.8)].copy()  
test\_pos\_data = pos\_data.iloc[int(len(pos\_data) \* 0.8):].copy()  
print("train\_pos\_data:",len(train\_pos\_data))  
print("test\_pos\_data:",len(test\_pos\_data))  
#合并  
train\_data=pd.concat([train\_neg\_data,train\_pos\_data])  
test\_data=pd.concat([test\_neg\_data,test\_neg\_data])  
print("正负样本比例：",len(pos\_data)/len(neg\_data))  
print("训练集的个数：",len(train\_data))  
print("训练集中正负样本比例",len(train\_pos\_data)/len(train\_neg\_data))  
print("测试集中正负样本比例",len(test\_pos\_data)/len(test\_neg\_data))  
  
#4.类别型特征的处理  
#onehotencoder 独热编码---必须要求数据是整数类型  
#labelencoder 标签编码---0-(claass-1)  
from sklearn.preprocessing import LabelEncoder  
# "Gender","MaritalStatus","OverTime"  
gender\_label\_enc=LabelEncoder()  
train\_data["Gender\_enc"]=gender\_label\_enc.fit\_transform(train\_data["Gender"])  
marital\_label\_enc=LabelEncoder()  
train\_data["MaritalStatus\_enc"]=marital\_label\_enc.fit\_transform(train\_data["MaritalStatus"])  
OT\_label\_enc=LabelEncoder()  
train\_data["OT\_enc"]=OT\_label\_enc.fit\_transform(train\_data["OverTime"])  
print("=="\*100)  
print(train\_data.groupby("Gender\_enc").size())  
print(train\_data.groupby("MaritalStatus\_enc").size())  
print(train\_data.groupby("OT\_enc").size())  
print("=="\*100)  
#独热编码  
from sklearn.preprocessing import OneHotEncoder  
ohe\_enc=OneHotEncoder()  
train\_cat\_feats=ohe\_enc.fit\_transform(train\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]).toarray()  
print(type(train\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]))  
print(type(train\_cat\_feats))  
print(train\_cat\_feats[:5,:])  
# 测试集部分  
test\_data["Gender\_enc"]=gender\_label\_enc.transform(test\_data["Gender"])  
test\_data["MaritalStatus\_enc"]=marital\_label\_enc.transform(test\_data["MaritalStatus"])  
test\_data["OT\_enc"]=OT\_label\_enc.transform(test\_data["OverTime"])  
test\_cat\_feats=ohe\_enc.fit\_transform(test\_data[["Gender\_enc","MaritalStatus\_enc","OT\_enc"]]).toarray()  
  
#整合所有的特征  
print(type(train\_data[num\_cols]))  
print(type(train\_data[num\_cols].values))  
train\_num\_feats=train\_data[num\_cols].values  
train\_col\_feats=train\_data[ord\_cols].values  
import numpy as np  
train\_feats=np.hstack([train\_num\_feats,train\_col\_feats,train\_cat\_feats])  
train\_target=train\_data[target\_col].values  
print(len(train\_feats))  
print(len(train\_target))  
# 879  
# 879  
#测试数据  
test\_num\_feats=test\_data[num\_cols].values  
test\_ord\_feats=test\_data[ord\_cols].values  
test\_feats=np.hstack([test\_num\_feats,test\_ord\_feats,test\_cat\_feats])  
test\_target=test\_data[target\_col].values  
print(len(test\_feats))  
print(len(test\_target))  
# 370  
# 370  
  
  
#仅需要对训练数据进行采样  
from imblearn.over\_sampling import SMOTE  
smoto=SMOTE()  
train\_feats,train\_target=smoto.fit\_sample(train\_feats,train\_target)  
from collections import Counter  
# print(sorted(Counter(train\_target).items()))  
  
#  
#使用随机森林建立模型  
from sklearn.ensemble import RandomForestClassifier  
rf=RandomForestClassifier(n\_estimators=100,criterion="gini")  
rf.fit(train\_feats,train\_target)  
#预测  
y\_pred=rf.predict(test\_feats)  
print("model in trainset score is:",rf.score(train\_feats,train\_target))  
print("model in testsize score is:",rf.score(test\_feats,test\_target))  
# model in trainset score is: 1.0  
# model in testsize score is: 0.972972972972973  
#混淆矩阵  
from sklearn.metrics import confusion\_matrix,classification\_report  
print("混淆矩阵：\n",confusion\_matrix(test\_target,y\_pred))  
print(classification\_report(test\_target,y\_pred))  
#使用逻辑斯特回归模型--分类模型  
from sklearn.linear\_model import LogisticRegression  
lr=LogisticRegression(penalty="l2")  
lr.fit(train\_feats,train\_target)  
#预测  
y\_pred1=lr.predict(test\_feats)  
print("model in trainset score is:",lr.score(train\_feats,train\_target))  
print("model in testsize score is:",lr.score(test\_feats,test\_target))  
# model in trainset score is: 1.0  
# model in testsize score is: 0.972972972972973  
#混淆矩阵  
from sklearn.metrics import confusion\_matrix,classification\_report  
print("混淆矩阵：\n",confusion\_matrix(test\_target,y\_pred1))  
print(classification\_report(test\_target,y\_pred1))

## 4.7整合

#在python3下面请注意修改print函数  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn import preprocessing  
import os  
# https://pypi.python.org/pypi/imblearn/0.0  
# https://pypi.python.org/pypi/imbalanced-learn/  
# from imblearn.over\_sampling import SMOTE  
from sklearn import metrics  
  
# 指定数据集路径  
datafile\_path = os.path.join('.', 'train.csv')  
  
DO\_OVERSAMPLE = False  
  
def FeatureChuli():  
 """  
 主建模函数  
 """  
 # 加载数据  
 sample\_data = pd.read\_csv(datafile\_path)  
 # ==== 预览数据 ====  
 print('数据预览：')  
 print(sample\_data.head())  
 print(sample\_data.info())  
 print(sample\_data.describe())  
  
 # 正负样本的比例  
 pos\_data = sample\_data[sample\_data['Attrition'] == 1]  
 neg\_data = sample\_data[sample\_data['Attrition'] == 0]  
 print('正样本记录数：{}，所占比例：{}'.format(len(pos\_data), len(pos\_data) / len(sample\_data)))  
 print('负样本记录数：{}，所占比例：{}'.format(len(neg\_data), len(neg\_data) / len(sample\_data)))  
  
 # ==== 变量关系可视化 ====  
 # visualize\_data(sample\_data)  
  
 # ==== 离职预测 ====  
 # 数值型数据  
 num\_cols = ['Age', 'MonthlyIncome', 'NumCompaniesWorked', 'PercentSalaryHike', 'TotalWorkingYears',  
 'TrainingTimesLastYear',  
 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']  
 # 类别型数据  
 # 所有类别型数据  
 # cat\_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus',  
 # 'Over18', 'OverTime']  
 # 本案例只选取3个作为例子  
 cat\_cols = ['Gender', 'MaritalStatus', 'OverTime']  
 # 有序类别数据  
 ord\_cols = ['DistanceFromHome', 'Education', 'EnvironmentSatisfaction', 'JobInvolvement', 'JobLevel',  
 'JobSatisfaction',  
 'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'WorkLifeBalance']  
  
 # 目标列  
 target\_col = ['Attrition']  
  
 # 所有特征列  
 total\_cols = num\_cols + cat\_cols + ord\_cols  
  
 used\_data = sample\_data[total\_cols + target\_col]  
  
 print('使用{}列数据作为特征'.format(len(total\_cols)))  
  
 # 分割训练集，测试集，80%作为训练集，20%作为测试集  
 # 保证训练集和测试集中的正负样本的比例一样  
 # 正负样本的比例  
  
 pos\_data = used\_data[used\_data['Attrition'] == 1].reindex()  
 train\_pos\_data = pos\_data.iloc[:int(len(pos\_data) \* 0.8)].copy()  
 test\_pos\_data = pos\_data.iloc[int(len(pos\_data) \* 0.8):].copy()  
  
 neg\_data = used\_data[used\_data['Attrition'] == 0].reindex()  
 train\_neg\_data = neg\_data.iloc[:int(len(neg\_data) \* 0.8)].copy()  
 test\_neg\_data = neg\_data.iloc[int(len(neg\_data) \* 0.8):].copy()  
  
 train\_data = pd.concat([train\_pos\_data, train\_neg\_data])  
 test\_data = pd.concat([test\_pos\_data, test\_neg\_data])  
  
 print('训练集数据个数', len(train\_data))  
 print('正负样本比例', len(train\_pos\_data) / len(train\_neg\_data))  
 train\_data.head()  
  
 print('测试集数据个数', len(test\_data))  
 print('正负样本比例', len(test\_pos\_data) / len(test\_neg\_data))  
 test\_data.head()  
  
 # ==== 特征工程 ====  
 # 对类别型数据进行“独热编码” One-Hot Encoding  
  
 # 先进行Label Encoding  
 # Gender数据  
 gender\_label\_enc = preprocessing.LabelEncoder()  
 train\_data['Gender\_Label'] = gender\_label\_enc.fit\_transform(train\_data['Gender'])  
  
 # MaritalStatus数据  
 marital\_label\_enc = preprocessing.LabelEncoder()  
 train\_data['Marital\_Label'] = marital\_label\_enc.fit\_transform(train\_data['MaritalStatus'])  
  
 # OverTime数据  
 ot\_label\_enc = preprocessing.LabelEncoder()  
 train\_data['OT\_Label'] = ot\_label\_enc.fit\_transform(train\_data['OverTime'])  
  
 print('Gender数据:')  
 print(train\_data.groupby('Gender\_Label').size())  
  
 print()  
 print('MaritalStatus数据:')  
 print(train\_data.groupby('Marital\_Label').size())  
  
 print()  
 print('OverTime数据:')  
 print(train\_data.groupby('OT\_Label').size())  
  
 # 再进行One-Hot Encoding  
 one\_hot\_enc = preprocessing.OneHotEncoder()  
 train\_cat\_feats = one\_hot\_enc.fit\_transform(train\_data[['Gender\_Label', 'Marital\_Label', 'OT\_Label']]).toarray()  
 # print(train\_cat\_feats[:5, :])  
  
 # 对测试集数据进行相应的编码操作  
 # 重点：先将类别型编码为数字，在根据数字产生one-hot编码  
 # 注意要使用从训练集中得出的encoder  
  
 # 标签编码  
 # Gender数据  
 test\_data['Gender\_Label'] = gender\_label\_enc.transform(test\_data['Gender'])  
  
 # MaritalStatus数据  
 test\_data['Marital\_Label'] = marital\_label\_enc.transform(test\_data['MaritalStatus'])  
  
 # OverTime数据  
 test\_data['OT\_Label'] = ot\_label\_enc.transform(test\_data['OverTime'])  
  
 # 独热编码  
 test\_cat\_feats = one\_hot\_enc.transform(test\_data[['Gender\_Label', 'Marital\_Label', 'OT\_Label']]).toarray()  
  
 # 整合所有特征  
 train\_num\_feats = train\_data[num\_cols].values  
 train\_ord\_feats = train\_data[ord\_cols].values  
 train\_feats = np.hstack((train\_num\_feats, train\_ord\_feats, train\_cat\_feats))  
 train\_targets = train\_data[target\_col].values  
 print("="\*100)  
 print(train\_feats[:5,:])  
 # 整合所有特征  
 test\_num\_feats = test\_data[num\_cols].values  
 test\_ord\_feats = test\_data[ord\_cols].values  
 test\_feats = np.hstack((test\_num\_feats, test\_ord\_feats, test\_cat\_feats))  
 test\_targets = test\_data[target\_col].values  
  
 print('训练数据：', train\_feats.shape)  
 print('测试数据：', test\_feats.shape)  
  
 # if DO\_OVERSAMPLE:  
 # # 处理不平衡数据  
 # # 过采样“少”的样本  
 # # 安装imblearn包：conda install -c glemaitre imbalanced-learn  
 # # 或者：pip install -U imbalanced-learn  
 #  
 # print('重采样前：')  
 # print('正样本个数：', len(train\_targets[train\_targets == 1]))  
 # print('负样本个数：', len(train\_targets[train\_targets == 0]))  
 #  
 # sm = SMOTE(random\_state=0)  
 # train\_resampled\_feats, train\_resampled\_targets = sm.fit\_sample(train\_feats, train\_targets)  
 # print('重采样后：')  
 # print('正样本个数：', len(train\_resampled\_targets[train\_resampled\_targets == 1]))  
 # print('负样本个数：', len(train\_resampled\_targets[train\_resampled\_targets == 0]))  
  
 return test\_feats, test\_neg\_data, test\_pos\_data, test\_targets, train\_feats, train\_targets  
  
  
def ModelCreate(test\_feats, test\_neg\_data, test\_pos\_data, test\_targets, train\_feats, train\_targets):  
 # ==== 数据建模 ====  
 if DO\_OVERSAMPLE:  
 # 如果选择“重采样”请取消以下的注释  
 # 随机森林  
 # rf\_clf = RandomForestClassifier(random\_state=0)  
 # rf\_clf.fit(train\_resampled\_feats, train\_resampled\_targets)  
 #  
 # # 逻辑回归  
 # lr\_clf = LogisticRegression()  
 # lr\_clf.fit(train\_resampled\_feats, train\_resampled\_targets)  
 pass  
 else:  
 # 随机森林  
 rf\_clf = RandomForestClassifier(random\_state=0)  
 rf\_clf.fit(train\_feats, train\_targets)  
  
 # 逻辑回归  
 lr\_clf = LogisticRegression()  
 lr\_clf.fit(train\_feats, train\_targets)  
  
 # ==== 模型验证 ====  
 # 随机森林  
 if DO\_OVERSAMPLE:  
 print('重采样后ReSampling：')  
 print('测试集中正样本数', len(test\_pos\_data))  
 print('测试集中负样本数', len(test\_neg\_data))  
 print('随机森林RndomState：')  
 test\_pred = rf\_clf.predict(test\_feats)  
 print(metrics.confusion\_matrix(test\_targets, test\_pred, labels=[1, 0]))  
 print('准确率Accuracy：', metrics.accuracy\_score(test\_targets, test\_pred))  
 # 逻辑回归  
 print('逻辑回归LogisticRegression：')  
 test\_pred = lr\_clf.predict(test\_feats)  
 print(metrics.confusion\_matrix(test\_targets, test\_pred, labels=[1, 0]))  
 print('准确率Accuracy：', metrics.accuracy\_score(test\_targets, test\_pred))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data =FeatureChuli()  
 ModelCreate(\*data)