[模型标准化](#header-n2)  
 [数据输入输出](#header-n3)  
 [配置数据库登录信息](#header-n5)  
 [数据读取](#header-n16)  
 [数据导入](#header-n25)  
 [数据探索](#header-n34)  
 [实例化](#header-n36)  
 [输出数值型数据的统计信息](#header-n44)  
 [输出非数值型数据的统计信息](#header-n50)  
 [输出变量相关性热力图](#header-n56)  
 [输出各变量的分布图](#header-n61)  
 [输出变量关系图](#header-n64)  
 [数据预处理](#header-n73)  
 [实例化](#header-n75)  
 [数据无量纲化](#header-n83)  
 [空值填充](#header-n89)

唯一值检验  
 [数据编码](#header-n99)  
 [对数转换](#header-n113)  
 [boxcox转换](#header-n114)  
 [johnson转换](#header-n126)  
 [特征选择](#header-n137)  
 [实例化](#header-n139)  
 [基于统计特征选择](#header-n147)  
 [方差筛选](#header-n148)  
 [过滤法](#header-n157)  
 [基于模型特征选择](#header-n174)  
 [包装法](#header-n175)  
 [嵌入法](#header-n187)  
 [模型训练](#header-n198)  
 [实例化](#header-n200)  
 [XGBoost模型](#header-n210)  
 [LightGBM模型](#header-n218)  
 [模型评估](#header-n227)  
 [单模型评估](#header-n229)  
 [实例化](#header-n230)  
 [评估分数](#header-n239)  
 [评估指标可视化](#header-n245)

模型区间结果  
 [多模型评估对比](#header-n251)  
 [实例化](#header-n252)  
 [评估分数对比](#header-n261)  
 [评估指标可视化对比](#header-n267)  
 [模型保存和加载](#header-n273)  
 [实例化](#header-n275)  
 [模型保存](#header-n278)  
 [模型加载](#header-n285)  
 [预测数据](#header-n291)  
 [实例化](#header-n293)  
 [数据预处理](#header-n296)  
 [特征选择](#header-n305)  
 [模型预测](#header-n314)

模型可解释  
 [日志生成](#header-n325)  
 [实例化](#header-n327)  
 [日志输出到控制台和写入日志文件](#header-n336)

# 模型标准化

## 数据输入输出

***data\_io.py***

### 配置数据库登录信息

* 先在config.ini配置文件上将数据库的登录信息配置完成，示例如下：
* [server]  
  user = dbc  
  password = pass  
  host = 192.168.253.231  
  port = 22  
  dbname = server1
* 将配置文件的地址传入data\_io.database类并实例化后可实现数据输入输出
* path：配置文件的地址，参数类型为string
* from data\_io import database  
  path = r'D:\troywu666\business\_stuff\民生对公项目\模型标准化\config.ini'  
  database = Database(path)

### 数据读取

* 输入sql语句和数据库类型可实现数据读取
* sql：所使用的sql查询语句，参数类型为string
* server：数据库类型，可选的有'teradata'和'mysql'两种
* sql = 'SELECT empmo, ename, mgr FROM emp'  
  database.read\_data(sql = sql, server = 'teradata')

### 数据导入

* 在使用Database(path).data*to*sql函数时传入数据和数据库类型可实现数据读取
* data：需要储存到数据库的数据，类型为pandas.DataFrame
* server：数据库类型，可选的有'teradata'和'mysql'两种
* database.data\_to\_sql(data = data, server = 'teradata')

## 数据探索

***data\_exploring.py***

### 实例化

* 传入需要探索的数据进行并实例化
* data：需要探索的数据，类型为pandas.DataFrame
* from data\_exploring imoprt Explore  
  explore = Explore(data)

### 输出数值型数据的统计信息

* 统计数据的'0%', '12.5%', '25%', '37.5%', '50%', '62.5%', '75%', '87.5%', '95%', '99%', '100%'分位数、数据个数、偏度、峰度、平均值、标准差、空值个数、空值比例
* explore\_frame = explore.describe\_num()

### 输出非数值型数据的统计信息

* 统计数据的非空值个数、出现频率最高的数据、出现频率最高的数据、唯一值个数、空值个数、空值比例
* explore\_frame = explore.describe\_obj()

### 输出变量相关性热力图

figsize：设定图像的大小，参数类型为tuple

explore.corr\_and\_plot(figsize = (32, 18))

### 输出各变量的分布图

explore.distplot()

### 输出变量关系图

* 该图展现变量两两之间的关系（线性或非线性，有无较明显的相关关系）
* vars：想要研究的变量，数据类型为list
* hue：使用指定变量为分类变量画图，数据类型是string
* explore.pairplot(vars = df.columns[: -1], hue = 'age')

## 数据预处理

***preprocessing.py***

### 实例化

* 传入需要预处理的数据进行并实例化
* data：需要处理的数据，类型为pandas.DataFrame
* from preprocessing imoprt Transformer  
    
  process = Transformer(data)

### 数据无量纲化

method：无量纲化的策略，有'min\_max'和'standard'两种取值，'min\_max'为归一化，'standard'为标准化

**返回结果**：**已训练好的数据转化器（方便后续储存及对测试集数据进行转化）**和已无量纲化的训练数据

transformer, tranformed\_data = process.scaler(method = 'min\_max')

### 空值填充

* 可通过设定按不同策略进行填充，如均值、中位数、众数和设定值
* strategy：填充策略，可选'mean'，'median'，'most\_frequent'， 'constant'
* fill*value：当strategy选为'constant'时，fill*value需要额外设定，默认值是None
* **返回结果**：**已训练好的数据转化器（方便后续储存及对测试集数据进行转化）**和已填充空值的训练数据
* strategy = 'constant'  
  fill\_value = 9  
  transformer, tranformed\_data = process.fillna(strategy = strategy, \  
   fill\_value = fill\_value)

### 唯一值检验

* 统计特征唯一值占比，如果大于阈值可直接处理掉。
* data:待处理数据
* single\_threshold:检验阈值
* process.find\_single\_feature(data = data, single\_threshold)

### 数据编码

* 可将分类型和连续型数据进行编码
* method：数据编码的方法，有'onehot'、'ordinal'和'binarizer'三种可填方式，参数类型为 string
* 'onehot'使用于**无序**分类变量进行独热编码
* 'ordinal'使用于**有序**分类变量进行编码
* 'binarizer'使用于将**连续型**变量进行二值化
* threshold：当method选为'binarizer'时，threshold为二值化的阈值。默认值为None，参数类型为float
* **返回结果**：**已训练好的数据转化器（方便后续储存及对测试集数据进行转化）**和已编码的训练数据
* method= 'binarizer'  
  threshold = 3  
  transformer, tranformed\_data = process.encoder(method = method, \  
   threshold = threshold)

### 对数转换

#### boxcox转换

* 使特征数据经过boxcox转换，使其转换后的分布为最接近正态分布
* **输入数据必须为正**
* lmbda：需要转换时使用的lambda值，默认值为None，此时所转换得到的数据最接近正态分布

**返回结果**：**已训练好的数据转化器（方便后续储存及对测试集数据进行转化）**和已对数转换的训练数据

lmbda = None  
transformer, tranformed\_data = process.box\_cox(lmbda = lmbda)

#### johnson转换

* 使特征数据经过johnson转换，使其转换后的分布为最接近正态分布
* **输入数据不为0即可**
* standardize：使输出结果进行标准化，参数类型为boolean，默认值为True
* **返回结果**：**已训练好的数据转化器（方便后续储存及对测试集数据进行转化）**和已对数转换的训练数据
* standardize = True  
  transformer, tranformed\_data = process.johnson(standardize = True)

## 特征选择

***feature\_selection.py***

### 实例化

* 传入需要筛选的数据并进行实例化
* fea：需要筛选的特征数据，类型为pandas.DataFrame
* from feature\_selection import Selector  
  select = Selector(fea)

### 基于统计特征选择

#### 方差筛选

* 通过特征本身的方差来对特征进行筛选
* threshold：方差的阈值，表示将舍弃方差小于阈值的特征。默认值为0，参数类型为float
* **返回结果**：**已训练好的特征筛选器（方便后续储存及对测试集数据进行特征筛选）**和已筛选特征的训练数据
* selector, selected\_data = select.variance(threshold = 0.1)

#### 过滤法

* 根据各种统计检验中的分数以及相关性的各项指标来选择特征
* y：训练数据的额label，参数类型为维度是1的numpy.array
* k：选择需要保留的特征个数，参数类型为int
* method：特征过滤的策略，有'chi2\_filter'、'f\_clas\_filter'、'f\_reg\_filter'、'mutual\_clas\_filter'和'mutual\_reg\_filter'五种可填方式，参数类型为string
* 'chi2\_filter'：适用于在分类问题中计算特征和标签的卡方统计量进行筛选
* 'f\_clas\_filter'：适用于在分类问题中计算特征和标签的F统计量进行筛选
* 'f\_reg\_filter'：适用于在回归问题中计算特征和标签的F统计量进行筛选
* 'mutual\_clas\_filter'：适用于在分类问题中计算特征和标签的互信息量进行筛选
* 'mutual\_reg\_filter'：适用于在回归问题中计算特征和标签的互信息量进行筛选
* **返回结果**：**已训练好的特征筛选器（方便后续储存及对测试集数据进行特征筛选）**和已筛选特征的训练数据
* y = data.target  
  k = 100  
  method = 'chi2\_filter'  
  selector, selected\_data = select.filter(y = y, k = k, method = method)

### 基于模型特征选择

#### 包装法

* 特征选择和算法训练同时进行，反复迭代创建模型，并在每次迭代时保留最佳特征或剔除最差特征，下一次迭代时，算法使用上一次建模中保留的特征来构建下一个模型
* estimator：用于特征筛选的算法，**该算法为sklearn中的算法**，具有feature\_importances\_或coef\_属性
* y：训练数据的额label，参数类型为维度是1的numpy.array
* n：选择需要保留的特征个数，参数类型为int
* step：每次迭代中希望移除的特征个数 ，参数类型为int
* **返回结果**：**已训练好的特征筛选器（方便后续储存及对测试集数据进行特征筛选）**和已筛选特征的训练数据
* from sklearn.ensemble import RandomForestClassifier   
    
  y = data.target  
  n= 100  
  step = 20  
  selector, selected\_data = select.wrapper(RandomForestClassifier(), \  
   y = y, n = n, step = step)

#### 嵌入法

* 特征选择和算法训练同时进行，得到各个特征的权值系数，根据权值系数从大到小选择特征
* estimator：用于特征筛选的算法，**该算法为sklearn中的算法**，具有feature\_importances\_或coef\_属性
* y：训练数据的标签，参数类型为维度是1的numpy.array
* threshold：权值系数的阈值，参数类型为float
* **返回结果**：**已训练好的特征筛选器（方便后续储存及对测试集数据进行特征筛选）**和已筛选特征的训练数据
* from sklearn.ensemble import RandomForestClassifier   
    
  y = data.target  
  threshold = 0.5  
  selector, selected\_data = select.embedded(RandomForestClassifier(), \  
   y = y, threshold = threshold)

## 模型训练

***train\_val.py***

### 实例化

* 传入需要训练数据和验证数据占训练数据的比例，并进行实例化
* train：训练数据的特征，参数类型为pandas.Dataframe
* target：训练数据的标签，参数类型为维度是1的numpy.array
* test\_size：验证数据占训练数据的比例，参数类型为float
* from train\_val import Model\_training  
    
  training = Model\_training(train, target, test\_size = 0.3)

### XGBoost模型

num*boost*round：表示boosting的 迭代数量，参数类型为int

early*stopping*rounds：早停参数，参数类型为int

params：包含训练参数的字典

**返回结果**：**已训练好的算法模型（方便后续储存及对测试集数据进行预测）**、验证集标签值和验证集预测值

num\_boost\_round = 1000  
early\_stopping\_rounds = 100  
params = {'booster': 'gbtree',  
 'objective': 'binary:logistic',  
 'mat\_depth': 3,  
 'subsample': 0.7,  
 'colsample\_bytree': 0.7,  
 'eta': 0.1,  
 'lambda': 0.1,  
 'eval\_metric': 'auc',  
 'scale\_pos\_weight': 0.1}  
model\_xgb, y\_true, y\_pred = training.xgb\_model(num\_boost\_round = num\_boost\_round,  
 early\_stopping\_rounds = early\_stopping\_rounds,  
 params = params)

### LightGBM模型

num*boost*round：表示boosting的 迭代数量，参数类型为int

early*stopping*rounds：早停参数，参数类型为int

params：包含训练参数的字典

categorical\_feature：为属于多分类的特征名列表。默认值为'auto'，需要强调特征名时参数类型为list

**返回结果**：**已训练好的算法模型（方便后续储存及对测试集数据进行预测）**、验证集标签值和验证集预测值

num\_boost\_round = 1000  
early\_stopping\_rounds = 100  
params = {'boosting': 'gbdt',  
 'objective': 'binary',  
 'mat\_depth': 3,  
 'feature\_fraction': 0.7,  
 #'is\_unbalance': True,  
 'eta': 0.1,  
 'lambda\_l1': 0.1,  
 'metric': 'auc',  
 'scale\_pos\_weight': 0.1}  
model\_lgb, y\_true, y\_pred = training.lgb\_model(num\_boost\_round, \  
 early\_stopping\_rounds, \  
 params = params)

## 模型评估

***evalution.py***

### 单模型评估

#### 实例化

* 传入验证集数据的标签以及预测值
* y\_true：验证集标签，数据类型为维度是1的numpy.array
* y\_pred：验证集的预测值，数据类型为维度是1的numpy.array
* from evalution import Metrics  
    
  metrics = Metrics(y\_true, y\_pred)

#### 评估分数

* 输出 准确率、召回率、精确率、AUC分数、f1分数
* accuracy\_score, recall\_score, \  
  precision\_score, roc\_auc\_score, f1\_score = metrics.eval\_score()

#### 评估指标可视化

* 输出ROC曲线、混淆矩阵、KS曲线、PR图、提升曲线、累计增益曲线、概率校准曲线
* metrics.eval\_plot()

#### 模型区间结果评估

* 分别输出测试集各区间结果，区间准确率，召回率，提升度，累计正样本，累计准确率，累计召回率，累计提升度。
* metrics.eval\_area\_result()

### 多模型评估对比

#### 实例化

* 传入验证集数据的标签以及预测值
* y\_true：验证集标签，参数类型为维度是1的numpy.array
* y*pred：不同模型下验证集的预测值，数据类型为dictionary，其中dictionary的keys的参数类型为 string，values的参数类型为维度是1的numpy.array， 如`{'xgb': y*pred*xgb, 'lgb': y*pred\_lgb}`

from evalution import Metrics\_comparison  
   
 metrics\_comparison = Metrics\_comparison(y\_true, y\_pred)

#### 评估分数对比

* 输出 准确率、召回率、精确率、AUC分数、f1分数
* compare\_frame = metrics\_comparison.compare\_score()

#### 评估指标可视化对比

* 输出ROC曲线、混淆矩阵、KS曲线、PR图、提升曲线、累计增益曲线、概率校准曲线的对比图
* metrics\_comparison.compare\_plot()

## 模型保存和加载

***model\_io.py***

### 实例化

from model\_io import Model\_pickle  
  
model\_pickle = Model\_pickle()

### 模型保存

model：需要保存的模型

path：模型存放的路径，参数类型为string

model\_name：模型的命名，参数类型为string

model = model\_lgb  
path = r'D:\troywu666\business\_stuff\民生对公项目\模型标准化  
model\_name = 'model\_lgb'  
model\_pickle.dump(model = model, path = path, model\_name = model\_name)

### 模型加载

path：模型存放的路径，参数类型为string

model\_name：已保存模型的命名，参数类型为string

path = r'D:\troywu666\business\_stuff\民生对公项目\模型标准化  
model\_name = 'model\_xgb'  
model\_pickle.load(mpath = path, model\_name = model\_name)

## 预测数据

***predict.py***

### 实例化

from predict import Prediction  
  
predictor = Prediction()

### 数据预处理

* 基于**已训练好的**数据处理器对测试数据进行数据预处理
* transformer：已训练好的数据处理器
* test：测试集数据，参数类型为pandas.DataFrame
* transformed\_data = predictor.data\_preprocess(transformer, test)

### 特征选择

* 基于**已训练好的**特征选择器对测试数据进行特征筛选
* selector：已训练好的特征选择器
* test：测试集数据，参数类型为pandas.DataFrame
* selected\_data = predictor.feature\_select(selector, test)

### 模型预测

* 基于**已训练好的**模型对测试数据进行预测
* model：已训练好的模型
* test：测试集数据，参数类型为pandas.DataFrame
* model\_cate：模型的类别，可选类别有'xgb'和'lgb'
* pred\_leaf：是否输出叶子节点
* y\_pred = predictor.model\_predict(model, test, model\_cate = 'xgb', \  
   pred\_leaf = False)

### 模型可解释

* 可视化重要特征对模型影响以及重要特征
* shap\_values:特征重要性
* data:训练数据
* shap.summary\_plot(shap\_values,data)
* 对单个样本的预测影响程度
* expected\_value:特征对样本的影响程度
* data\_x:待解释样本特征
* data\_y:待解释样本标签
* shap.force\_plot(expected\_value,data\_x,data\_y)

## 日志生成

***printinlog.py***

### 实例化

* 初始化日志生成器，生成器可以同时将日志输出到控制台和写入日志文件
* path：日志存放的位置，参数类型为string
* logfile\_name：日志的命名，参数类型为string
* from print\_in\_log import Save\_log  
    
  path = r'D:/troywu666/business\_stuff/民生对公项目/模型标准化'  
  logfile\_name = 'test'  
  save\_log = Save\_log(path = path, logfile\_name = logfile\_name)

### 日志输出到控制台和写入日志文件

statement：日志内容，参数类型为string

save\_log.logging(statement = 'Predict complete.')