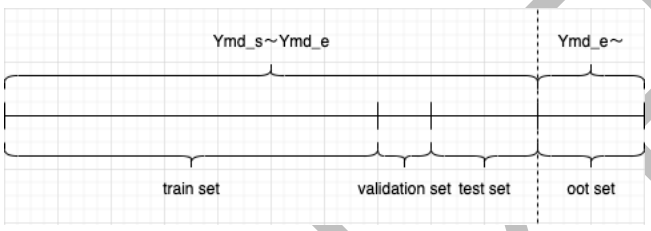
机器学习建模流程要点总结

一、数据集处理

1. 数据集划分

为了构建合理的模型并作出准确评估，我们首先需要对数据集进行合理划分



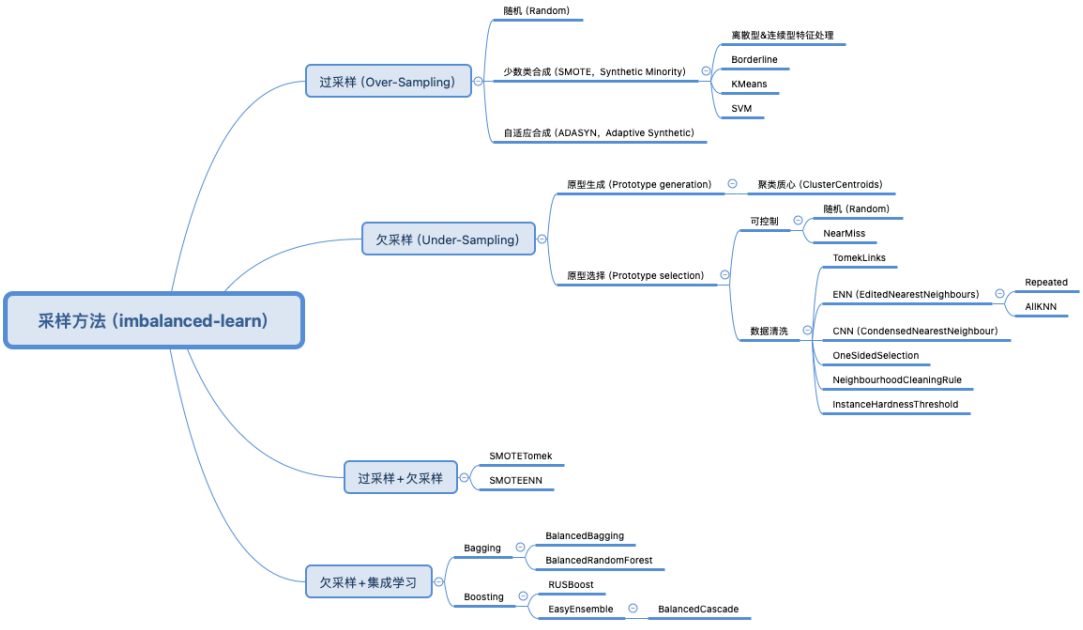
数据集	作用	备注
训练集（train set）	拟合模型	数据集的 60%～80%
验证集（validation set）	模型超参数调整，选择最优模型	1. 数据集的 5%～20% 2. 结合训练集使用交叉验证方法： a) HoldOut：固定比例划分，例：8:2 b) K-Fold：均分 K 组（>=2），循环 K-1 组训练，余下 1 组验证，计算平均效果 c) 留一：循环 N-1 个样本训练，余下单样本测试，计算平均效果 d) … 3. 解决过拟合问题
测试集（test set）	模型性能检验	数据集的 10%～20%
跨时间测试集（oot set）	模型性能检验（稳定性、预期线上效果、模型版本比较）	1. 一般适用于对模型生命周期有一定要求的场景，比如金融、风控等 2. 在时间序列数据中也用作一般测试集

1.1 代码片段

```python
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')

df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)

# HoldOut 交叉验证
df_train_train, df_train_eval = train_test_split(df_train, test_size=0.2, random_state=2024)

# k-Fold 交叉验证
kfold = KFold(n_splits=5, shuffle=True, random_state=2024)
for i, (tdi, vdi) in enumerate(kfold.split(df_train.values)):
    train_train = df_train.values[tdi, :]
    train_eval = df_train.values[vdi, :]
```

2. 数据集采样

实际业务中数据往往是不均衡的
采取适当的采样方法，改善建模样本，更好的拟合模型



2.1 过采样

方法	思想	备注
随机过采样	对少数类样本进行随机抽样产生新样本	1. 一般使用有放回（bootstrap）方法 2. 简单，但有过拟合风险
SMOTE	在少数类样本之间插值产生新样本 对于一个少数类样本x_i，使用 KNN（K 近邻）计算邻近的 K 个少数类样本，再随机选取其中一个样本，插值公式如下： $$x_{new} = x_i + \lambda * (x_{zi} - x_i)$$	1. 可能会产生噪声 2. 对于离散型特征，该方法使用 VDM 距离度量（SMOTEN）或转为 ohe 编码（SMOTENC）后处理 3. 衍生方法： a) Borderline：对「danger」样本（超过一半的 KNN 样本属于多数类，「边界」样本）进行生成；v1，插值时只选取少数类样本；v2，插值时任意选取样本 b) KMeans：聚类→过滤（保留高比例少数类簇）→生成（少数类越少，生成样本越多） c) SVM：对支持向量进行生成 d) …
ADASYN	生成过程类似 SMOTE，区别在于自动决定每	1. Δ为 K-NN 中多数类样本数量

	个少数类生成多少样本 样本总量：$G = (S_{maj} - S_{min}) * \beta$ 少数类样本权重：$\Gamma_i = \Delta_i / K / Z$ 少数类样本生成数量：$g_i = \Gamma_i * G$	2. 易受少数类离群点影响

2.2 欠采样

方法	思想	备注
随机欠采样	对多数类样本进行随机抽样产生子集	1. 原型选择类，可控制欠采样数量 2. 可使用有放回或无放回方法 3. 简单，但有信息丢失风险
NearMiss	采用启发式方法，从多数类样本中选取子集 1. v1：选择多数类样本中「N 少数类近邻」平均距离最小的 2. v2：选择多数类样本中「N 少数类远邻」平均距离最小的 3. v3：Step1，保留少数类样本的 M 近邻；Step2，选择多数类样本中 N 近邻平均距离最大的	1. 原型选择类，可控制欠采样数量 2. 计算复杂度高，v1 易受离群点影响
TomekLinks	从多数类样本中剔除类间重叠样本（Tomek's link，噪声） Tomek's link：两样本互为最近邻且分属不同类别	1. 原型选择类，数据清洗 2. 无法控制欠采样数量 3. 存在难分类信息丢失问题
ENN	从多数类样本中剔除「低支持」的样本 「低支持」：K 近邻中超一半不为该多数类	1. 原型选择类，数据清洗 2. 无法控制欠采样数量 3. 衍生方法： a) 无支持：K 近邻均为其他类 b) Repeated：重复多次剔除 c) AllKNN：重复多次剔除，每轮迭代中递增 KNN 尺度 d) …
CNN	从多数类样本中保留 1-NN 错分样本 具体步骤： Step 1.少数类样本集合 C，多数类样本集合 S Step 2.从 S 中随机选取一个样本放入 C 中，并训练 1-NN Step 3.遍历 S，逐样本，使用 1-NN 进行分类，将错分的样本加入 C 并更新 1-NN 规则 Step 4.重复 Step 3 直至没有可加入 C 的样本	1. 原型选择类，数据清洗 2. 无法控制欠采样数量 3. 对噪声敏感
OneSidedSelection	使用 TomekLinks 剔除噪声样本，类似 CNN，但在所有样本上应用 1-NN，并将错分样本加入 C，且不对 S 进行迭代	1. 原型选择类，数据清洗 2. 无法控制欠采样数量 3. 解决 CNN 噪声敏感问题
Neighbourh	从多数类样本中剔除「低支持」样本（使用	1. 原型选择类，数据清洗

oodCleanRule	ENN）和 3-NN 输出样本（正确分类）的并集	2. 无法控制欠采样数 3. 相比 CNN，关注剔除而非压缩
InstanceHardnessThreshold	从样本中剔除「分类概率低」的样本（使用任意分类算法）	1. 原型选择类，数据清洗 2. 无法控制欠采样数 3. 依赖分类算法精度
ClusterCentrolds	使用 KMeans 对多数类样本进行聚类，生成的聚类中心作为子集	1. 原型生成类 2. 子集样本是生成得到的而非选择

2.3 代码片段

```python
# imbalanced-learn==0.8.0
import pandas as pd

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler, SMOTE, SMOTENC
from imblearn.under_sampling import RandomUnderSampler, EditedNearestNeighbours


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_X, df_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]

## 上采样
# 随机上采样，2倍
ros = RandomOverSampler(sampling_strategy=2.0, random_state=2024)
df_X_=ros, df_y_ros = ros.fit_resample(df_X, df_y)

# SMOTE，5-NN
smote = SMOTE(random_state=2024, k_neighbors=5)
df_X_smote, df_y_smote = smote.fit_resample(df_X, df_y)

## 下采样
# 随机下采样，10%，不放回
rus = RandomUnderSampler(sampling_strategy=0.1, random_state=2024, replacement=False)
df_X_rus, df_y_rus = rus.fit_resample(df_X, df_y)

# ENN，10%，低支持
enn = EditedNearestNeighbours(sampling_strategy=0.1, random_state=2024, kind_sel='mode')
df_X_enn, df_y_enn = enn.fit_resample(df_X, df_y)
```

```python
## 上采样+下采样
# SMOTENC+TomekLink，2倍
smote_tl = SMOTETomek(sampling_strategy=2, random_state=2024, smote=SMOTENC)
df_X_smote_tl, df_y_smote_tl = smote_tl.fit_resample(df_X, df_y)

## 欠采样+集成学习
# EasyEnsemble，lightgbm * 10，10%
params = {
    'boosting_type': 'gbdt', 'objective': 'binary',
    'n_estimators': 100, 'learning_rate':0.1,
    'max_depth': 5, 'num_leaves':30,
    'max_bin':50, 'min_data_in_leaf':20,
    'feature_fraction': 0.8, 'bagging_fraction':0.8, 'bagging_freq':10,
    'reg_alpha': 0.01, 'reg_lambda':0.01,
    'feature_fraction_seed': 2024, 'bagging_seed': 2024,
    'n_jobs': -1
}
clf_base_lgb = lgb.LGBMClassifier(**params)
clf_ee = EasyEnsembleClassifier(base_estimator=clf_base_lgb, n_estimators=10,
                                sampling_strategy=0.1,
                                n_jobs=-1, random_state=2024, verbose=1)
clf_ee.fit(df_X.values, df_y.values)
```

二、特征工程

数据和特征决定了机器学习的上限，模型和算法只是逼近这个上限

1. 特征预处理

未经处理的特征可能存在以下问题：

1. 不同特征间量纲不一致，无法放在一起比较
2. 离散型特征无法直接入模
3. 存在缺失值
4. 存在异常值
5. 特征分布不满足模型假设
6. 特征中的噪声导致模型过拟合

1.1 无量纲化

针对连续型特征，通过去除不同特征间由于量纲不同引起的误差，从而提升模型收敛速度和模型精度（对于涉及距离计算的模型，无量纲化是必须的）

方法	思想	备注
StandardScaler	z-score 标准化，零均值、单位方差 $x' = (x - \mu)/\sigma$	1. 改善分布，消除分布产生的度量偏差 2. 使数据更符合模型统计假设，正态分布→标准正态分布， 3. 在噪声影响下可以更好的保持样本间距（相对于 MinMaxScaler）
MinMaxScaler	离差标准化（归一化），将数据映射到[0,1] $x' = (x - x_{min})/(x_{max} - x_{min})$	1. 输出范围确定 2. 易受异常值影响，鲁棒性差 3. 类似有 MaxAbsScaler， a) $x' = x/\max(\lvert x \rvert)$ b) 将数据映射到[-1,1]（正负值） c) 不会破坏稀疏结构
RobustScaler	鲁棒标准化，减去中位数再除以四分位距 $x' = (x - x_{median})/IQR$	1. 鲁棒性好，不受异常值影响 2. 同时可以保持异常值（离群点）特性

1.2 离散型特征处理

对于离散型特征，通常原始输入为字符串形式，大部分模型无法接收该输入；而数值型离散型特征入模后通常会被当作连续型特征处理，从而造成训练误差

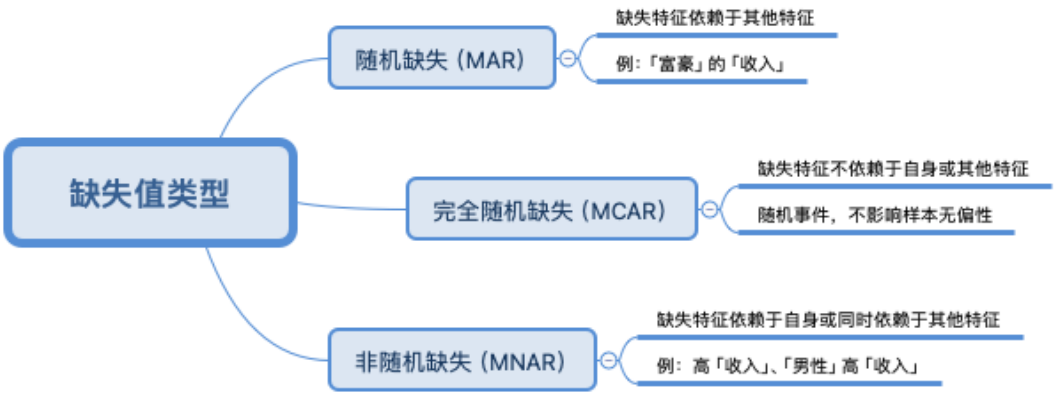
方法	思想	备注
Label Encoding	标签编码，使用字典方式，将特征的类别取值和递增整数相关联 [['A'],['B'],['C']]→[[0],[1],[2]]	1. 无监督方法 2. 适用于处理有序特征（针对目标） 3. 除作为连续型特征应用于一般模

		型外，更适合支持类别性质特征的模型（LightGBM、XGBoost）或作为 NN embedding 层的输入 4. 与 Ordinal Encoding（序列编码）类似，实际使用时仅输入有区别
One-Hot Encoding	独热编码，也称一位有效编码，使用 N 位状态寄存器对特征的 N 个状态编码 [['A'],['B'],['C']]→[[1,0,0],[0,1,0],[0,0,1]]	1. 无监督方法 2. 适用于处理无序特征（特征取值扩展到欧式空间，使距离度量计算更合理），对于有序特征则会丢失顺序信息（相较于 Label Encoding） 3. 在一定程度上扩增了特征维度，引入非线性，提升模型拟合能力，且便于后续做特征组合 4. 特征类别数量过多时（高基数），会产生高维稀疏矩阵，计算复杂度上升，有过拟合风险（维度灾难） 5. 相比基于树的模型（LightGBM、XGBoost），更适用于线性模型（LR）
Target Encoding	目标编码，也称均值编码，将特征的类别取值表示为其对应的目标概率估计 使用先验概率估计和后验概率估计的凸组合（加权平均）表示，并加入正则化方法（参数、CV）避免过拟合 $\hat{P}_k = \lambda * prior + (1-\lambda) * posterior$ $prior = \hat{P}(y = y_c)$ $posterior = \hat{P}(y = y_c \mid x = x_k)$ $\lambda = 1/(1+e^{(n-t)/f})$	1. 有监督方法 2. 适合对高基数特征编码 3. 对于回归任务，概率估计使用均值 4. 对于多分类任务，使用 one-vs-all 策略，会生成多个特征（目标类别） 5. 避免过拟合方法： a) 调整参数：阈值 t、斜率 f b) CV：k 折交叉验证，结果合并
WOE Encoding	证据权重编码（Weight of Evidence），将特征的类别取值表示为其对应的正负样本之间的差异 $WOE_k = \ln\left(\#y_k^+/\#y^+ \Big/ \#y_k^-/\#y^-\right)$	1. 有监督方法 2. 一般用于二分类任务 3. 量纲统一，特征内不同取值的间隔标准化（单位 WOE 变化差值恒定） 4. WOE 和 ln(odds)单调性一致（WOE 经转换可表示为 ln(odds)和常量），可将非线性特征转为线性，对 GLM 的使用很有必要 5. 结合分箱（类别取值多）可用于计算 IV，衡量特征预测能力 6. 常用于 LR 评分卡建模

1.3 缺失值处理

实际业务数据中，绝大部分数据都包含缺失值，且大部分模型无法接收含有缺失值的输入，

因此一般需要对含有缺失值的特征做预先处理

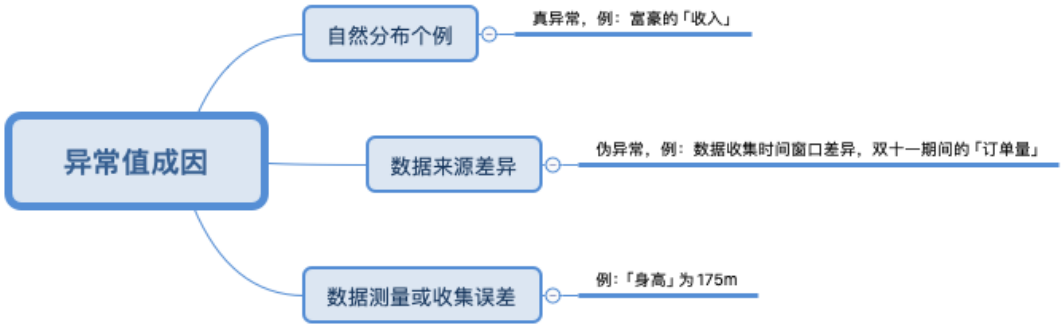


方法	思想	备注
删除	1. 删除含有缺失值的样本（存在多个特征缺失，且该类样本占比少） 2. 删除缺失值较多的特征（特征覆盖度<10%）	1. 删除法 2. 对于非 MCAR，存在信息丢失 3. 可以通过对完整样本加权，降低删除含缺失值样本的影响
简单填充	1. 连续型特征：均值、中位数、最大值、最小值 2. 离散型特征： a) 无序：众数 b) 有序：中位数、最大值、最小值 3. 时间序列：上次、下次观测结转，前后平均	1. 填充法 2. 存在人为增加噪声 3. 结合其他特征条件，例：同一「小区」下的平均「房价」 4. 最大值、最小值填充侧重于将特征取值归到两端，便于后续进行分箱处理或应用树模型寻找最优分裂点
回归填充	使用任意回归方法，将含有缺失值的特征作为 y，其他特征作为 X，使用完备样本进行拟合，对缺失数据进行预测	1. 填充法 2. 计算复杂度较高 3. 可根据特征相关性，选择合适的特征进行回归拟合 a) 低相关性：预测偏差大 b) 高相关性：特征冗余 c) 一般介于两者之间 4. 有过拟合风险
K 近邻填充	使用 K-NN 方法，计算含有缺失值的样本的 K 个近邻样本，并使用均值对缺失值进行填充	1. 填充法 2. 样本距离度量使用「缺失值加权欧氏距离」
多重填充	1. 重复展开单次填充（对填充值、模型参数、模型样本加噪），得到多个完整的样本集 2. 采用同样的分析方法对所有样本集进行分析 3. 基于分析结果对填充结果做选择或综合	1. 填充法 2. 计算复杂度高 3. 相比使用某一方法进行单次填充置信度更高
特殊值填充	使用-99999、-1、0、99999、'unknown'等非特征取值范围内的值进行填充，表示特征的「缺失值」取值	1. 填充法 2. 后续处理： a) 连续型特征：结合分箱

		处理（单独视为一箱），或视为填充最大值、最小值（效果同上） b) 离散型特征：结合编码处理（lbe 两端、ohe）
不处理	对含缺失值的样本不做任何处理，直接输入可以处理缺失值的模型	1. 忽略法 2. 相关模型有：RF、LightGBM、XGBoost（最大增益划分方向、默认划分方向）

1.4 异常值处理

异常数据，也称离群点，是实际业务数据分布的常态，该类数据在整体或某些特征维度（连续型特征）上显著不同于其他数据，与「正常数据」的分布有显著差异



在对异常数据中部分特征维度的异常值处理之前，我们还需要识别哪些是异常值。除了人工经验筛选外，我们还可以借助以下方法对大数据量样本进行异常值检测：

方法	思想	备注
统计模型方法	1. 3σ 准则：$(\mu-3\sigma,\mu+3\sigma)$范围外的数据为异常值（范围内占比 99.74%） 2. Grubbs 检验（最大归一化残差检验）：$G_i=(x_i-\mu)/\sigma$，设定置信度水平，查表确定是否为异常值（大于查表临界值） 3. 卡方检验：$\chi^2=\sum_{i=1}^{n}\frac{(x_i-E_i)^2}{E_i}$，同上 4. 切比雪夫不等式：$P\{\lvert X-\mu\rvert>\varepsilon\}<\frac{\sigma^2}{\varepsilon^2}$，至少有$(1-1/k^2)$的数据在$\mu\pm k\sigma$内。（$\mu\pm 4.5\sigma$范围内占比 95%） 5. 四分位距（箱线图）：$(Q_1-1.5*IQR, Q_3+1.5*IQR)$（范围内占比 99.3%）	1. 1、2、3 为参数化方法： a) 先验分布假设（正态） b) 基于坚实的统计学理论 c) 大多针对一元数据（3 适用于多维，2 用于多维时使用马氏距离） 2. 4、5、6 为非参数化方法： a) 没有先验分布假设 b) 5、6 可以通过观察数据分布，直观的找到异常值的大致范围 c) 需要指定其他参数，如 6 中直方图箱的数量、宽度、面积等。影响检测效果

	6. HBOS（基于直方图的离群点分数）： $HBOS(x) = \sum_{i=1}^{n} \ln(1/hist_i(x_i))$，分数高为离群点	
聚类方法	1. 使用任意聚类方法，离群点聚类为一小簇 2. 基于密度聚类方法，利用密度定义识别不属于任何簇的离群点	1. 1 为基于簇的方法 2. 2 为基于密度的方法，例如 DBSCAN 中的「噪声点」、FSDP 中的「cluster halo」 3. 检测效果依赖于聚类效果 4. 密度聚类方法参数多
K 近邻方法	通过计算 KNN 平均距离，降序得到头部离群点	1. 计算复杂度高 2. 参数 K 的选择影响检测效果
奇异点检测	One-Class SVM（单类支持向量机）： 1. OCSVM：通过构建与特征空间中的零点距离最大的超平面，将零点和所有样本分开 2. SVDD：构建超球体，最小化体积（样本到中心的距离，半径 R）	1. 训练集中不包含异常样本 2. 半监督学习 3. 使用训练后的模型对新数据检测离群点
离群点检测	1. Robust covariance（稳健协方差）： 假设数据（一维）服从正态分布，估计最适合的一组参数（均值、方差）进行拟合 2. isolation Forest（孤立森林）： 通过子采样构建孤立树（递归随机分割数据子集），计算样本点通过孤立树的路径长度，再计算整体孤立森林得分，识别离群点 $s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$ 3. LOF（局部异常因子）： 计算 KNN 内其他样本点的局部可达密度的平均值与该样本点的局部可达密度的比值，越大于 1 越可能为离群点 局部可达密度：KNN 内其他样本点到该点的可达距离的均值的倒数	1. 训练集中包含异常样本 2. 无监督学习 3. 用于检测自身离群点和新数据离群点 4. 同样可用于奇异点检测 5. 1 中，对于高维数据，估计参数为均值向量和协方差矩阵 6. 2 中，分布稀疏且距离密度高的样本较远的样本点（离群点），在孤立树中被切分的次数低，路径长度短 7. 3 同样为基于密度的方法。其中，可达距离表示样本点 KNN 的范围距离

通过异常值检测，我们确定了样本中的「正常数据」和「异常数据」，在建模之前，我们需要对「异常数据」中存在异常的特征维度进行处理，避免可能带来的模型拟合精度问题

方法	思想	备注
删除	1. 删除含有异常值的样本 2. 删除有较多明显异常的特征	1. 存在信息丢失问题 2. 2 中主要倾向于人工经验识别的「错误数据」
截断	根据人工经验或统计模型方法，对异常值进行截断： 1. 人工经验：例：「正常数据」的范围为（0,200），「异常数据」10000 截断至 500 2. 统计模型方法：例：四分位距方法的范围两端	1. 人工经验处理可以在降低异常值影响的情况下保持离群点特性 2. 统计模型方法更加鲁棒
分箱	对特征进行分箱转换，将异常值归到近邻箱中	1. 对异常值有很强的鲁棒性 2. 同《特征转换》中「分箱转换」
视为缺失值	将异常值视为缺失值，使用缺失值处理方法	同《缺失值处理》

不处理	对含有异常值的特征不做单独处理，依赖其他处理方法或模型： 1. 「无量纲化」处理方法降低异常值影响 2. 使用鲁棒性更好的模型进行拟合 3. 使用可以识别异常值的聚类方法	1. 1 中主要方法有 StandardScaler、RobustScaler 2. 2 中主要模型包括：RF、LightGBM、XGBoost（基于树的划分统计） 3. 3 中主要聚类模型包括：DBSCAN、FSDP

1.5 特征转换

除了前述的「无量纲化」（标准化转换）和「离散型特征处理」（编码转换），对于连续型特征，我们还可以通过其他转换，改善存在的分布问题和噪声问题，进一步提升模型拟合能力

方法	思想	备注
非线性转换	通过幂次变换改善特征分布（接近正态分布），尽可能满足模型假设（线性回归），提升模型拟合精度： 1. 对数转换： $$x^{(\lambda)} = log_{\lambda}(x)$$ 2. Box-Cox 变换： $$x^{(\lambda)} = \begin{cases} (x^{\lambda}-1)/\lambda, if\ \lambda \neq 0 \\ \ln(x), \quad if\ \lambda = 0 \end{cases}$$ 3. Yeo-Johnson 变换： $$x^{(\lambda)} = \begin{cases} \dfrac{[(x+1)^{\lambda}-1]}{\lambda}, & if\ \lambda \neq 0, x \geq 0 \\ \ln(x+1), & if\ \lambda = 0, x \geq 0 \\ -\dfrac{[(-x+1)^{2-\lambda}-1]}{2-\lambda}, & if\ \lambda \neq 2, x < 0 \\ -\ln(-x+1), & if\ \lambda = 2, x < 0 \end{cases}$$	1. 有利于方差稳定（消除异方差）和偏度最小化（正态分布指标），便于后续统计推断和假设验证 2. 一般使用 EDA 方法（直方图、QQ 图）观察确定存在分布问题的特征，再进行处理 3. 1 适用于取值范围较大的特征。通过缩小特征尺度，使特征更平滑，降低离群点影响（2、3 中的对数变换同理） 4. 1、2 要求特征值为正数 5. 2、3 中的参数λ可以使用极大似然、Bayes 方法进行估计 6. 1 也常用于对回归预测的 y 值进行转换，避免预测结果为负
分箱转换	通过将特征取值范围划分为连续的不同区间（箱），并按序对区间进行统一编码，使连续型特征转换为离散型有序特征（连续特征离散化），从而避免噪声和异常值的影响，增强模型泛化性： 1. 人工分箱：人工设定分箱阈值 2. 等距分箱：每个区间的宽度相同 3. 等频分箱：每个区间的样本数相同 4. 聚类分箱：使用聚类方法，聚类得到的簇即为分箱结果 5. CART 分箱：使用 CART 算法对 y 进行拟合，得到的叶子节点即为分箱结果 6. 卡方分箱：循环计算相邻区间（初始为单个样本）的卡方值，对最小卡方值区间对进行合并，直至满足设定条件	1. 2、3、4 为无监督方法 a) 需要人工设定分箱数 b) 4 中一般使用 KMeans 2. 5、6 为有监督方法 a) 5 为 top-down 方法，6 为 bottom-up 方法 b) 5 中使用 Gini 系数作为分箱指标，也可使用其他指标： i. KS：对应 Best-KS 分箱方法（要求 bad rate 单调） ii. IV… c) 5 中需要设定最小分箱

	$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{J}\left(N_{i,j} - E_{i,j}\right)^2 / E_{i,j}$ $E_{i,j} = N_i * C_j / N$	占比（0.05）、Gini 系数阈值、最大分箱数（3～10）等参数 d) 为防止过拟合，5 中常使用剪枝方法进行处理 e) 6 适用于分类任务；可以通过设定卡方阈值（自由度&置信度）或最小、最大区间数确定分箱结果 3. 可以对分箱后的特征使用 ohe 编码处理： a) 引入非线性 b) 稀疏向量运算快 c) 结合「特征组合」方法，可进一步提升模型效果 4. 一般用于「海量离散特征+简单模型」策略 5. 可用于计算 IV，衡量特征预测能力 6. 常用于 LR 评分卡建模（结合 WOE 编码）

1.6 代码片段

「无量纲化」

```python
# 无量纲化
import pandas as pd
import joblib

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# z-score标准化
ss = StandardScaler()

# 训练集拟合&转换
X_train_ss = ss.fit_transform(df_train_X)
joblib.dump(ss, 'ss.pickle')

# 测试集以训练集标准进行转换
ss = joblib.load('ss.pickle')
X_test_ss = ss.transform(df_test_X)
```

「离散型特征处理」

```python
# 离散型特征处理
import pandas as pd
import joblib

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# Label Encoding
# 训练集拟合&转换
dict_lbe = {}
list_X_train_lbe = []

for x in df_train_X.columns:
    lbe = LabelEncoder()
    df_X_lbe_each = pd.DataFrame(lbe.fit_transform(df_train_X[x]), columns=[x])
    dict_lbe[x] = lbe
    list_X_train_lbe.append(df_X_lbe_each)

df_X_train_lbe = pd.concat(list_X_train_lbe, axis=1)
joblib.dump(dict_lbe, 'dict_lbe.pickle')

# 测试集以训练集标准进行转换
dict_lbe = joblib.load('dict_lbe.pickle')
list_X_test_lbe = []

for x in df_test_X.columns:
    list_feat_values_unseen = list(set(df_test_X[x].unique())-set(dict_lbe[x].classes_))
    if len(list_feat_values_unseen) > 0: # 未见过的特征取值处理，归为-1（保证训练集对应特征有该取值）
        df_test_X[x].replace(list_feat_values_unseen, -1, inplace=True)
    df_X_lbe_each = pd.DataFrame(dict_lbe[x].transform(df_test_X[x]), columns=[x])
    list_X_test_lbe.append(df_X_lbe_each)

df_X_test_lbe = pd.concat(list_X_test_lbe, axis=1)
```

```python
# One-Hot Encoding
# 训练集拟合&转换
enc = OneHotEncoder(categories='auto', handle_unknown='ignore')
clf_ohe = enc.fit(df_train_X)

df_X_train_ohe = pd.DataFrame(clf_ohe.transform(df_train_X).toarray(), columns=clf_ohe.get_feature_names_out())
joblib.dump(clf_ohe, 'clf_ohe.pickle')

# 测试集以训练集标准进行转换
clf_ohe = joblib.load('clf_ohe.pickle')

for i, x in enumerate(df_test_X.columns):
    list_feat_values_unseen = list(set(df_test_X[x].unique())-set(clf_ohe.categories_[i]))
    if len(list_feat_values_unseen) > 0: # 同上
        df_test_X[x].replace(list_feat_values_unseen, -1, inplace=True)

df_X_test_ohe = pd.DataFrame(clf_ohe.transform(df_test_X).toarray(), columns=clf_ohe.get_feature_names_out())
```

```python
# Target Encoding, scikit-learn>=1.3.2
from sklearn.preprocessing import TargetEncoder
# 训练集拟合&转换
te = TargetEncoder(target_type='binary', smooth='auto', cv=5)
X_train_te = enc_auto.fit_transform(df_train_X.values, df_train_y.values)
df_X_train_te = pd.DataFrame(X_train_te, columns=df_train_X.columns)
joblib.dump(te, 'te.pickle')

# 测试集以训练集标准进行转换
te = joblib.load('te.pickle')
X_test_te = enc_auto.transform(df_test_X.values)
df_X_test_te = pd.DataFrame(X_test_te, columns=df_test_X.columns)
```

「缺失值处理」

```
# 缺失值处理
import pandas as pd
import numpy as np
import joblib

from sklearn.model_selection import train_test_split
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# 删除
def df_des(df):
    df_miss = df.isnull().mean() * 100
    df_miss = pd.DataFrame({'Miss Percent(%)': df_miss})

    return pd.concat([df.describe().T, df_miss], axis=1)

# 训练集处理
df_train_X_des = df_des(df_train_X)
df_train_X = df_train_X[[x for x in df_train_X.columns if x not in df_train_X_des[df_train_X_des['Miss Percent(%)']>90.0].index]]
df_train_X_des.to_csv('df_train_X_des.csv', encoding='utf-8')

# 测试集以训练集标准处理
df_train_X_des = pd.read_csv('df_train_X_des.csv', encoding='utf-8')
df_test_X = df_test_X[[x for x in df_test_X.columns if x not in df_train_X_des[df_train_X_des['Miss Percent(%)']>90.0]['Unnamed: 0']]]
```

```
# 填充—简单&特殊
# 训练集处理
dict_fillna = {
    'aaa': -1,
    'bbb': df_train_X['bbb'].mean(),
}
df_train_X['aaa'].fillna(dict_fillna['aaa'], inplace=True)
df_train_X['bbb'].fillna(dict_fillna['bbb'], inplace=True)
joblib.dump(dict_fillna, 'dict_fillna.pickle')

# 测试集以训练集标准处理
dict_fillna = joblib.load('dict_fillna.pickle')
df_test_X['aaa'].fillna(dict_fillna['aaa'], inplace=True)
df_test_X['bbb'].fillna(dict_fillna['bbb'], inplace=True)

# 填充—回归
# 训练集处理
ii = IterativeImputer(missing_values=np.nan, max_iter=10, n_nearest_features=50, random_state=2024)
ii.fit(df_train_X.values)
train_X_ii = ii.transform(df_train_X.values)
joblib.dump(ii, 'ii.pickle')

# 测试集以训练集标准处理
ii = joblib.load('ii.pickle')
test_X_ii = ii.transform(df_test_X.values)
```

「异常值处理」

```
# 异常值处理
import pandas as pd
import numpy as np
import joblib

from sklearn.model_selection import train_test_split


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# 使用四分位距检测并截断
Q1 = np.quantile(df_train_X['aaa'], 0.25)
Q3 = np.quantile(df_train_X['aaa'], 0.75)
IQR = Q3 - Q1
dict_outlier = {
    'aaa': [Q1-1.5*IQR, Q3+1.5*IQR]
}

index_outlier_down = df_train_X[(df_train_X['aaa']<dict_outlier['aaa'][0])].index
index_outlier_up = df_train_X[(df_train_X['aaa']>dict_outlier['aaa'][1])].index

df_train_X.loc[index_outlier_down, 'aaa'] = dict_outlier['aaa'][0]
df_train_X.loc[index_outlier_up, 'aaa'] = dict_outlier['aaa'][1]
joblib.dump(dict_outlier, 'dict_outlier.pickle')

# 测试集以训练集标准处理
dict_outlier = joblib.load('dict_outlier.pickle')
index_outlier_down = df_test_X[(df_test_X['aaa']<dict_outlier['aaa'][0])].index
index_outlier_up = df_test_X[(df_test_X['aaa']>dict_outlier['aaa'][1])].index

df_test_X.loc[index_outlier_down, 'aaa'] = dict_outlier['aaa'][0]
df_test_X.loc[index_outlier_up, 'aaa'] = dict_outlier['aaa'][1]
```

「特征转换」

```python
# 特征转换
import pandas as pd
import numpy as np
import joblib

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PowerTransformer, KBinsDiscretizer


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# Yeo-Johnson变换，z-score标准化
pt = PowerTransformer(method='yeo-johnson', standardize=True, copy=True)
pt.fit(df_train_X)
train_X_pt = pt.transform(df_train_X)
joblib.dump(pt, 'pt.pickle')

# 测试集以训练集标准处理
pt = joblib.load('pt.pickle')
test_X_pt = pt.transform(df_test_X)

# 等频分箱，5箱，lbe
kbd = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
kbd.fit(df_train_X)
train_X_kbd = kbd.transform(df_train_X)
joblib.dump(pt, 'kbd.pickle')

# 测试集以训练集标准处理
kbd = joblib.load('kbd.pickle')
test_X_pt = kbd.transform(df_test_X)
```

2. 特征选择

特征预处理完成后，基于以下原因，我们还需要对特征进行筛选，再输入算法模型中训练：

1. 存在无关特征（X 与 y），对模型来说是噪声，容易造成过拟合
2. 存在冗余特征（X 与 X），影响模型的可解释性
3. 特征维度高，模型计算复杂度高，建模资源和成本消耗大
4. 样本量一定时，模型效果不会随着特征维度增加而一直升高（维度灾难），对于使用高维度特征得到的模型，根据奥卡姆剃刀原理，如果能找到拟合效果相同且特征维度更低的模型，低维特征模型往往是更好的模型，具有更好的泛化性

2.1 Filter

过滤法，按照特征自身的发散性或与目标（因变量）、其他特征（自变量）之间的相关性进行评分，通过设定阈值或选择特征个数，进行特征筛选

方法	思想	备注
方差	计算各特征方差，设定预期阈值，筛选方差大于阈值的特征	1. 基于特征自身发散性 2. 可基于特征分布计算阈值： a) 布尔特征：0-1 分布，$Var(x) = p * (1 - p)$，过滤 0 占比为 p 的特征 b) … 3. 对于实际业务，通常筛掉方差为 0 的特征（缺失导致全部填充为默认值）

<table>
<tr>
<td>单因素方差分析</td>
<td>通过假设检验方法（F 检验），计算特征对于目标、其他特征是否有显著性影响，筛选 P 值小于等于显著性水平（置信度）α的特征（原假设为不显著）
$$F=\frac{S_b/(r-1)}{S_w/(n-r)}\sim F(r-1,n-r)$$
$$P(F(r-1,n-r),F)\le\alpha$$</td>
<td>1. 基于特征与目标、其他特征间的相关性
2. 适用于离散型特征在回归任务（目标为连续型）下的相关性计算；对于分类任务（目标为离散型）下的连续型特征，可以反向计算
3. 也可用于离散型特征和连续型特征间的相关性计算
4. 置信度α一般设定为 0.05
5. 对于多因素与目标间的相关性计算（先验条件、因果推断控制组+特征），可以通过多元线性回归分析计算单特征 p 值（整体 F 检验，单特征不显著假设）</td>
</tr>
<tr>
<td>卡方检验</td>
<td>通过假设检验方法（皮尔逊卡方检验），计算特征对于目标、其他特征是否有显著性影响，筛选 P 值小于等于显著性水平α的特征
$$\chi^2_{Pearson}=\sum_{i=1}^{R}\sum_{j=1}^{C}\frac{(A_{i,j}-T_{i,j})^2}{T_{i,j}}\sim\chi^2(R-1,C-1)$$
$$P(\chi^2(R-1,C-1),\chi^2_{Pearson})\le\alpha$$</td>
<td>1. 基于特征与目标、其他特征间的相关性
2. 适用于离散型特征在分类任务下的相关性计算
3. 也可用于离散型特征之间的相关性计算
4. 置信度α一般设定为 0.05</td>
</tr>
<tr>
<td>IV 值</td>
<td>信息价值（Information value），由特征各分组 WOE 加权求和计算（消除分组数量差异带来的误差），衡量特征对目标的预测能力。筛选 IV 值大于阈值的特征
$$IV=\sum_{i=0}^{K}(\#y_i^+/\#y^+-\#y_i^-/\#y^-)*WOE_i$$
<table>
<tr><th>IV 值范围</th><th>预测能力</th></tr>
<tr><td>(0,0.02)</td><td>无</td></tr>
<tr><td>[0.02,0.1]</td><td>弱</td></tr>
<tr><td>(0.1,0.3]</td><td>中等</td></tr>
<tr><td>(0.3,0.5]</td><td>强</td></tr>
<tr><td>(0.5,+∞)</td><td>异常</td></tr>
</table>
</td>
<td>1. 基于特征与目标间的相关性
2. WOE 相关同《离散型特征编码》中「WOE Encoding」
3. 一般用于二分类任务，常用于 LR 评分卡建模
4. 计算时需要对特征做「分箱转换」；受分箱效果影响，一般分 3～5 箱，分箱过多 IV 值偏高（不准确）</td>
</tr>
<tr>
<td>相关系数</td>
<td>计算特征与目标、其他特征的下列系数以及对应 P 值，筛选大于阈值且显著的特征：
1. 皮尔逊相关系数：
$$\rho_{X,Y}=\frac{cov(X,Y)}{\sigma_X*\sigma_Y}$$
2. 斯皮尔曼相关系数：
$$r_s=1-\frac{6*\sum_{i=1}^{n}d_i^2}{n*(n^2-1)}$$
3. 肯德尔相关系数：</td>
<td>1. 基于特征与目标、其他特征间的相关性
2. 1、2、3 的取值范围为[-1,1]，表示负相关→不相关→正相关，绝对值越大相关性越强
3. 1 为线性相关系数；属于参数统计方法（t 检验要求数据为正态分布）；适用于连续型特征；易受异常值影响</td>
</tr>
</table>

	$$\tau_b = \frac{n_c - n_d}{\sqrt{(n_0 - n_1) * (n_0 - n_2)}}$$ 4. MIC（最大信息系数）： 不同尺度下归一化互信息的最大值 $$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) * log_2\left(\frac{p(x,y)}{p(x) * p(y)}\right)$$ $$mic(X;Y) = max_{m*n<B} \frac{I(X;Y)}{log_2(\min(m,n))}$$	4. 2 为秩相关系数，通过计算秩次（特征值 rank，值相同取平均值）差值 d 表示相关性，可用于表示非线性相关性；属于非参数统计方法；适用于连续型特征或离散型有序特征 5. 3 中为秩相关系数，通过计算协和对（n_c）、不协和对（n_d）、不变对（n_1、n_2）的数量差异表示相关性，可用于表示非线性相关性；属于非参数统计方法；适用于连续型特征或离散型有序特征 6. 3 中τ_b适用于特征尺度相等的情况，特征尺度不等则可以使用$\tau_c = \frac{2*(n_c - n_d)}{n^2*(m-1)/m}$ 7. 4 可以表示非线性相关性；适用于连续型特征或离散型有序特征，使用时需要对特征做离散化处理（$m * n$），不同尺度表示离散化的个数和方式 8. 4 相对于其他方法，计算复杂度低，鲁棒性高，但数据量要求高

2.2 Wrapper

包装法，基于模型和评价指标，对特征集进行迭代多次训练，在每轮迭代中，根据设定的阈值或特征数量要求，进行特征筛选

方法	思想	备注
递归特征消除	根据模型的特征权重系数或重要性指标，在每次训练完成后过滤低于阈值或尾部的特征，迭代多次处理直至满足要求，余下的特征子集即为所求	1. 计算复杂度较高，$o(n)$ 2. 用于过滤无关特征 3. 可结合 CV 方法和评价指标输出每次迭代处理后的模型效果，选择最合适的特征子集 4. 相关的模型和指标： a) 广义线性回归：线性回归、逻辑回归等；特征权重系数、P 值 b) SVM：线性核；特征权重系数 c) 树模型：XGBoost、

		LightGBM 等；特征重要性
后向逐步特征消除	基于模型和评价指标，在每轮迭代处理中，循环使用n_i-1维特征进行模型训练（得到n_i个模型），过滤“当前指标最优且优于上一轮”的模型所删除的特征，迭代多次处理直至满足要求，余下的特征子集即为所求	1. 计算复杂度高，$o(n^2)$ 2. 用于过滤无关、冗余特征 a) 无关特征：目标为 y；评价指标包括：AUC、loss（logloss、rmse）、acc、recall、R^2或 AIC、BIC（特征数量角度）等 b) 冗余特征：目标为 1 维 X；降低多重共线性，改善模型可解释性，提升模型收敛速度；评价指标：$VIF=1/(1-R^2)$；过滤最高且高于阈值（一般为 10）的特征 3. 可引入验证集或结合 CV 方法，得到的评价指标更准确，防止过拟合
前向逐步特征构造	基于模型和评价指标，在每轮迭代处理中，循环增加 1 维特征（有放回；初始 0）进行模型训练，筛选“当前指标最优且优于上一轮”的模型所增加的特征，迭代多次处理直至满足要求，最终的特征子集即为所求	1. 计算复杂度高，$o(n^2)$ 2. 用于过滤无关特征 3. 其他同「后向逐步特征消除」

2.3 Embedded

嵌入法，利用模型自身的特征选择能力，不进行额外的特征筛选；或考虑到实际工程的计算资源成本，根据模型的特征权重系数和重要性指标，通过设定阈值或选择特征个数，进行特征筛选（类似于《Wrapper》中非迭代的「递归特征消除」）

方法	思想	备注
基于惩罚项	基于带 L1 正则项的线性模型，对特征进行拟合，筛选权重系数不为 0 特征	1. 仅适用于线性模型($y=W^TX+b$)在 L1 正则项的作用下产生稀疏解的情况 2. 大部分权重系数为 0 的特征为无关特征 3. 具有高相关性的特征可能存在某一个的权重系数被置 0 的情况，可结合 L2 正则项（降权平滑）做进一步筛选：额外筛选 L2 相近且 L1 为 0 的特征（相对于已筛选特征），再决定是否去除冗余特征
基于树模型	基于树模型对特征重要性的计算，筛选大于阈值的特征	1. 特征重要性计算，以 XGBoost 为例： a) 特征分裂次数 b) 特征分裂覆盖样本数 c) 特征分裂增益

2.4 其他方法

方法	思想	备注
PI	Permutation Importance（置换特征重要性），计算改变特征后，模型预测的误差： Step 1. 将数据划分为训练集和验证集 Step 2. 使用基模型对训练集做拟合 Step 3. 对训练好的模型使用验证集进行评价 Step 4. 循环计算每个特征的重要性 Step 4.1 对验证集上的单个特征进行随机打乱 Step 4.2 重新对验证集进行评价，计算单次评价差异 Step 4.3 重复 Step 4.1 ~ 4.2 n 次，计算平均评价差异	1. 没有模型限制以及对特定类型特征的偏好 2. 通过对验证集的评价，体现了特征的泛化能力 3. Step 4.3 中 n 越大统计结果越显著

2.5 代码片段

```python
# 特征选择
import pandas as pd
import numpy as np
import joblib
import lightgbm as lgb

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.inspection import permutation_importance


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

list_feats_x = list(df_train_X.columns)

# Step1：方差筛选
df_train_des = df_train_X.describe().T
list_feats_x_std = [x for x in list_feats_x if x in df_des[df_des['std']>0].index] # 保持顺序，下同

# Step2：单因素方差分析
# 连续型特征，分类任务，P值显著
selector = SelectKBest(f_classif, k=1).fit(df_train_X[list_feats_x_std], df_train_y)
df_ht = pd.DataFrame(
    {
        'feature': list_feats_x_std,
        'P_value': selector.pvalues_
    }
)
list_feats_x_ht = [x for x in list_feats_x_std
                   if x in df_ht[df_ht['P_value']<=0.05]['feature'].values.tolist()]
```

```python
# Step3：PI
df_train_train_X, df_train_val_X, df_train_train_y, df_train_val_y = \
    train_test_split(df_train_X[list_feats_x_ht], df_train_y, test_size=0.2, random_state=2024)

estimator_pi = lgb.LGBMClassifier(importance_type='gain')
estimator_pi.fit(df_train_train_X, df_train_train_y)

pi = permutation_importance(estimator=estimator_pi, X=df_train_val_X, y=df_train_val_y, n_jobs=8)

df_pi = pd.DataFrame(
    {
        'feature': list_feats_x_ht,
        'permutation_importance_mean': pi.importances_mean
    }
)
list_feats_x_pi = [x for x in list_feats_x_ht
                   if x in df_pi[df_pi['permutation_importance_mean']<=0.0]['feature'].values.tolist()]

# 训练集应用筛选结果
df_train_X_ = df_train_X_[list_feats_x_pi]
joblib.dump(list_feats_x_pi, 'list_feats_x_pi.pickle')

# 测试集以训练集标准处理
list_feats_x_pi = joblib.load('list_feats_x_pi.pickle')
df_test_X_ = df_test_X[list_feats_x_pi]
```

3. 降维

对于特征维度较高的数据，模型计算复杂度高，建模资源和成本消耗大。除了使用《特征选择》方法降低特征维度，我们还可以使用其他方法，通过将高维样本映射到低维空间的方式，实现特征降维

P.S.相对于《特征选择》方法，可解释性差

方法	思想	备注
PCA	Principal Component Analysis（主成分分析），通过对中心化数据矩阵（特征维度 0 均值）的协方差矩阵进行特征值分解，将高维数据矩阵映射到 top 特征值对应的特征向量空间（投影方差大（从高到低）且两两正交（线性不相关）），从而实现降维 $E^T\left(\frac{1}{m}XX^T\right)E=\Lambda=\begin{pmatrix}\lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \\ & & & \lambda_n\end{pmatrix}$	1. 无监督方法，以自身方差衡量信息量，保留主成分 2. 降低噪声（小特征值对应的特征向量维度） 3. 特征值分解计算效率低，可以使用 SVD（存在高效、准确的特征值分解方法） 4. 本质上是线性变换方法，对于非线性数据，可以通过核技巧（KPCA），将数据映射到高维线性可分空间，再进行降维
LDA	Linear Discriminant Analysis（线性判别分析），通过将高维数据映射到低维空间，使得目标类内散度尽可能小，类间散度尽可能大，从而实现降维 $\underset{W}{\operatorname{argmax}}J(W)=\frac{\prod_{diag}W^TS_bW}{\prod_{diag}W^TS_wW}$	1. 有监督方法，本质上用于分类 2. 降维后的维度上限：min(n_class-1,n_feature)；适用于高维多分类（人脸识别） 3. 最终计算同样使用特征值分解思想
Embedding	神经网络嵌入学习，通过构建维度递减的 MLP，对高维数据进行目标拟合，最小化目标损失，抽取输出层的低维输出数据，从而实现降维 $\hat{y}=MLP(X)$ $min\ loss(y,\hat{y})$	1. 有监督方法 2. 输出层的低维数据表示对高维输入特征的深层主要信息抽取（对于目标） 3. 类似的有传统的词嵌入学习
AutoEncoder	自动编码器（神经网络），通过构建编码器→低维隐变量→解码器结构，重构输入特征，最小化重构损失，中间的低维隐变量即为所求 $h=Encoder(X)$ $\hat{X}=Decoder(h)$ $min\ loss(X,\hat{X})$	1. 无监督方法（自监督方法） 2. 中间的低维隐层输出表示对原始高维输入特征的主要信息压缩（对于自身重构） 3. 编码器和解码器可以使用 MLP 结构，组成深度自动编码器（整体训练；不同于栈式自动编码器，逐层训练）

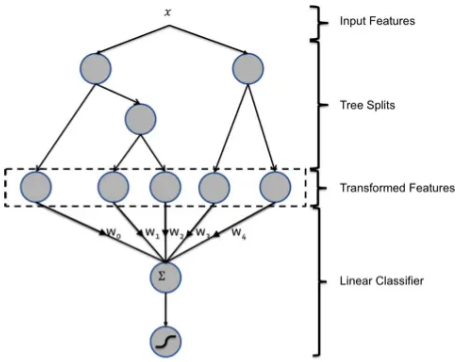
3.1 代码片段

```python
# 降维
import pandas as pd
import numpy as np
import joblib

from sklearn.model_selection import train_test_split
from sklearn.decomposition import KernelPCA


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# KPCA，降低到10维
kpca = KernelPCA(n_components=10, kernel='rbf')
kpca.fit(df_train_X, df_train_y)
train_X_kpca = kpca.transform(df_train_X)
joblib.dump(kpca, 'kpca.pickle')

# 测试集以训练集标准处理
kpca = joblib.load('kpca.pickle')
test_X_kpca = kpca.transform(df_test_X)
```

4. 特征组合

为了进一步提升模型拟合能力，增强模型的非线性表达能力，除了依赖人工经验对特征进行进一步加工、衍生（可解释性强，但费时费力），我们还可以通过一些自动化的方法对特征进行交叉（与）组合（或），发现潜在的有效特征
P.S.相对于人工经验加工，可解释性差

方法	思想	备注
简单交叉组合	通过对原始特征两两进行+、-、*、/等操作得到新特征	1. 使用数学方法，可生成$(n*(n-1)/2)*n_operation$个交叉组合新特征 2. 适用于连续型特征 3. 生成的绝大部份特征可能为无效特征，可以结合《特征筛选》方法选择有效特征 4. 相关实例：mljar
多项式生成	通过给定参与生成的特征和最大阶数，根据范德蒙矩阵，生成阶数和小于等于最大阶数的所有特征组合（乘积） $\begin{bmatrix}1 & x_1 & x_1^2\\ 1 & x_2 & x_2^2\end{bmatrix}$ $(x_1,x_2)\overset{d=2}{\Longrightarrow}(1,x_1,x_2,x_1^2,x_1x_2,x_2^2)$	1. 使用数学方法，可生成$(d-1)*C_n^1+(d-1)*C_n^2+(d-2)*C_n^3+\cdots+C_n^d$个最高为 d 阶的交叉新特征 2. 适用于连续型特征 3. 生成的大量无效特征处理同上 4. 参与生成的特征数和最大阶数越大，计算复杂度越高
GBDT+LR	通过对原始特征使用 GBDT 进行拟合，得到的叶子节点的稀疏编码表示（索引 ohe）即为新特征，再输入到 LR 中进行拟合	1. 有监督方法 2. 叶子节点的稀疏编码表示为树模型隐式特征交叉的结果

		3. GBDT 相比于单个决策树: a) 多个决策树集成，表达能力强，泛化性好，能够发现更多且有效的特征交叉表示（单个决策树需要更多深度，容易过拟合，且生成的特征更稀疏） b) Boosting 对残差的不断拟合优先发现对整体区分度大的特征，思路更合理 4. 海量离散特征+简单模型策略 5. 生成的特征维度高，因此数据量要求高，否则容易过拟合
模型内在处理	对原始特征直接使用特定的模型进行拟合，应用模型内部的特征交叉组合处理，不做额外的显式处理: 1. FM：Factorization Machine（因子分解机） $y(X) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j$ 2. 树模型: 内部节点根据最大增益选择合适的特征和划分点不断进行分裂, 每层分裂结果相当于对上一层做特征交叉处理 3. 神经网络: 每层神经元的输入表示对上一层神经元输出的多项式组合的激活函数变换 $A^l = \sigma^l(Z^l) = \sigma^l(W^l A^{l-1} + B^l)$	1. 有监督方法 2. 隐式特征交叉组合 3. 1 应用 2 阶多项式模型, 隐含了对特征的 2 阶特征交叉；用于解决稀疏数据的特征组合问题；由此衍生的模型有 FFM 4. 2 中例，深度为 2 的分裂节点 $(x_1 > a)\&(x_2 < b)$，表示 2 阶特征交叉；特征交叉阶数随深度增加 5. 3 本质为特征的多项式组合, 通过非线性激活函数增加非线性表达, 如 sigmoid、tanh、relu 等; 可结合 1 的特征交叉思想，衍生模型有 DeepFM、DCN、xDeepFM、NFM 等

4.1 代码片段

```python
# 特征组合
import pandas as pd
import numpy as np
import joblib
import lightgbm as lgb

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

# GBDT+LR (ElasticNet, L1+L2)
params = {
    'boosting_type': 'gbdt', 'objective': 'binary',
    'n_estimators': 10, 'learning_rate':0.1,
    'max_depth': 5, 'num_leaves':30,
    'max_bin':50, 'min_data_in_leaf':20,
    'feature_fraction': 0.8, 'bagging_fraction':0.8, 'bagging_freq':10,
    'reg_alpha': 0.01, 'reg_lambda':0.01,
    'feature_fraction_seed': 2024, 'bagging_seed': 2024,
    'n_jobs': -1
}

clf_lgb = lgb.LGBMClassifier(**params)
clf_lgb.fit(df_train_X, df_train_y)
joblib.dump(clf_lgb, 'clf_lgb.pickle')

train_X_lgb = clf_lgb.predict(df_train_X, pred_leaf=True)

enc = OneHotEncoder(categories='auto', handle_unknown='ignore')
clf_ohe = enc.fit(train_X_lgb)
joblib.dump(clf_ohe, 'clf_ohe.pickle')

train_X_lgb_ohe = clf_ohe.transform(train_X_lgb)

clf_lr = LogisticRegression(penalty='elasticnet', C=0.1)
clf_lr.fit(train_X_lgb_ohe, df_train_y)
joblib.dump(clf_lr, 'clf_lr.pickle')

clf_lr.predict_proba(train_X_lgb_ohe)
```

```python
# 测试集以训练集标准处理
clf_lgb = joblib.load('clf_lgb.pickle')
test_X_lgb = clf_lgb.predict(df_test_X, pred_leaf=True)

clf_ohe = joblib.load('clf_ohe.pickle')
test_X_lgb_ohe = clf_ohe.transform(test_X_lgb)

clf_lr = joblib.load('clf_lr.pickle')
clf_lr.predict_proba(test_X_lgb_ohe)
```

三、模型构建

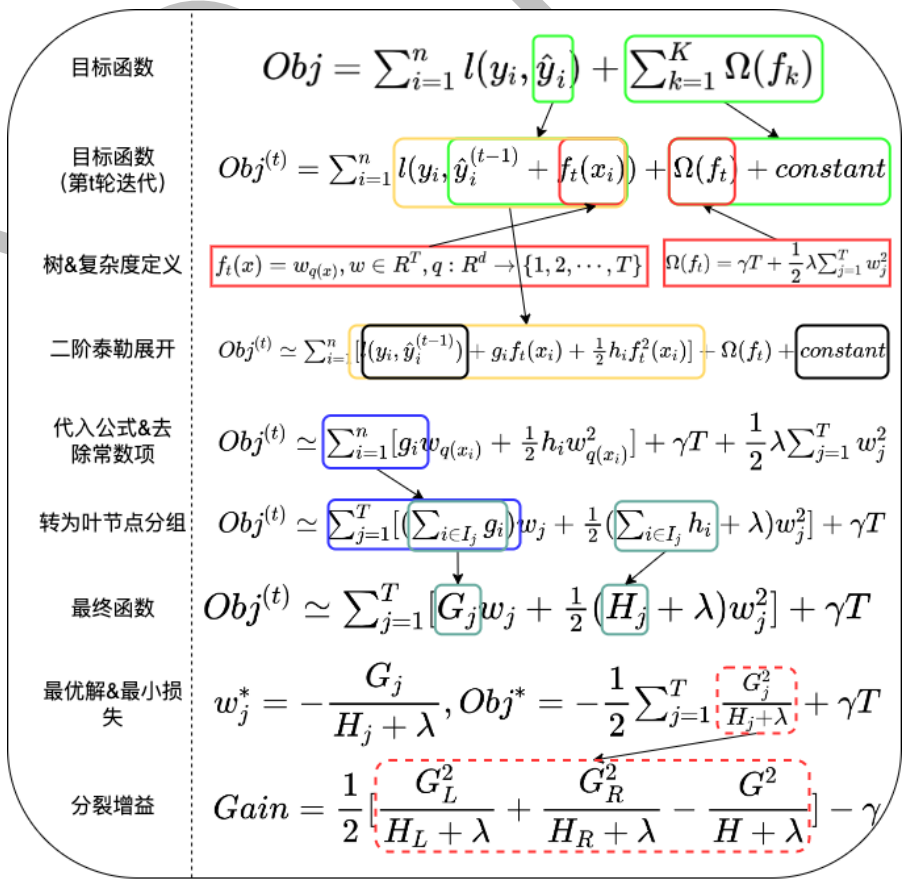
根据业务目标，选择合适的模型，对经过《数据集处理》和《特征工程》的数据进行建模。这里主要介绍业界常用做 baseline 的 XGBoost 模型的训练和优化

1. 模型训练

XGBoost，eXtreme Gradient Boosting（极限梯度提升），本质上是 GBDT 的改进算法，同样为加法模型，前向分步算法；其相对于 GBDT 主要有以下几方面改进：

改进点	思想	备注
正则项	显式增加基于叶子节点个数（T）、叶子节点权重（w）计算的表示每棵树的复杂度的正则项	1. 用于 gbtree 基分类器 2. 相当于做了预剪枝
二阶导数	代价函数使用二阶泰勒展开，可以同时考虑一阶、二阶导数	1. 一阶导数无法保证全局最优（除非目标为凸函数）
基分类器	支持多种基分类器，如：gbtree、gblinear、DART 等	1. 根据具体情况选择
行列采样	在每轮迭代中，可以基于行（数据）抽样、列（特征）抽样构建当前基分类器	1. 避免过拟合
缺失值处理	通过将缺失值带入分裂后的左右子节点，计算最大增益，自动学习分裂方向	1. 对于稀疏特征（ohe）、含有大量 0 值的特征同理 2. 预测时缺失值（训练不含缺失）默认分裂到左侧子节点

理论推导一图流：



在具体构建每棵树时（每轮迭代中），我们根据以下方法进行树节点分裂，选择最佳分裂点：

方法	思想	备注
精确贪心算法	枚举所有特征和可能的划分，计算最优增益 $\max\left(\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{G^2}{H+\lambda}\right)$	1. 计算复杂度高，$o(n*m)$ 2. 按特征值排序
近似算法	对于每个特征，只计算分位点的增益	1. 计算复杂度低，$o(n)$ 2. 实际对特征值排序后的数据按照h_i加权计算分位点 $Obj^{(t)}=\frac{1}{2}\sum_{i=1}^{n}\frac{1}{2}h_i[f_t(x_i)+g_i/h_i]^2+\Omega(f_t)+constant$

实际训练时，利用现有的 API，根据目标（分类（二、多）、回归、排序）选择合适的基分类器和损失函数（对应《模型优化》中的「Booster」和「objective」参数），设定好模型超参数，即可对经过《数据集处理》和《特征工程》的数据进行训练，具体见 1.1《代码片段》

1.1 代码片段

```python
# 模型训练
import pandas as pd
import numpy as np
import xgboost as xgb

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

df_X_y_train_dm = xgb.DMatrix(df_train_X, label=df_train_y)
df_X_test_dm = xgb.DMatrix(df_test_X)

# XGBoost，二分类，梯度提升树（100颗，深度5），交叉熵损失
params = {
    'booster': 'gbtree',
    'objective': 'binary:logistic',
    'eta': 0.1,
    'max_depth':5, 'min_child_weight': 1,
    'gamma': 0.1,
    'subsample': 0.8, 'colsample_bytree': 0.8,
    'alpha': 0.01, 'lambda': 0.01,
}

clf_xgb = xgb.train(params, df_X_y_train_dm, num_boost_round=100)
clf_xgb.save_model('xgb.model')

# 测试集预测，AUC评价
clf_xgb = xgb.Booster(model_file='xgb.model')
pred_prob = clf_xgb.predict(df_X_test_dm)

ras = roc_auc_score(df_test_y, pred_prob)
```

2. 模型优化

模型训练完成后，由于各种原因，效果很可能没达到预期或还有提升空间。除了更精细化的对数据进行《数据集处理》或《特征工程》方向的再次优化外，对模型学习过程的优化也至关重要，因此这里主要介绍基于超参数调整的 XGBoost 模型优化策略

XGBoost 中主要的超参数：

参数	思想	备注
Booster	基学习器，模型每轮迭代中使用的模型 包括：gbtree、gblinear、DART	1. 通用参数 2. 默认使用 gbtree，算法原始理论推导中的树模型构建方法 3. gblinear 为 GBM 框架下的线性模型构建方法 4. DART 为带 dropout 的 gbtree
num_boost_round	Boosting 迭代轮次，对应实际构建的基学习器数量	1. 通用参数 2. 默认 10，一般结合 early stopping 来确定
objective	学习目标，主要对应最小化的损失函数： 1. 回归任务： a) reg:absoluteerror，绝对值损失 $loss(y, f(x)) = \lvert y - f(x) \rvert$ b) reg:squarederror，平方损失 $loss(y, f(x)) = (y - f(x))^2$ c) reg:tweedie，tweedie 分布损失 $loss(y, f(x)) = -y\frac{f(x)^{1-\rho}}{1-\rho} + \frac{f(x)^{2-\rho}}{2-\rho}$ d) … 2. 分类任务： a) binary:logistic，二分类，交叉熵损失 $loss(y, f(x)) = -yln(f(x)) - (1-y)\ln(1-f(x))$ b) multi:softprob，多分类，多个二分类模型（one-vs-rest）经 softmax 转换后的交叉熵损失 $loss(y, f(x)) = -\sum_{i=1}^{k} y_i log \frac{e^{f_i(x)}}{\sum_{j=1}^{k} e^{f_i(x)}}$ c) … 3. 排序任务： a) rank:pairwise，ranknet 损失 $loss(y, f(x)) = -P_{i,j}\ln(\hat{P}_{i,j}) - (1-P_{i,j})\ln(1-\ln(\hat{P}_{i,j}))$ $\hat{P}_{i,j} = \frac{1}{1+e^{-(f(x_i)-f(x_j))}}$ b) ... 4. …	1. 学习目标参数 2. 1.c 中$f(x)$表示 tweedie 分布的均值μ；方差ρ取值范围为(1,2)，为选用该学习目标下的超参数（表示泊松和伽马的混合分布）；适用于长尾和 0 值较多的数据分布情况，如金融信贷等 3. 2.b 中输出为样本在各类别下的预测概率；可以使用 multi:softmax 输出预测类别 4. 可通过设定「obj」参数使用自定义的损失函数，如：focalloss（需要输出一阶、二阶导数）

eta	学习率，控制每轮迭代中权重 w 的占比	1. Booster 参数（gbtree） 2. 范围(0,1]，默认 0.3，经验值一般取[0.01,0.2] 3. 较小的取值可以提升模型泛化性
max_depth	最大树深，控制每轮迭代中树生成的深度	1. Booster 参数（gbtree） 2. 预剪枝参数 3. 默认 6，经验值一般取[3,10] 4. 较小的值对应的树更简单（深度小），可以避免过拟合
min_child_weight	最小叶子节点样本权重和（二阶导数），控制每轮迭代中树生成时的节点分裂 若节点分裂后存在子节点的样本权重和小于该值，则不进行此次分裂	1. Booster 参数（gbtree） 2. 预剪枝参数 3. 默认 1 4. 较大的值对应的树更简单（不易分裂），可以避免过拟合
gamma	节点分裂所需的最小损失减少量，控制每轮迭代中树生成时的节点分裂 对应树的复杂度定义中的γ，即只有$Gain > 0$的情况才会进行分裂	1. Booster 参数（gbtree） 2. 预剪枝参数 3. 默认 0 4. 较大的值对应的树更简单（不易分裂），可以避免过拟合
subsample	行抽样比例，控制每轮迭代中所需的样本数量	1. Booster 参数（gbtree） 2. 范围(0,1]，默认 1，经验值一般取[0.5,1] 3. 较小的值可避免过拟合 4. 可通过设定「sampling_method」参数指定抽样方法： a) uniform，均匀，默认 b) gradient_base，基于梯度，$\sqrt{g^2+\lambda h^2}$，需要设置「tree_method」参数为 gpu_hist
colsample_bytree	列抽样比例，控制每轮迭代中所需的特征数量	1. Booster 参数（gbtree） 2. 范围(0,1]，默认 1，经验值一般取[0.5,1] 3. 较小的值可避免过拟合 4. 类似的列抽样参数有： a) 「colsample_bylevel」，层 b) 「colsample_bynode」，节点 c) 以上「colsample_by*」参数的作用是累积的 5. 可通过设置 DMatrix 的「feature_weights」参数控制抽样权重
lambda	权重 w 的 L2 正则化项 对应树的复杂度定义中的λ	1. Booster 参数（gbtree） 2. 默认 1

		3. 值越大权重 w 越小，避免过拟合
alpha	权重 w 的 L1 正则化项 带 L1 正则化项的树的复杂度定义，$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 + \frac{1}{2}\alpha\sum_{j=1}^{T}\lvert w_j\rvert$ 对应上式中的α	1. Booster 参数（gbtree） 2. 默认 0 3. 值越大权重 w 越稀疏，避免过拟合

我们主要对上述的 Booster 参数和 num_boost_round 进行优化调整，具体调参策略：

方法	思想	备注
Grid Search	网格搜索，对每个超参数的预期取值范围进行自定义网格划分，遍历所有取值组合，训练模型并记录最优超参数	1. 计算复杂度高，$o(n^m)$ 2. 一般依赖经验或使用等间距划分 3. 衍生的优化方法： a) 贪心策略：每次只对部分超参数进行网格搜索，获得最优超参数后固定这部分超参数的取值 b) 随机搜索：对每个超参数的预期取值范围使用随机数进行采样；需要设定搜索次数；相比网格搜索可能获得更好的效果
贝叶斯优化	通过建立超参数与模型输出之间的函数关系，利用每轮迭代中搜索点（超参数组合）的信息确定下一个搜索点（加入到下一轮迭代中），以此来寻找最优超参数 Step 1. 采样多组超参数及对应的模型输出，组成初始数据集$D = \{(x_1, f(x_1)), \dots, (x_n, f(x_n))\}$ Step 2. 迭代确定下一个超参数组合 Step 2.1 对D使用预先假设的模型（函数关系）进行拟合 $$p(y\lvert x, D) = fit(M, D)$$ Step 2.2 利用采集函数计算“最优超参数”x_i $$x_i \leftarrow \underset{x\in\chi}{argmax}\, S(x, p(y\lvert x, D))$$ Step 2.3 计算x_i对应的$f(x_i)$，并加入到D中 Step 3. D中$f(x)$最优的超参数组合x即为所求	1. 计算复杂度较低，依赖迭代次数 2. 依靠概率分布迭代搜索“可能的更好的”超参数 3. 函数关系一般假设D满足高斯分布，使用高斯过程回归进行拟合 4. 采集函数包括：PI、EI、UCB 等 5. 存在冷启动问题

2.1 代码片段

```
# 模型优化
import pandas as pd
import numpy as np
import xgboost as xgb

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, log_loss


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)
df_train_train, df_train_eval = train_test_split(df_train, test_size=0.2, random_state=2024)

df_train_X, df_train_y = df_train_train.iloc[:, 1:], df_train_train.iloc[:, 0]
df_eval_X, df_eval_y = df_train_eval.iloc[:, 1:], df_train_eval.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

df_X_y_train_dm = xgb.DMatrix(df_train_X, label=df_train_y)
df_X_y_eval_dm = xgb.DMatrix(df_eval_X, label=df_eval_y)
df_X_eval_dm = xgb.DMatrix(df_eval_X)
df_X_test_dm = xgb.DMatrix(df_test_X)

# XGBoost，二分类，梯度提升树，Grid Search（贪心策略）
best_loss = np.inf
best_params = {
    'booster': 'gbtree',
    'objective': 'binary:logistic'
}
best_num_boost_round = 0

params = {
    'booster': 'gbtree',
    'objective': 'binary:logistic',
    'eta': 0.1,
    'max_depth':5, 'min_child_weight': 1,
    'gamma': 0.1,
    'subsample': 0.8, 'colsample_bytree': 0.8,
    'alpha': 0.01, 'lambda': 0.01,
    'eval_metric': ['auc', 'logloss']
}
list_watch = [(df_X_y_train_dm, 'train'), (df_X_y_eval_dm, 'eval')]
```

```
# num_boost_round（使用early stopping确定）、eta、max_depth、min_child_weight、gamma
for e in [i/100.0 for i in range(1, 21, 5)]:
    for md in range(3, 6):
        for mcw in range(1, 6):
            for g in [i/100.0 for i in range(0, 110, 10)]:
                params['eta'] = e
                params['max_depth'] = md
                params['min_child_weight'] = mcw
                params['gamma'] = g
                dict_eval = {}
                clf_xgb = xgb.train(params, df_X_y_train_dm,
                                    num_boost_round=10000, # 迭代轮次上限
                                    evals=list_watch,
                                    early_stopping_rounds=100, # 早停监听轮次
                                    evals_result=dict_eval)
                loss = pd.DataFrame(dict_eval['eval']).loc[clf_xgb.best_iteration+1, 'logloss']
                if loss < best_loss:
                    best_loss = loss
                    best_num_boost_round = clf_xgb.best_iteration + 1
                    best_params['eta'] = e
                    best_params['max_depth'] = md
                    best_params['min_child_weight'] = mcw
                    best_params['gamma'] = g

params['eta'] = best_params['eta']
params['max_depth'] = best_params['max_depth']
params['min_child_weight'] = best_params['min_child_weight']
params['gamma'] = best_params['gamma']

# subsample、colsample_bytree
for ss in [i/10.0 for i in range(5, 11)]:
    for csb in [i/10.0 for i in range(5, 11)]:
        params['subsample'] = ss
        params['colsample_bytree'] = csb
        clf_xgb = xgb.train(params, df_X_y_train_dm, num_boost_round=best_num_boost_round)
        loss = log_loss(df_eval_y, clf_xgb.predict(df_X_eval_dm))
        if loss < best_loss:
            best_loss = loss
            best_params['subsample'] = ss
            best_params['colsample_bytree'] = csb

params['subsample'] = best_params['subsample']
params['colsample_bytree'] = best_params['colsample_bytree']
```

```
# alpha、lambda
for a in [0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]:
    for l in [0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]:
        params['alpha'] = a
        params['lambda'] = l
        clf_xgb = xgb.train(params, df_X_y_train_dm, num_boost_round=best_num_boost_round)
        loss = log_loss(df_eval_y, clf_xgb.predict(df_X_eval_dm))
        if loss < best_loss:
            best_loss = loss
            best_params['alpha'] = a
            best_params['lambda'] = l

# 最优超参数训练
clf_xgb = xgb.train(best_params, df_X_y_train_dm, num_boost_round=best_num_boost_round)
clf_xgb.save_model('xgb.model')

# 测试集预测，AUC评价
clf_xgb = xgb.Booster(model_file='xgb.model')
pred_prob = clf_xgb.predict(df_X_test_dm)

ras = roc_auc_score(df_test_y, pred_prob)
```

四、模型评价

模型构建完成后，我们需要对模型进行合理的评估，以帮助我们准确的衡量模型的性能，从而进行优化和改进。不同场景（任务）下的评价方法（指标）往往是不同的

1. 分类任务

二分类混淆矩阵		预测值	
		正样本（Positive）	负样本（Negative）
真实值	正样本（Positive）	TP（True Positive）	FN（False Negative）
	负样本（Negative）	FP（False Positive）	TN（True Negative）

指标	思想	备注
Accuracy	准确率，正确分类的样本占整体样本的比重 $(TP+TN)/(TP+FP+FN+TN)$	1. 受分类阈值影响（样本不平衡） 2. 表示整体样本效果
Precision	精确率，预测为正的样本中分类正确的比例 $TP/(TP+FP)$	1. 受分类阈值影响（样本不平衡） 2. 对于二分类，表示正样本 3. 对于多分类，表示某一类样本；表示整体时一般使用加权平均（各类样本数量）
Recall	召回率，也称灵敏度（TPR），实际为正的样本中分类正确的比例 $TP/(TP+FN)$	1. 受分类阈值影响（样本不平衡） 2. 对于二分类，表示正样本 3. 对于多分类，表示某一类样本；表示整体时一般使用加权平均（各类样本数量）
$F_1\ score$	精确率和召回率的调和平均 $2*\frac{Precision*Recall}{Precision+Recall}$	1. 受分类阈值影响（样本不平衡） 2. 综合评价方法，解决评估过程中精确率与召回率存在的冲突问题 3. 一般化定义，$F_\beta\ score$（β为召回率相对于精确率的权重）： $(1+\beta^2)*\frac{Precision*Recall}{\beta^2*Precision+Recall}$ 4. 对于多分类，表示某一类样本；表示整体时一般使用加权平均（各类样本数量）
AUC	Area Under Curve（ROC 曲线下的面积），表示模型将某个随机正样本排列在某个随机负样本前的概率（正样本预测得分大于负样本预测得分） ROC 曲线（Receiver Operating Characteristic，受试者曲线），通过不同阈值下计算的 TPR（纵坐标）和 FPR（1-特异度，横坐标）绘制得出	1. 一般用于二分类 2. 对样本不平衡不敏感 3. 特异度：负样本召回，$TN/(FP+TN)$；即 FPR，$FP/(FP+TN)$ 4. ROC 曲线越靠近左上角，AUC 值越大，模型性能越好 5. 其他计算方法：对预测得分降序排序，统计正负样本对中，正样

<table>
<tr><td></td><td></td><td>本预测得分大于负样本预测得分的组合数占比
$\frac{\sum_{i\in Pos}^{M} Rank_i - M*(M+1)/2}{M*N}$</td></tr>
<tr><td>KS</td><td>Kolmogorov-Smirnov 统计量, KS 曲线的最大差值，通过衡量正负样本的累积分布差异, 评估模型的区隔力
KS 曲线，通过不同阈值（横坐标）下计算的 TPR、FPR（纵坐标）绘制得出
$max\ |TPR - FPR|$
<table><tr><td>KS 值范围</td><td>区隔能力</td></tr><tr><td>(0,0.2)</td><td>无</td></tr><tr><td>[0.2,0.4]</td><td>较好</td></tr><tr><td>(0.4,0.5]</td><td>好</td></tr><tr><td>(0.5,0.6]</td><td>很好</td></tr><tr><td>(0.6,0.75]</td><td>非常好</td></tr><tr><td>(0.75,1]</td><td>异常</td></tr></table></td><td>1. 一般用于二分类
2. 对样本不平衡不敏感
3. 适用于寻找最优切分阈值
4. 一般用于风险模型</td></tr>
</table>

2. 回归任务

指标	思想	备注
MAE	Mean Absolute Error，平均绝对误差 $\frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$	1. 越接近 0，模型拟合程度越好 2. L1 范数损失 3. 与目标真实值量级一致 4. 相比「MSE」、「RMSE」，受异常值的影响小
MAPE	Mean Absolute Percentage Error，平均绝对百分比误差 $\frac{1}{N}\sum_{i=1}^{N}\lvert\frac{y_i - \hat{y}_i}{y_i}\rvert$	1. 越接近 0，模型拟合程度越好；大于 1 表示“劣质模型” 2. 考虑了残差在目标真实值中的占比，进一步弱化了异常值的影响 3. 不适用于目标真实值有 0 的情况 4. 存在目标真实值较小时导致计算结果过大的问题，修正指标「SMAPE」: $\frac{1}{N}\sum_{i=1}^{N}\lvert\frac{y_i - \hat{y}_i}{(\lvert y_i\rvert + \lvert\hat{y}_i\rvert)/2}\rvert$
MSE	Mean Squared Error，均方误差 $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	1. 越接近 0，模型拟合程度越好 2. L2 范数损失 3. 放大了预测误差，易受异常值的影响 4. 相比「MAE」，由于可微分特性，被用作回归模型默认损失函数
RMSE	Root Mean Squared Error，均方根误差	1. 越接近 0，模型拟合程度越好

	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}$	2. 与目标真实值量级一致，因此相比「MSE」使用更广泛 3. 放大了预测误差，易受异常值的影响
R^2	可决系数，已解释变差（SSR，回归平方和）在总变差（SST，总平方和）中的占比，用于评估拟合回归线周围数据点的散布情况 总变差由已解释变差和未解释变差（SSE，误差平方和）组成 $\frac{SSR}{SST}=1-\frac{SSE}{SST}$ $SSR=\sum_{i=1}^{N}(\hat{y}_i-\bar{y}_i)^2$ $SSE=\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$ $SST=\sum_{i=1}^{N}(y_i-\bar{y}_i)^2$	1. 越接近 1，模型拟合程度越好 2. 适用于线性回归 3. 表示自变量对因变量的解释程度 4. 可用于计算 VIF，过滤冗余特征 5. 存在自变量增加导致R^2增大的问题，使用「调整后R^2」： $\frac{SSR/k}{SST/(n-k-1)}$

3. 代码片段

```python
# 模型评价
import pandas as pd
import numpy as np
import xgboost as xgb

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, classification_report, root_mean_squared_error, r2_score


df = pd.read_csv('xxx.txt', sep='\t', encoding='utf-8')
df_train, df_test = train_test_split(df, test_size=0.2, random_state=2024)

df_train_X, df_train_y = df_train.iloc[:, 1:], df_train.iloc[:, 0]
df_test_X, df_test_y = df_test.iloc[:, 1:], df_test.iloc[:, 0]

df_X_y_train_dm = xgb.DMatrix(df_train_X, label=df_train_y)
df_X_test_dm = xgb.DMatrix(df_test_X)

# 分类任务，XGBoost
params = {
    'booster': 'gbtree',
    'objective': 'binary:logistic',
    'eta': 0.1,
    'max_depth':5, 'min_child_weight': 1,
    'gamma': 0.1,
    'subsample': 0.8, 'colsample_bytree': 0.8,
    'alpha': 0.01, 'lambda': 0.01,
}

clf_xgb = xgb.train(params, df_X_y_train_dm, num_boost_round=100)
clf_xgb.save_model('xgb.model')

clf_xgb = xgb.Booster(model_file='xgb.model')
pred_prob = clf_xgb.predict(df_X_test_dm)
```

```python
# Precision、Recall、F1 score，多阈值（预测打分）评价
df_y = pd.DataFrame({'y_true': df_test_y, 'y_pred_prob': pred_prob})
df_y.sort_values(by='y_pred_prob', ascending=False, inplace=True)
df_y.reset_index(drop=True, inplace=True)

list_df_prf = []
for t in [i/100.0 for i in range(1, 51)]:
    test_p = np.array([int(x) for x in df_y['y_pred_prob']>=t])
    cr = classification_report(df_y['y_true'], test_p, output_dict=True)
    c = test_p.sum()
    p = cr['1']['precision']
    r = cr['1']['recall']
    f1 = cr['1']['f1-score']
    df_prf_temp = pd.DataFrame(
        {
            'threshold': [i],
            'count': [c],
            'precision': [p],
            'recall': [r],
            'F1': [f1]
        }
    )
    list_df_prf.append(df_prf_temp)

df_prf = pd.concat(list_df_prf, axis=0)
df_prf.reset_index(drop=True, inplace=True)

# 回归任务，XGBoost
params = {
    'booster': 'gbtree',
    'objective': 'reg:squarederror',
    'eta': 0.1,
    'max_depth':5, 'min_child_weight': 1,
    'gamma': 0.1,
    'subsample': 0.8, 'colsample_bytree': 0.8,
    'alpha': 0.01, 'lambda': 0.01,
}

clf_xgb = xgb.train(params, df_X_y_train_dm, num_boost_round=100)
clf_xgb.save_model('xgb.model')

clf_xgb = xgb.Booster(model_file='xgb.model')
pred_prob = clf_xgb.predict(df_X_test_dm)

# RMSE、R2
rmse = root_mean_squared_error(df_test_y, pred_prob)
r2 = r2_score(df_test_y, pred_prob)
```

五、附录

相关文档&博客

1. 数据集处理——数据集划分
   a) https://www.jianshu.com/p/45aa52002fc8 如何正确使用数据集
   b) https://zhuanlan.zhihu.com/p/33651227 数据集划分与特征工程的先后关系
   c) https://zhuanlan.zhihu.com/p/488284413 交叉验证
2. 数据集处理——数据集采样
   a) https://imbalanced-learn.org/stable/index.html 不平衡样本学习文档
   b) https://www.cnblogs.com/massquantity/p/9382710.html 不平衡样本采样方法
   c) https://zhuanlan.zhihu.com/p/237792038 EasyEnsemble
3. 特征工程——特征预处理
   a) https://www.cnblogs.com/charlotte77/p/5622325.html 特征预处理
   b) https://scikit-learn.org/stable/modules/preprocessing.html sklearn 文档
4. 特征工程——特征预处理——无量纲化
   a) https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html sklearn 文档
   b) https://zhuanlan.zhihu.com/p/454711078 数据集预处理——数据缩放
5. 特征工程——特征预处理——离散型特征处理
   a) https://www.biaodianfu.com/categorical-feature.html 类别特征处理
   b) https://zhuanlan.zhihu.com/p/26308272 目标编码
   c) https://zhuanlan.zhihu.com/p/30026040 WOE 编码
6. 特征工程——特征预处理——缺失值处理
   a) https://scikit-learn.org/stable/modules/impute.html sklearn 文档
   b) https://www.zhihu.com/question/26639110 机器学习中如何处理缺失数据
   c) https://zhuanlan.zhihu.com/p/579468047 R 语言——多重填充
7. 特征工程——特征预处理——异常值处理
   a) https://www.jianshu.com/p/8e5291cc74e7 异常值检测
   b) https://scikit-learn.org/stable/modules/outlier_detection.html 异常值检测 sklearn 文档
   c) https://zhuanlan.zhihu.com/p/84587517 HBOS
   d) https://blog.csdn.net/itplus/article/details/38926837 FSDP
   e) https://zhuanlan.zhihu.com/p/32784067 One-Class SVM
   f) https://blog.csdn.net/extremebingo/article/details/80108247 孤立森林
   g) https://zhuanlan.zhihu.com/p/358944859 异常值处理常用方法
8. 特征工程——特征预处理——特征转换
   a) https://scikit-learn.org/stable/auto_examples/preprocessing/plot_map_data_to_normal.html 非线性转换 sklearn 文档
   b) https://www.lianxh.cn/news/581607c3114c9.html 正态转换方法
   c) https://blog.csdn.net/weixin_46649052/article/details/115117950 为什么要正态分布
   d) https://zhuanlan.zhihu.com/p/442674762 线性回归因变量正态分布
   e) https://scikit-learn.org/stable/auto_examples/preprocessing/plot_discretization_strategies.html 分箱

转换 sklearn 文档

f) https://blog.csdn.net/u013421629/article/details/78416748 卡方分箱

9. 特征工程——特征选择
   a) https://scikit-learn.org/stable/modules/feature_selection.html sklearn 文档
   b) https://www.jianshu.com/p/867193608bbd 特征维度灾难
10. 特征工程——特征选择——Filter
   a) https://zhuanlan.zhihu.com/p/33357167 方差分析
   b) https://zhuanlan.zhihu.com/p/139151375 F 检验
   c) https://zhuanlan.zhihu.com/p/140043959 卡方检验
   d) https://zhuanlan.zhihu.com/p/80134853 WOE 和 IV 值
   e) https://zhuanlan.zhihu.com/p/676392044 皮尔逊、斯皮尔曼相关系数
   f) https://zhuanlan.zhihu.com/p/658467717 肯德尔相关系数
   g) https://zhuanlan.zhihu.com/p/53092905 MIC
11. 特征工程——特征选择——Wrapper
   a) https://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html 递归特征消除 sklearn 文档
   b) https://zhuanlan.zhihu.com/p/547264476 逐步回归
   c) https://www.cnblogs.com/wqbin/p/11109650.html VIF
   d) https://zhuanlan.zhihu.com/p/142489599 AIC、BIC
12. 特征工程——特征选择——Embedded
   a) https://zhuanlan.zhihu.com/p/29360425 L1、L2
   b) https://blog.csdn.net/sujinhehehe/article/details/84201415 XGBoost 特征重要性
13. 特征工程——特征选择——其他方法
   a) https://blog.csdn.net/weixin_38037405/article/details/127131599 PI
14. 特征工程——降维
   a) https://zhuanlan.zhihu.com/p/77151308 PCA
   b) https://www.cnblogs.com/pinard/p/6244265.html LDA
   c) https://zhuanlan.zhihu.com/p/133207206 AutoEncoder
   d) https://scikit-learn.org/stable/modules/decomposition.html 信号成分分解 sklearn 文档
15. 特征工程——特征组合
   a) https://supervised.mljar.com AutoML 文档
   b) https://scikit-learn.org/stable/auto_examples/linear_model/plot_polynomial_interpolation.html 多项式生成、B 样条曲线生成 sklearn 文档
   c) https://www.jianshu.com/p/96173f2c2fb4 GBDT+LR
   d) https://tech.meituan.com/2016/03/03/deep-understanding-of-ffm-principles-and-practices.html FM
16. 模型构建——模型训练
   a) https://xgboost.readthedocs.io/en/latest XGBoost 文档
   b) https://blog.csdn.net/a819825294/article/details/51206410 XGBoost 原理与推导
17. 模型构建——模型优化
   a) https://zhuanlan.zhihu.com/p/115879247 gbtree 和 gblinear 的区别
   b) https://blog.csdn.net/uncle_ll/article/details/136080064 DART

c) https://zhuanlan.zhihu.com/p/629883594 tweedie 损失
d) https://zhuanlan.zhihu.com/p/91652813 GBDT 多分类
e) https://zhuanlan.zhihu.com/p/631575097 pairwise——ranknet
f) https://www.cnblogs.com/marsggbo/p/9866764.html 贝叶斯优化

18. 模型评价
    a) https://zhuanlan.zhihu.com/p/666774052 模型评价指标
19. 模型评价——分类任务
    a) https://www.zhihu.com/question/30643044/answer/48955833 评价指标各自优缺点
    b) https://www.cnblogs.com/gczr/p/10354646.html KS 深入理解
20. 模型评价——回归任务
    a) https://www.cnblogs.com/wqbin/p/11109650.html 可决系数