中国研究生创新实践系列大赛
“华为杯”第十八届中国研究生数学建模竞赛

学　　校	电子科技大学
参赛队号	21106140097
队员姓名	1.杨松洁
	2.梁雯茜
	3.徐乐薇

中国研究生创新实践系列大赛
“华为杯”第十八届中国研究生
数学建模竞赛

题 目　　抗乳腺癌候选药物的优化建模

摘　　要：

乳腺癌常被称为“粉红杀手”，其发病率位居女性恶性肿瘤的首位，并且它是目前世界上最常见，致死率较高的癌症之一。因此，对于如何治疗乳腺癌是一个极具意义的问题。有研究发现，雌激素受体α亚型（Estrogen receptors alpha, ERα）在不超过 10%的正常乳腺上皮细胞中表达，但大约在 50%-80%的乳腺肿瘤细胞中表达；而对 ERα基因缺失小鼠的实验结果表明，ERα确实在乳腺发育过程中扮演了十分重要的角色。因此，ERα被认为是治疗乳腺癌的重要靶标，能够拮抗 ERα活性的化合物可能是治疗乳腺癌的候选药物。

本文即是基于此背景进行相关问题的研究，其问题可简述为四个方面：挑选对 ERα拮抗剂的生物活性影响重要的分子描述符、建立预测模型定量预测 ERα生物活性、根据 ADMET 性质构建分类预测模型以及选择最优的分子描述符决策变量以及它们的取值或取值范围。

针对第一个方面的问题，我们首先通过规范化、方差过滤等方法对数据进行预处理，将 729 个分子描述符特征降维到 505 个，接着通过构建的集成评分降维法 RSDRM 选出得分位于各基学习器前 100 名的特征加权打分，按频次及总得分进行排名。经过第一轮相关性筛选后得到 26 个特征，之后对于这 26 个特征进行第二轮独立性筛选，筛除其中与其他特征高度相关（相关性>0.9）的 6 个特征，最终确定加权得分最高的 20 个分子描述符特征。

针对第二个方面的问题，我们选取了 XGBoost、LightGBM、随机森林等学习器，以 pIC_{50} 为目标函数构造回归预测模型（IC50 的值只需根据负对数关系即可推导出），并在后续采取了贝叶斯优化结合模型线性融合的方式进行对比。我们发现，模型线性融合后的预测效果更好，在大多数预测中，效果都略优于或等于效果最佳的学习器，RMSE 可达到 0.089。同时我们给出了各预测模型的具体超参数值。

针对第三个方面的问题，我们考虑了多模型单标签分类和单模型多标签分类两种方案对 ADMET 五种性质进行分类预测。其中多模型单标签分类表示我们采用多个模型分别对 ADMET 五种性质进行分类预测，它们之间独立进行并无相关性。对于这一类方案，我们采用 LightGBM 分类方法构建五个独立的模型。此种方案的优劣性非常明显，即方便处理

但是独立预测无法挖掘标签之间的关联性，可能会有性能损失或泛化性不强的问题。第二种方案是单模型多标签分类，这意味着我们只需建立一个模型来预测 ADMET 五种标签性质的分类，这种方案考虑到了标签之间的关联性，并且可以通过挖掘关联性信息辅助模型分类预测的准确度。因此，为了佐证该题中 ADMET 五个性质之间的关联性，我们采用一种专门针对二进制分类变量的相关性分析方法——克莱姆相关性分析对于 ADMET 五种性质进行了相关性分析，并且从网上查询了关于 ADMET 预测的相关资料，最后证明了这五种性质之间的确具有相关性。所以我们采用一种在各个数据集都表现得很出色的一个多标签分类方法——MPVAE 进行 ADMET 性质的分类，并且阐述了 MPVAE 用于我们的问题的合理性。然后我们对比了基于 MPVAE 的单模型多标签分类法和基于 LightGBM 的多模型单标签分类法，并以 AUC 为评估标准，其显示单模型多标签方法具有独特的优点。此外，在这个问题中，我们同样对变量进行了降维处理，大致按照了问题一中的特征降维过程。

针对第四个方面的问题，我们首先进行决策变量的选择优化，然后对选择后的决策变量进行取值优化。首先，我们结合考虑了问题 2 和问题 3 中的决策变量选择的过程，并以此来选出既与生物活性相关性较高又与 ADMET 性质相关性较高的决策变量，然后基于这些决策变量以最大化生物活性为目标，且满足 ADMET 的 3 个性质较好的非线性约束以及决策变量的取值约束。所构建的优化问题主要包含两个难点，一是非线性约束，二是决策变量中既有整数型变量又有连续型变量，使优化问题变成了混合整数规划问题。为了解决所构建的优化问题，我们先采用了惩罚函数法处理非线性约束，即将非线性约束转化为惩罚项，合并于目标函数。最后，我们采用了一种智能优化算法——麻雀搜索处理此简化后的混合整数规划问题。

关键词：集成评分降维法、线性融合预测、多标签分类、麻雀搜索、混合整数规划。

目录

一、 问题重述 6

1.1 问题背景 6

1.2 拟解决的问题 6

二、 模型的假设 7

三、 符号说明 7

四、 问题分析 8

4.1 问题 1 分析 8

4.2 问题 2 分析 9

4.3 问题 3 分析 9

4.4 问题 4 分析 9

五、 问题 1：基于生物活性的分子描述符变量筛选 10

5.1 数据预处理 10

5.2 集成评分降维法 11

5.2.1 包装法 12

5.2.2 过滤法 13

5.3 相关性降维 14

5.4 独立性降维 18

六、 问题 2：生物活性的定量预测 20

6.1 线性融合预测模型 20

6.1.1 随机森林 21

6.1.2 XGBoost 21

6.1.3 LightGBM 22

6.2 模型评估 22

6.2.1 基于贝叶斯优化的模型调参 24

七、 问题 3：ADMET 性质的分类预测 25

7.1 决策变量选择 26

7.2 基于 LightGBM 的单标签分类 26

7.3 多标签分类 27

7.3.1 多标签相关性分析 27

7.3.2 多元概率比变分自编码器 28

7.4 分类结果分析 30

八、 问题 4：分子描述符优化 30
8.1 决策变量选择 31
8.2 变量取值优化 32
8.2.1 外点法 33
8.2.2 麻雀搜索 34
8.3 优化结果分析 35
8.3.1 参数说明 35
8.3.2 优化结果 38
九、 模型的评价和推广 38
9.1 模型评价 38
9.1.1 模型优点 38
9.1.2 模型缺点 39
9.2 模型推广 39
附录 A 主要源代码 41

一、 问题重述

1.1 问题背景

乳腺癌是目前最常见且致死率较高的癌症之一，ERα 被认为是治疗乳腺癌的重要靶标，因此能够拮抗 ERα 活性的化合物可能对于治疗乳腺癌效果显著。重点是找到具有良好生物活性和 ADMET 特性的活性化合物作为候选药物。

本题中，生物活性方面选取 IC_{50} 和 pIC_{50} 两个指标表示化合物对 ERα 的生物活性值。除具备良好的生物活性外，还需要在人体内具备良好的药代动力学性质和安全性，即 ADMET（Absorption 吸收、Distribution 分布、Metabolism 代谢、Excretion 排泄、Toxicity 毒性）性质。如果化合物活性再好，而 ADMET 性质不佳，那么依然不能成为药物，所以需要进行 ADMET 性质优化。

本试题仅考虑化合物的 5 种 ADMET 性质：1）小肠上皮细胞渗透性（Caco-2），度量化合物被人体吸收的能力；2）细胞色素 P450 酶 3A4 亚型（CYP3A4），可度量化合物的代谢稳定性；3）化合物心脏安全性评价（hERG），度量化合物的心脏毒性；4）人体口服生物利用度（HOB）, 可度量药物进入人体后被吸收进入人体血液循环的药量比例；5）微核试验（MN），是检测化合物是否具有遗传毒性的一种方法。

现有 1974 个化合物的 729 个分子描述符信息，分子描述符与 ERα 的生物活性相关数据以及 5 种 ADMET 性质的相关数据，找到合适的药物需要综合考虑生物活性以及 ADMET 性质。现通过数据挖掘技术来解决制药过程建模问题。

1.2 拟解决的问题

在针对 ERα 靶标的药物研发中，需要构建化合物生物活性的定量预测模型和 ADMET 性质的分类预测模型，进而通过预测来优化 ERα 拮抗剂的生物活性和 ADMET 性质。

化合物生物活性方面通常采用建立化合物活性预测模型的方法来筛选潜在活性化合物，其中自变量为化合物的 729 个分子描述符信息，用于描述化合物的结构和性质特征的参数，因变量为 IC_{50} 和 pIC_{50} 两个指标。ADMET 性质方面的分类预测模型，自变量为 729 个分子描述符信息，因变量为五种 ADMET 性质。

请根据以上背景以及所提供数据分别完成以下四个问题：

1. 建立模型，对于 1974 个化合物的 729 个分子描述符进行变量选择，根据变量对生物活性影响的重要性进行排序，筛选对生物活性最具有显著影响的 20 个分子描述符，并描述筛选的过程及其合理性。
2. 结合 1 中筛选的 20 个分子描述符变量，构建化合物对 ERα 生物活性的定量预测模型，要求叙述建模过程及其合理性。然后通过构建的预测模型，对“ERα_activity.xlsx”的 test 表中的 50 个化合物进行 IC50 值和对应的 pIC50 值预测，并填入对应位置。

3. 利用 729 个分子描述符变量，对于 ADMET 性质分别针对 Caco-2、CYP3A4、hERG、HOB、MN 五种性质构建分类预测模型，要求叙述建模过程及其合理性。然后通过构建的 5 个分类预测模型，对文件“ADMET.xlsx”的 test 表中的 50 个化合物进行 ADMET 的五种性质预测，并填入表格对应位置。
4. 在 729 个分子描述符中确定哪些分子描述符使得化合物具有更好生物活性，并确定其取值或相应取值范围，能够使至少有三个 ADMET 性质同时较好。

二、 模型的假设

- 假设样本不存在异常值，全为准确实验所生成的数据
- 假设 ADMET 性质中：A 较优情况为渗透性好（值为 1）;D 较优情况为生物利用性好（值为 1）;M 较优情况为化合物稳定性好，即被 CYP3A4 代谢差（值为 0）;E 较优情况为无遗传毒性（值为 0）;T 较优情况为无心脏毒性（值为 0）
-

三、 符号说明

我们将重要的符号进行了说明，如下所示：

符号	意义
ρ	斯皮尔曼相关系数
g_t	相关性分析函数 (t 有四种取值)
$\mathcal{M}_g$	高斯代理模型
j_t	单个预测模型 (t 有三种取值)
$\mathcal{M}_p$	线性融合预测模型
$\mathcal{M}_c$	MPVAE 分类模型
$\mathcal{L}$	EI 采集函数
$\mathcal{D}$	样本空间
L	ADMET 性质个数
N	样本数量
$\mathbf{v}$	问题 4 决策变量个数
$\mathbf{v}^r$	问题 4 连续变量个数
$\mathbf{v}^z$	问题 4 整数变量个数
$\mathbf{h}$	MPVAE 分类模型预测值
E	线性融合预测模型预测值

四、 问题分析

4.1 问题 1 分析

本问要求对于数据集中提供的 1974 个化合物信息的 729 个分子描述符进行变量选择，并根据变量对生物活性影响的重要性进行排序，给出前 20 个对生物活性最具有显著影响的分子描述符。问题一属于数据挖掘中的特征降维部分，本题中无需构建新特征，只需对原有的 729 个特征进行降维。可以先通过数据预处理的方式对特征进行预处理，再通过特征降维的方法，使用多种模型对于特征重要性进行分别打分，并对各特征得分加权，得到第一轮特征筛选结果。为确保最后筛选的特征数大于 20，第一轮可以适当放宽数量约束，之后对于所得特征进行第二轮独立性分析，通过对自变量的相关性系数设定阈值的方式，筛除与剩下特征高度相关的自变量，最终确定加权得分最高的 20 个分子描述符。

4.2 问题 2 分析

问题二属于数据挖掘中的预测模型构建部分，通过问题一中降维得到的 20 个分子描述符，构建自变量为 20 个分子描述符、因变量为 IC_{50} 和 pIC_{50} 的回归预测模型，并对于 test 表中的 50 个化合物信息进行 IC_{50} 和 pIC_{50} 的预测，填入相关表中以检验本题模型的准确性。问题二中，由于因变量 IC_{50} 和 pIC_{50} 都可表示化合物对生物活性的影响，且两者之间有系数为 10^{-9} 的负对数关系，因此只需要选择其一作为因变量进行回归预测模型的构建，另一变量通过负对数关系推算即可确定预测值。

4.3 问题 3 分析

问题 3 要求构建分类模型对于 ADMET 的 5 种特性 Caco-2、CYP3A4、hERG、HOB、MN 分别进行分类预测。这种分类问题可从两个角度进行求解，即多模型单标签分类和单模型多标签分类。前者的意思是根据多个标签分别构建 1 个分类预测模型，此种方案忽略了各个标签的相关性，泛化性不强，但其优势在于结构简单，处理方便。后者的意思在于构建一个分类模型进行多个标签的分类，因此考虑了多个标签之间的关联性，在实际场景中更好得以应用。我们采用了上述两种方案分别构建了分类模型，并对题目所给的测试集进行标签分类：

- 单标签分类：以问题 1 的特征降维方法分别对 ADMET 的 5 个特性进行决策变量选择，并分别采用 LightGBM 方法构建分类模型，并实现各个标签的分类预测。
- 多标签分类：根据前面所挑选的决策变量，再进一步筛选对 ADMET 五个标签都相关的变量，然后采用多元概率比变分自编码器构建一个模型进行多标签分类。

4.4 问题 4 分析

对于问题 4 所要求的决策变量选择和变量取值优化，我们结合考虑了问题 2 和问题 3 中的决策变量选择的过程，并以此来选出既与生物活性相关性较高又与 ADMET 性质相关性较高的决策变量，然后基于这些决策变量以最大化生物活性为目标，且满足 ADMET 的 3 个性质较好的非线性约束以及决策变量的取值约束。针对这个优化问题，我们首先注意到决策变量中既包含连续性变量又包含整数型变量，因此这是一个混合整数规划问题，一般的优化方法不好求解，再者，此优化问题还需满足一个关于 ADMET 性质的非线性约束。因此，我们考虑先采用惩罚函数法处理非线性约束，即将非线性约束当作惩罚项加入目标函数。然后采用一种智能优化算法——麻雀搜索处理此混合整数规划问题。

五、 问题 1：基于生物活性的分子描述符变量筛选

5.1 数据预处理

文件“Molecular_Descriptor.xlsx”和“ERα_activity.xlsx”提供的数据集中，提供了 1974 个化合物的分子结构式、729 个分子描述符以及生物活性、ADMET5 方面的性质数据。问题一的求解可分为数据预处理及特征降维两大版块，整体问题思路如下图所示：

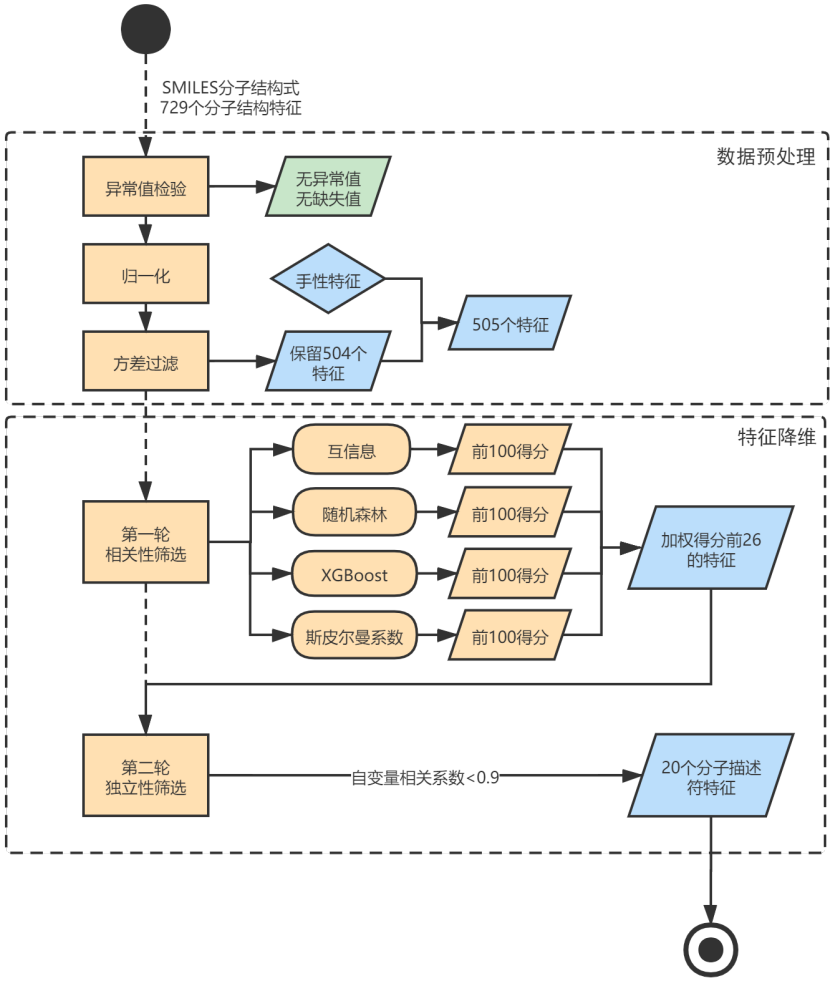

图 **1**　问题一解题流程图

- 第一轮：数据异常值检查经过对数据的检查，文件中无缺失值无异常值，pIC50 和 IC50 符合系数为 10^{-9} 的负对数关系。此外，我们观察到 SMILES 分子式中包含手性特征，即 SMILES 分子式中含有 @ 符号则记有手性特征，反之则无。因此我们将这个 0-1 特征也加入我们的总体原始特征中。
- 第二轮：规范化处理规范化处理也叫离差标准化，是对原始数据的线性变换，使结果落到 [0,1] 区间。规范化处理是考虑数据库尽量没有冗余，考虑数据的完整性和一致性；当不同量纲的数据进行比较时，就需要进行标准化处理来消除量纲的影响。

规范化处理公式：

$$y_i = \frac{x_i - \min_{1<i<n}(x_i)}{\max_{1<i<n}(x_i) - \min_{1<i<n\cap n}(x_i)} \tag{1}$$

- 第三轮：方差过滤在机器学习的数据预处理的过程中常常会使用到过滤法，方差过滤是过滤法之一，所谓的方差过滤即过滤掉那些特征方差较小的特征。如果一个特征本身的方差很小，就表示样本在这个特征上基本没有差异，可能特征中的大多数值都一样，甚至整个特征的取值都相同，那么此特征对于样本区分没有什么作用。所以可以设置一个过滤的阈值，过滤掉方差小的特征，以达到特征筛选的目的。
- 第四轮：特征降维通过 4 种模型融合的方法进行特征降维，本问只需要考虑相关性而无需考虑变量的独立性，即自变量 729 个分子描述符与因变量生物活性 IC50 和 pIC50 的关系。本问由于只在 729 个分子描述符中选择 20 个变量，因此不能采用 PCA 以及因子分析等会产生隐变量的降维方法。

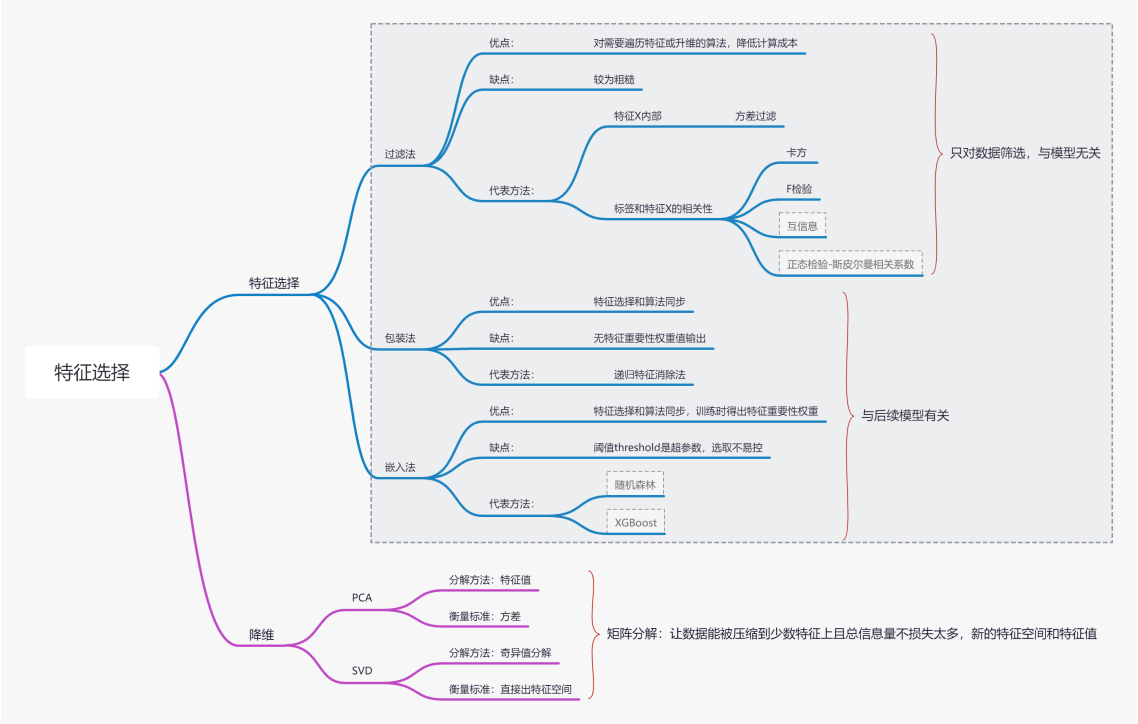

图 **2**　特征筛选方法及对比思路

5.2 集成评分降维法

针对本问，我们自定义了集成评分降维法（RSDRM, Integrated Scoring Dimension Reduction Method），结合嵌入法和过滤法的方法对于方差过滤后的 505 个分子描述符特征 (包含手性特征) 进行重要性评分，各选取得分进入前 100 的特征，具体模型包括嵌入法中的随机森林、XGBoost，以及过滤法中的互信息、正态检验-斯皮尔曼相关系数等共计四种模型。本问不采用包装法的原因在于：包装法可以根据重要性对于特征进行降维

选择，但对每个特征重要性无法量化到得分，无法进行后续加权综合评分。特征筛选方法的选择考虑角度如图 2所示。

5.2.1 包装法

包装法，是一个特征选择和算法训练同时进行的方法，与嵌入法十分相似。它依赖于算法自身的属性选择来完成特征选择。不同的是我们往往使用一个目标函数作为黑盒来帮助我们完成特征选择，而不是我们自己输入某个评估指标或统计量的阈值。

包装法在初始训练集上训练我们的评估器，并且通过 coef_ 属性或 feature_importances_ 属性获得每个特征的重要性，然后从当前的一组特征中删除掉一些最不重要的特征，然后在修改后的集合上重复该过程，直到最后剩下的特征是我们规定的数量。本问中包装法的缺点在于：对于每个特征没有得分，我们需要量化到得分，才能做到后续的加权打分。

随机森林 在降维中我们可以用随机森林进行特征选择。随机森林是一种广泛使用的特征选择算法，它会自动计算各个特征的重要性，所以无需单独编程。擅长选择出较小的特征子集，参考最佳选择特征数为原特征数取根号，故与本题较为匹配。对于 729 个特征的数据集，随机森林是较为适合的降维方法。

随机森林的优点：1）具有极高的准确率；2）随机性的引入，使得随机森林不容易过拟合；3）很好的抗噪声能力（能够更好的处理离群点）；4）能处理很高维度的数据，并且不用做特征选择；5）既能处理离散型数据，也能处理连续性数据，数据集无需规范化；6）训练速度快，可以得到变量重要性排序；7）容易实现并行化。随机森林的缺点：1）当随机森林中的决策树个数很多时，训练需要的空间和时间会很大；2）随机森林的解释性很差。

XGBoost 算法的基本思想与 GBDT 类似，不断地地进行特征分裂来生长一棵树，每一轮学习一棵树，其实就是去拟合上一轮模型的预测值与实际值之间的残差。当我们训练完成得到 k 棵树时，要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数，最后只需将每棵树对应的分数加起来就是该样本的预测值。

XGBoost 是经过优化的分布式梯度提升库，旨在高效、灵活且可移植。在工业界大规模数据方面，XGBoost 的分布式版本有广泛的可移植性，支持在 Kubernetes、Hadoop、SGE、MPI、Dask 等各个分布式环境上运行，使得它可以很好地解决工业界大规模数据的问题。

5.2.2 过滤法

目的：在维持算法表现的前提下，帮助算法们降低计算成本。

特点：独立操作，不与后续分类（或者回归）模型相关。

目标对象：需要遍历特征或升维的算法。最近邻算法 KNN，支持向量机 SVM，决策树，神经网络，回归算法等遍历特征或升维运算，本身的运算量很大，需要的时间很长，因此特征选择很重要。随机森林不需要遍历特征，每次选的特征就很随机，并非用到所有的特征，所以特征选择作用不大。

互信息 可以评价定性自变量对定性因变量的相关性，也可以评价类别型变量对类别型变量的相关性。互信息越大表明两个变量相关性越高，互信息为 0 时，两个变量相互独立。互信息的计算公式为：

$$I(X;Y)=\sum_{x\in X}\sum_{y\in Y}p(x,y)\log\frac{p(x,y)}{p(x)p(y)}=D_{KL}(p(x,y)\|p(x)p(y)) \tag{2}$$

其中，p(x) 和 p(y) 为 X 和 Y 的边际概率分布函数，p(x,y) 为 X 和 Y 的联合概率分布函数。直观上，互信息度量两个随机变量之间共享的信息，也可表示为由于 X 的引入而使 Y 的不确定性减少的量，这时互信息与信息增益相同。皮尔逊系数只能衡量线性相关性，而互信息系数能够很好地度量各种相关性，但计算相对复杂。

正态检验-斯皮尔曼相关系数 在统计学中，正态检验主要用于检验一个数据集是否服从正态分布。常用的 t 检验、方差分析（ANOVA）等参数检验都有一个共同的前提条件：样本数据必须服从正态分布，即样本数据必须来源于一个正态分布的总体，若样本数据不服从正态分布，就不能用以上参数检验对数据进行分析，而应该使用非参数检验（如卡方检验、置换检验等）。因此在对数据进行统计分析之前，第一步就需要对数据进行正态性检验，以检验该数据来自正态分布总体的概率有多大，再选择对应的参数或非参数检验方法进行分析。

正态性检验确定数据集是否服从整体分布后，进行相关性系数的计算。对于连续数据，满足正态分布，判断是否具有线性相关性时候，使用皮尔逊相关系数较为合适，如果不满足条件的话，应使用斯皮尔曼相关系数。

斯皮尔曼相关系数（Spearman）用于衡量两个变量之间的相关性，在进行皮尔逊相关系数运算的时候需要确定数据是否符合正态分布等等，较为麻烦，斯皮尔曼相关系数使用排序的方法消除量纲，在相关性分析中，用数据大小的排序代替原始的数据，也起到了消除量纲的作用。同时在分级数据中，我们适合使用斯皮尔曼相关系数。

斯皮尔曼相关系数的具体计算方法如下：

$$\rho=\frac{\Sigma_i\left(x_i-\bar{x}\right)\left(y_i-\bar{y}\right)}{\sqrt{\sum_i(xi-\bar{x})^2\sum_i\left(y_i-\bar{y}\right)^2}} \tag{3}$$

5.3 相关性降维

对于自变量和因变量的相关性检验，使用集成评分降维法 RSDRM 中的四种基学习器对于方差筛选后的 505 个特征变量分别进行特征评分，再使用集成评分降维法 RSDRM 进行加权打分，分别筛出其中得分排名进入前 100 的分子描述符进行加权打分，并统计在不同方法下评分排名进入前 100 的频次，最终综合两种评价指标，得到综合排名为前 26 的分子描述符特征得分表。

评价指标一：通过以上 4 种特征筛选方法，对于方差筛选后的 505 个特征分别进行打分，再加权打分得到特征的最终加权得分。

评价指标二：统计特征重要性进入前 100 的特征中，在 4 种特征筛选方法下得分排名在前 26 的频次。

集成评分降维法的打分公式如下：

$$G(\mathbf{x}) = \text{abs}\left(\sum_{t=1}^{T} \alpha_t \cdot g_t(\mathbf{x})\right),\ \alpha_t = 0.25 \tag{4}$$

其中 $g_t(),\ t = 1, 2, 3, 4$ 表示四种需要融合的相关性分析函数。对 26 个特征变量与因变量 pIC50 进行相关性分析，绘制热力图如下：

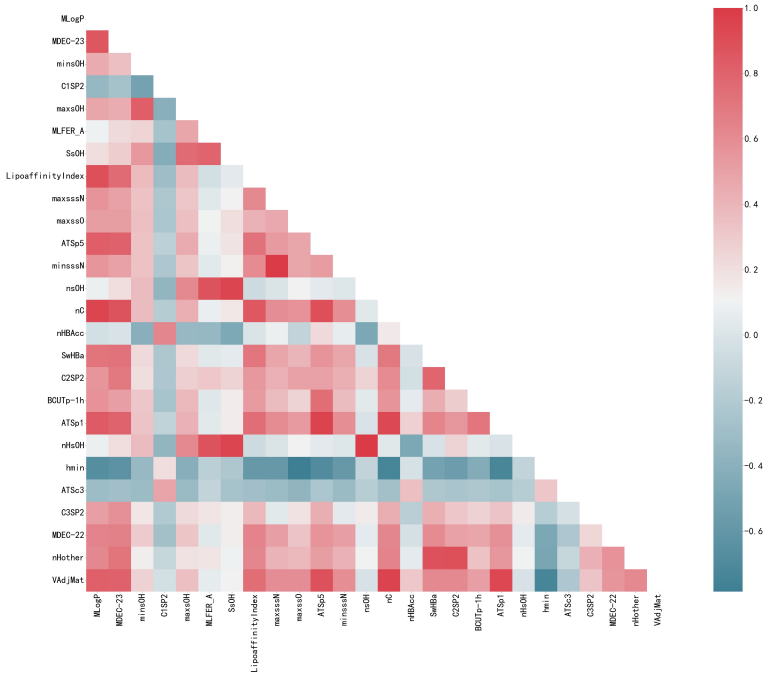

图 3　26 个特征变量与因变量 **pIC50** 的相关性分析

第一轮相关性特征降维结果及评分如下：

表 **1**　加权得分排名前 **26** 的特征及其得分表

特征	MIC 得分	Spearman 得分	随机森林得分	XGBoost 得分	加权得分	频次
MLogP		0.993	0.357	1	0.5875	3
MDEC-23		1	0.645	0.3273249	0.4930812	3
minsOH		0.777	0.884	0.0773166	0.4345791	3
C1SP2	0.0753	0.915	0.436	0.2606868	0.4217467	4
maxsOH		0.841	0.837		0.4195	2
MLFER_*A*	0.0694	0.721	0.829	0.0408723	0.415068	4
SsOH		0.723	0.907	0.0112448	0.4103112	3
LipoaffinityIndex		0.956	0.613	0.0602920	0.4073230	3
maxsssN		0.782	0.814	0.0217423	0.4044355	3
maxssO	0.0802	0.639	0.692	0.1503452	0.3903863	4
ATSp5		0.822	0.727		0.38725	2
minsssN		0.799	0.711	0.0113895	0.3803473	3
nsOH	0.209	0.67	0.634		0.37825	3
nC	0.0648	0.887	0.526		0.36945	3
nHBAcc	0.113	0.677	0.532	0.0549944	0.3442486	4
SwHBa		0.812	0.546		0.3395	2
C2SP2	0.0707	0.807	0.44	0.0105979	0.3320744	4
BCUTp-1h		0.788	0.5	0.0190066	0.326751	3
ATSp1		0.808	0.484	0.0081470	0.3250367	3
nHsOH	0.18	0.67	0.442		0.323	3
hmin	0.074	0.777	0.433		0.321	3
ATSc3	0.0628	0.699	0.433	0.0265883	0.3053470	4
C3SP2	0.121	0.675	0.307	0.077211	0.2950527	3
MDEC-22		0.755	0.412	0.0105435	0.2943858	3
nHother	0.0753	0.753	0.322		0.287575	3
VAdjMat	0.0681	0.725	0.348		0.285275	3

分别采用四种方法对特征重要性进行打分，具体过程如下：

正态检验-斯皮尔曼相关系数　对于分子描述符变量进行正态性检验，发现不符合正态分布，因此采用斯皮尔曼相关系数对于分子描述符变量的相关性进行打分。

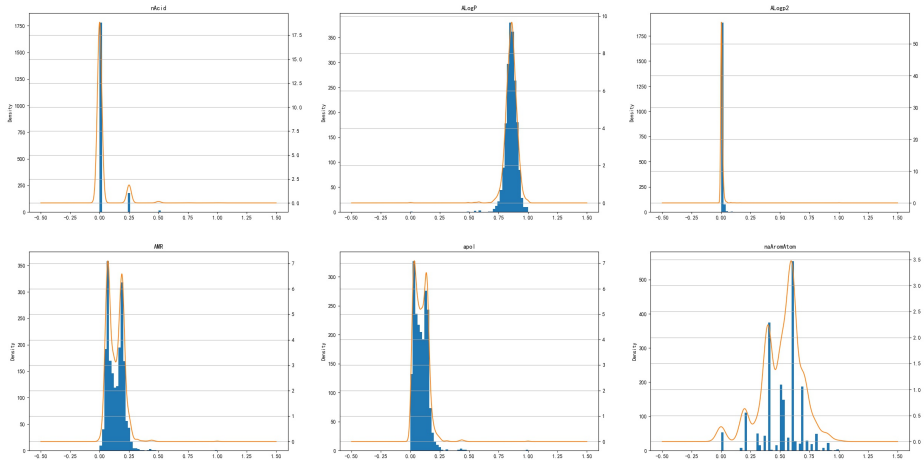

图 **4**　部分分子描述符变量的正态性检验示例

特征得分图：

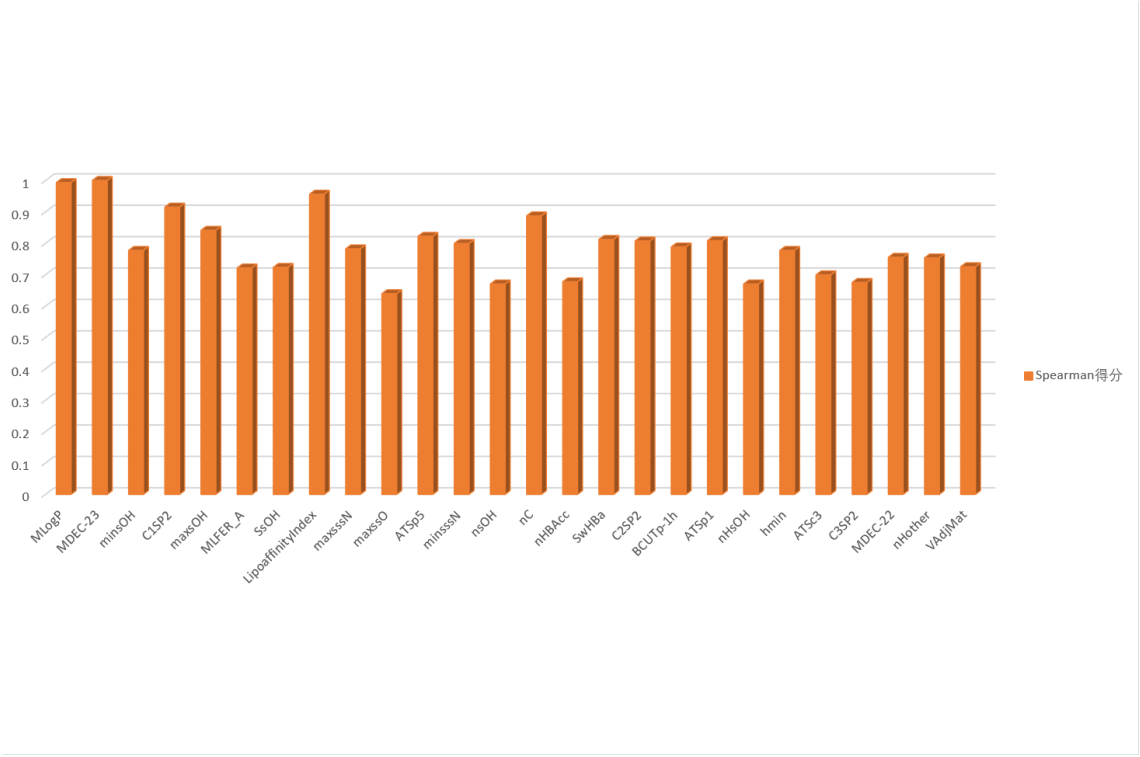

图 **5**　**Spearman** 方法下得分在前 **26** 的特征

随机森林　特征得分图：

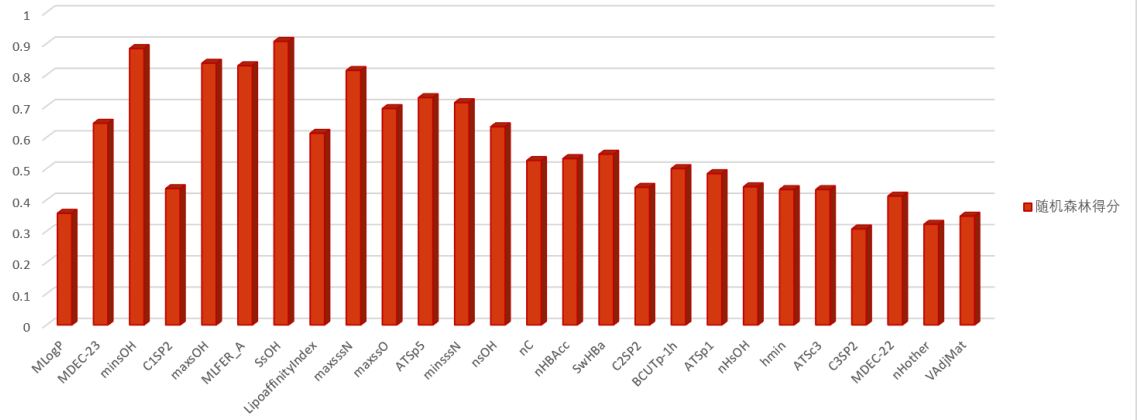

图 **6**　随机森林方法下得分在前 **26** 的特征

XGBoost　特征得分图：

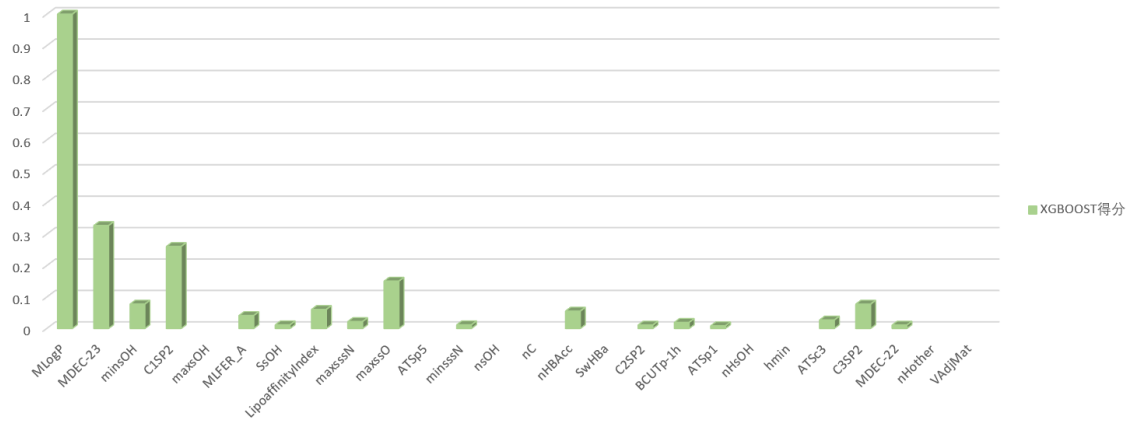

图 **7**　**XGBoost** 方法下得分在前 **26** 的特征

5.4 独立性降维

为去除高度相关的特征变量，对于第一轮筛选后的 26 个特征变量进行独立性检验，计算自变量之间的相关性。

筛选出与余下特征相关系数 >0.9 的 6 个变量，如下图所示。

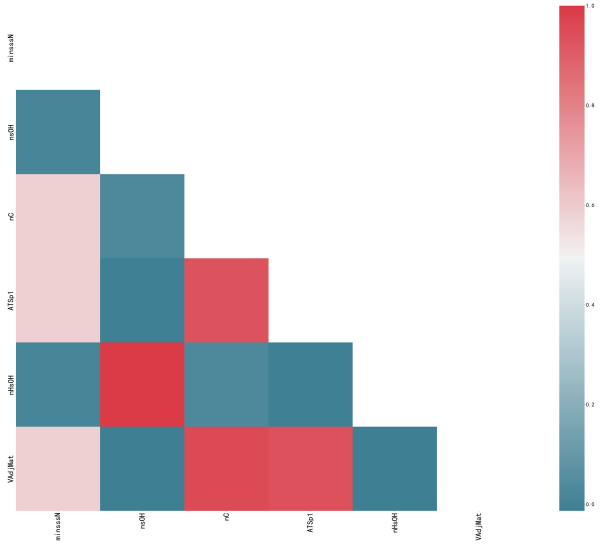

图 **8**　第二轮筛除的 **6** 个特征变量

筛除的特征具体信息如下：第二轮独立性降维后，最终得到 20 个特征变量，其得

表 **2**　相关系数 **>0.9** 的特征变量

特征	MIC 得分	Spearman 得分	随机森林得分	XGBoost 得分	加权得分	频次
minsssN		0.799	0.711	0.011389576	0.380347394	3
nsOH	0.209	0.67	0.634		0.37825	3
nC	0.0648	0.887	0.526		0.36945	3
ATSp1		0.808	0.484	0.008147077	0.325036769	3
nHsOH	0.18	0.67	0.442		0.323	3
VAdjMat	0.0681	0.725	0.348		0.285275	3

分排名如下：

表 **3**　第二轮筛选后的 **20** 个特征变量

特征	MIC 得分	Spearman 得分	随机森林得分	XGBoost 得分	加权得分	频次
MLogP		0.993	0.357	1	0.5875	3
MDEC-23		1	0.645	0.3273249	0.4930812	3
minsOH		0.777	0.884	0.0773166	0.4345791	3
C1SP2	0.0753	0.915	0.436	0.2606868	0.4217467	4
maxsOH		0.841	0.837		0.4195	2
MLFER_*A*	0.0694	0.721	0.829	0.0408723	0.415068	4
SsOH		0.723	0.907	0.0112448	0.4103112	3
LipoaffinityIndex		0.956	0.613	0.0602920	0.4073230	3
maxsssN		0.782	0.814	0.0217423	0.4044355	3
maxssO	0.0802	0.639	0.692	0.1503452	0.3903863	4
ATSp5		0.822	0.727		0.38725	2
nHBAcc	0.113	0.677	0.532	0.0549944	0.3442486	4
SwHBa		0.812	0.546		0.3395	2
C2SP2	0.0707	0.807	0.44	0.0105979	0.3320744	4
BCUTp-1h		0.788	0.5	0.0190066	0.326751	3
hmin	0.074	0.777	0.433		0.321	3
ATSc3	0.0628	0.699	0.433	0.0265883	0.3053470	4
C3SP2	0.121	0.675	0.307	0.077211	0.2950527	4
MDEC-22		0.755	0.412	0.0105435	0.2943858	3
nHother	0.0753	0.753	0.322		0.287575	3

最终得到 4 种特征筛选方法下的评分如图 9所示：

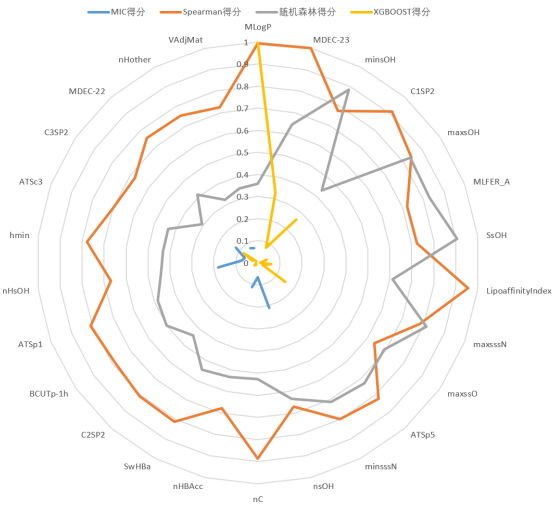

图 **9**　最终的 **20** 个特征变量在 **4** 种特征筛选方法下的评分

六、 问题 2：生物活性的定量预测

6.1 线性融合预测模型

我们采用模型线性融合的方式，通过独立/融合 XGBoost、随机森林、LightGBM 等三个模型对于生物活性 pIC50 进行预测，并通过负对数关系推导出 IC50 的值。问题二解题流程图如下所示：

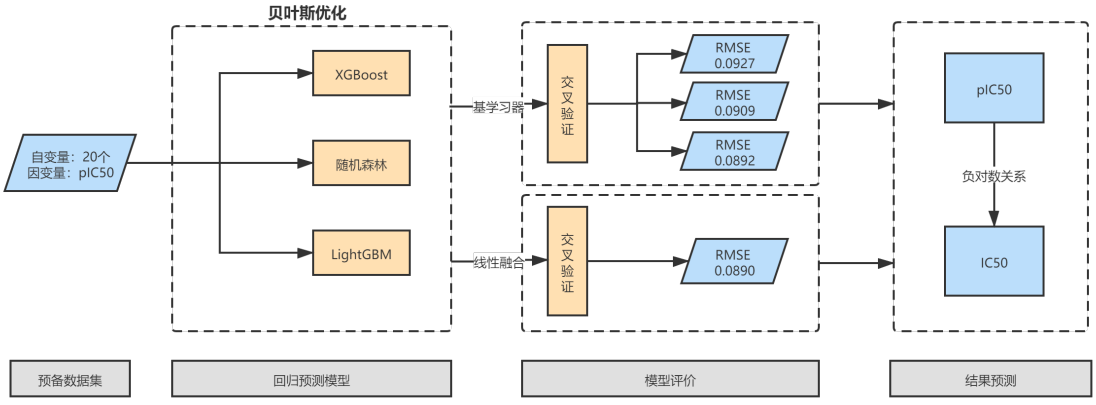

图 **10**　线性融合预测流程图

模型线性融合公式如图所示：

$$J(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot j_t(\mathbf{x})\right), \alpha_t = 1/3 \tag{5}$$

其中 $j_t(\mathbf{x}), t = 1, 2, 3$ 表示三个预测模型。本题最终反归一化后 pIC50 预测值如下，序号对应 SMILES 的序号：

表 **4**　**SMILES** 序号及其对应的 **pIC50** 值

1	2	3	4	5	6	7	8	9	10
6.7692	6.6486	6.9694	6.9365	7.0142	6.6759	6.9157	6.7919	7.1661	6.8036
11	12	13	14	15	16	17	18	19	20
6.7739	7.1154	6.6390	6.9771	6.9720	6.9660	6.7454	6.8262	6.2904	6.6715
21	22	23	24	25	26	27	28	29	30
6.8731	6.8282	5.8152	6.0249	5.7652	5.7153	6.2285	7.0004	6.2985	6.4925
31	32	33	34	35	36	37	38	39	40
5.3628	5.2342	5.1445	5.2428	5.2813	6.4554	6.5020	6.1033	5.9839	6.0671
41	42	43	44	45	46	47	48	49	50
6.0183	6.0202	6.0857	6.0113	6.0183	6.4394	7.3312	7.0398	7.0469	6.8251

6.1.1 随机森林

随机森林是一种基于 Bagging 的集成学习方法，就是用随机的方式建立一个森林，由很多决策树组成，且每一棵决策树之间是没有关联的。当出现新样本的时候，森林的每一棵决策树分别进行判断，看这个样本属于哪一类，然后用投票的方式，被选择的多的那一类作为最终的分类结果。在回归问题中，随机森林输出所有决策树输出的平均值。

6.1.2 XGBoost

全称是 eXtreme Gradient Boosting，如果 boost 算法每一步的弱预测模型生成都是依据损失函数的梯度方向，则称之为梯度提升 (Gradient boosting)，XGBoost 算法是采用分步前向加性模型，只不过在每次迭代中生成弱学习器后不再需要计算一个系数，模型形式如下：

$$F_T(X) = \sum_{m=0}^{T} f_m(X) \tag{6}$$

XGBoost 算法通过优化结构化损失函数（加入了正则项的损失函数，可以起到降低过拟合的风险）来实现弱学习器的生成，并且 XGBoost 算法没有采用搜索方法，而是直接利用了损失函数的一阶导数和二阶导数值，并通过预排序、加权分位数等技术来大大提高了算法的性能。

6.1.3 LightGBM

本问采用 LightGBM 模型进行预测。GBDT (Gradient Boosting Decision Tree) 是机器学习中一个长盛不衰的模型，其主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。GBDT 不仅在工业界应用广泛，通常被用于多分类、点击率预测、搜索排序等任务；在各种数据挖掘竞赛中也是致命武器，据统计 Kaggle 上的比赛有一半以上的冠军方案都是基于 GBDT。而 LightGBM（Light Gradient Boosting Machine）是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点。

为了避免上述 XGBoost 的缺陷，并且能够在不损害准确率的条件下加快 GBDT 模型的训练速度，lightGBM 在传统的 GBDT 算法上进行了如下优化：

- 基于 Histogram 的决策树算法
- 单边梯度采样 Gradient-based One-Side Sampling(GOSS)：使用 GOSS 可以减少大量只具有小梯度的数据实例，这样在计算信息增益的时候只利用剩下的具有高梯度的数据就可以了，相比 XGBoost 遍历所有特征值节省了不少时间和空间上的开销
- 互斥特征捆绑 Exclusive Feature Bundling(EFB)：使用 EFB 可以将许多互斥的特征绑定为一个特征，这样达到了降维的目的
- 带深度限制的 Leaf-wise 的叶子生长策略：大多数 GBDT 工具使用低效的按层生长 (level-wise) 的决策树生长策略，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销。实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。LightGBM 使用了带有深度限制的按叶子生长 (leaf-wise) 算法
- 直接支持类别特征 (Categorical Feature)
- 支持高效并行
- Cache 命中率优化

6.2 模型评估

评价指标 本题评价指标采用均方根误差 RMSE(Root Mean Square Error) 进行回归预测模型的性能评估：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{7}$$

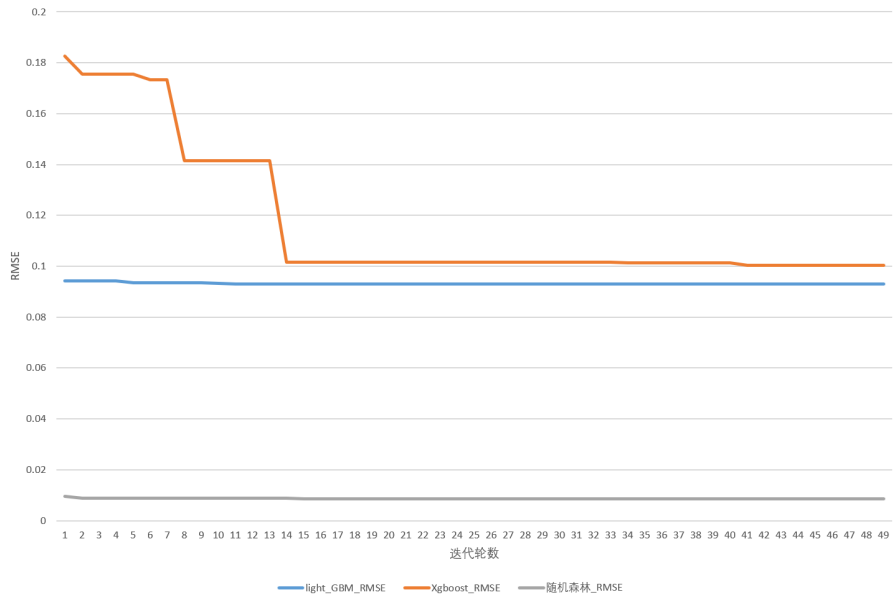

图 **11**　三个基模型评价指标 **RMSE** 迭代 **50** 次的数值变化

图 11是三个模型经过贝叶斯优化调参后得到的曲线性能。贝叶斯优化在下文讲述。

表 **5**　预测模型迭代 **50** 次时的模型性能对比

	随机森林	XGBoost	LightGBM	融合模型
RMSE	0.09087712	0.092696735	0.089217097	0.089068595

可以发现，线性融合模型相对于 3 个基学习器有一定的性能效果提升。

交叉验证：本题中化合物的数据样本量为 1974 条，数据量较小，因此采用交叉验证进行模型评估。根据切分方法不同，交叉验证分为下面三种：

1. 简单交叉验证，首先随机将样本数据分为两部分，然后用训练集来训练模型，在测试集上验证模型及参数。接着，再把样本打乱，重新选择训练集和测试集，继续训练数据和检验模型。最后我们选择损失函数评估最优模型和参数
2. K 折交叉验证（S-Folder Cross Validation），与简单交叉验证不同，K 折交叉验证先将数据集 D 随机划分为 S 个大小相同的互斥子集，即 $D = D_1 \cup D_2 \cup \ldots \cup D_S, D_i \cap D_j = \varnothing (i \neq j)$, 每次随机得选择 S-1 份作为训练集，剩下的 1 份做测试集。当这一轮完成后，重新随机选择 S-1 份来训练数据
3. 留一交叉验证（Leave-one-out Cross Validation），是第二种情况的特例，此时 S 等于样本数 N，这样对于 N 个样本，每次选择 N-1 个样本来训练数据，留一个样本来验证模型预测的好坏。此方法主要用于样本量非常少的情况，比如对于普通适中问题，

N 小于 50 时，一般采用留一交叉验证。除评价指标的数值之外，由于本题中需要对于不同模型在同一数据集上的预测准确性进行评估，因此采用简单交叉验证。

6.2.1 基于贝叶斯优化的模型调参

对于我们所提出的模型，为了使其发挥最大的效果，我们采用了一种自动调参算法——贝叶斯优化 (Bayesian Optimization, BO) 进行模型超参数优化。随着机器学习研究越发复杂，导致模型中许多超参数需要优化，因此超参数优化引起了大量学者的关注，已经变成了 AutoML 中的不可缺少的部分。目前比较常用的超参数优化算法有进化算法、贝叶斯优化和强化学习等。

BO 是近几年超参数优化极受欢迎的算法之一，原理图如下。但是因为典型的 BO 都是基于高斯过程，并且主要关注于低维连续优化问题，所以刚开始在神经网络结构搜索（NAS）比赛上没有应用（NAS 的意义在于解决机器学习模型的调参问题，是结合了优化和机器学习的交叉研究）。直到 2013 年，采用 BO 进行超参数优化开始有了一些动作，[1] 使用基于高斯过程的 BO 算法来推导出用于搜索网络结构的核函数，同时一些使用基于树模型的 BO 算法也开始研究，例如使用由 [2] 提出的 Treed Parzen Estimators 作为 BO 代理模型以及基于 Hutter 等人在 [3] 中提出的 SMAC 使用随机森林作为代理模型成功地在一系列问题上找到了高维条件空间并且取得了较优的表现。BO 自从 2013 年开始在 NAS 领域取得早期的成功，而后在 2015 年由贝叶斯优化所构建的模型在 CIFAR10 上取得了最好成绩。

贝叶斯优化算法的构成主要包括目标函数（黑盒）、搜索空间、代理模型和采集函数。目标函数即我们需调参的模型的精度，搜索空间为参数优化范围，代理模型根据已有数据点拟合目标函数，采集函数决定下一个最优采样点的位置得到新的观测值。

由于我们的三个预测模型都采用了贝叶斯优化进行超参数的优化，我们以 XGBOOST 模型为例, 我们的目标函数为最小化预测模型的 RMSE, 记为 $f(\cdot)$，我们选择 4 个需要优化的 4 个超参数，分别为学习率 (迭代决策树时的步长)，subsample (随机抽样时抽取样本的比例)，γ (复杂度的惩罚项) 以及 num_round (弱分类器的数量)，总记为 $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_4)$, 它们的搜索空间为 $\mathcal{A}$，即超参数取值范围，其中设置学习率为 [0.001, 0.2]，subsample 为 [0.6, 1]，γ 为 [0, 50] 以及 num_round 为 150-300 个。$\mathcal{D}_n = \{(\boldsymbol{\alpha}_i, f(\boldsymbol{\alpha})_i)\}_{i=1}^n$ 为样本空间。此外，我们采用最常见的高斯模型 ($\mathcal{GP}$) 为代理模型，记作 $\mathcal{M}_g$，以期望提升（EI）为采集函数，记为 $\mathcal{L}$。

我们的模型调参问题可以归结为

$$\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha} \in \mathcal{A}} f(\boldsymbol{\alpha}) \tag{8}$$

由于每次调参后都需训练模型导致大量时间成本, 贝叶斯优化算法旨在通过少量调参次数得到的样本空间 $\mathcal{D}$ 来获得最优的后验模型, 然后从这个模型得到最优超参

算法 1 贝叶斯优化

初始化：随机选 N_0 组超参数代入 $f(\boldsymbol{\alpha})$ 得到 $\mathcal{D}_{N_0}=\{(\boldsymbol{\alpha}_1,f(\boldsymbol{\alpha}_1)),\cdots,(\boldsymbol{\alpha}_{N_0},f(\boldsymbol{\alpha}_{N_0}))\}$，并且令 $n=N_0$。

当 $n\leq N$ 时:

1. 根据采集函数 $\mathcal{L}$ 选取下一个观测点 $\boldsymbol{\alpha}_{n+1}$，即 $\boldsymbol{\alpha}_{n+1}=\arg\max_{\boldsymbol{\alpha}\in\mathcal{A}}\mathcal{L}(\boldsymbol{\alpha};\mathcal{D}_n)$；
2. 得到 $f(\boldsymbol{\alpha}_{n+1})$，更新样本集 $\mathcal{D}_{n+1}=\{(\boldsymbol{\alpha}_1,f(\boldsymbol{\alpha}_1)),\cdots,(\boldsymbol{\alpha}_{N_0+1},f(\boldsymbol{\alpha}_{N_0+1}))\}$；
3. 根据样本集 $\mathcal{D}_{n+1}$ 更新后验 $p(\mathcal{M}_g|\mathcal{D}_{n+1})$;
4. $n=n+1$.

数组合。从本质上来说，贝叶斯优化是一种基于顺序模型的算法，其算法流程由算法1所示。首先从搜索空间中随机选取 N_0 组超参数分别进行模型训练，得到样本空间 $\mathcal{D}_{N_0}=\{(\alpha_i,f(\alpha)_i)\}_{i=1}^{N_0}$，若给定目标函数 f 一个先验分布 $\mathcal{M}_g$，此时可以根据贝叶斯公式得到后验模型：

$$p(\mathcal{M}_g|\mathcal{D}_{N_0})\propto p(\mathcal{D}_{N_0}|\mathcal{M}_g)p(\mathcal{M}_g) \tag{9}$$

图 12显示了三个预测模型通过贝叶斯优化后的相关参数及初始参数。

	参数	参数含义	取值范围	结果
随机森林	*n_estimators*	随机森林中树模型的数量	(100,500)	386
	min_samples_split	一个中间节点要分支所需要的最小样本量	(2,25)	2
	max_features	在做最佳分枝时，考虑的特征个数	(5,20)	5
Xgboost	eta	集成中的学习率	(0.001,0.2)	0.0216
	subsample	从样本中进行采样的比例	(0.6,1)	0.645
	gama	复杂度的惩罚项	(0,50)	0.03
	num_round	弱分类器的数量	(150,300)	190
	learning_rate	集成中的学习率	(0.001,0.05)	0.013
LightGBM	*num_leaves*	一棵树上的叶子节点个数	(30,50)	47
	feature_fraction	从特征中进行采样的比例	(0.8,1)	0.82
	bagging_fraction	从样本中进行采样的比例	(0.8,1)	0.801

图 **12**　三个基模型贝叶斯调优参数

然后根据采集函数选定下一次的最优观测点，得到观测点的样本集，再根据上述贝叶斯公式和新的样本空间更新 $p(\mathcal{M}|\mathcal{D}_{N_0+1})$。并以此种方式迭代至 N 次。

七、 问题 3：ADMET 性质的分类预测

此问的处理流程如图 13所示，首先，我们对于问题 1 中预处理得到的 505 个分子描述符分别对 ADMET 五个性质进行相关性分析，并采用问题 1 中的特征降维方法进行变量选择。并且对于该问题的分类模型构建，我们采取了两种方案：多模型单标签分类和单模型多标签分类。。前者的意思是根据多个标签分别构建 1 个分类预测模型，此种方案忽略了各个标签的相关性，泛化性不强，但其优势在于结构简单，处理方便。后者的

意思在于构建一个分类模型进行多个标签的分类，因此考虑了多个标签之间的关联性，在实际场景中更好得以应用。图中显示了我们所采用的单标签 LightGBM 分类模型以及多标签 MPVAE 分类模型。

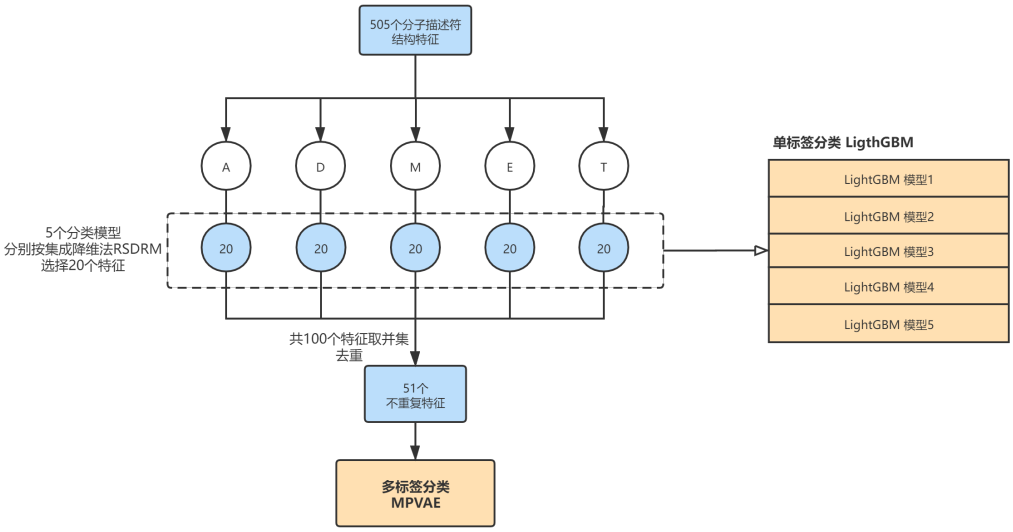

图 **13**　问题 **3** 的处理流程图

7.1 决策变量选择

为了进一步探索化合物的分子描述符和其 ADMET 性质之间的深层关系，构建有效的预测模型，我们使用问题一构建的集成降维算法，将经过方差过滤后的 505 个特征作为自变量，5 个 ADMET 性质 Caco-2，CYP3A4，hERG，HOB，MN 作为因变量分别进行降维。经过对线性加权分数大小的比对，我们针对每一个因变量，均筛选出了对其具有显著影响的 20 个分子描述符变量。我们可以从选择的变量中发现，针对不同因变量筛选出的重要特征中具有一定程度的重复性，例如分子描述符为“SP-1“的变量就同时对 Caco-2，CYP3A4，hERG 三种性质均产生了显著影响。为达到对化合物的 ADMET 性质进行准确预测的目的，我们对上表所列的变量取交集后形成新的特征集合，该集合内所包含的所有特征均可对化合物的 ADMET 性质产生不同程度的影响。新的特征集合所包含的分子描述符如下表所示。

7.2 基于 LightGBM 的单标签分类

在单标签分类方案的处理中，我们采用 LightGBM 模型进行二分类。由于在前面的预测模型中采用过了 LightGBM，这里就不介绍原理。

如图 14所示，展示了 ADMET 五种标签根据五个单独训练的 LightGBM 模型的 AUC 性能，可以看出对于 MN 的分类预测效果较好，最大 AUC(ROC 曲线下的面积) 达到了 0.98 以上，较差的是 HOB。

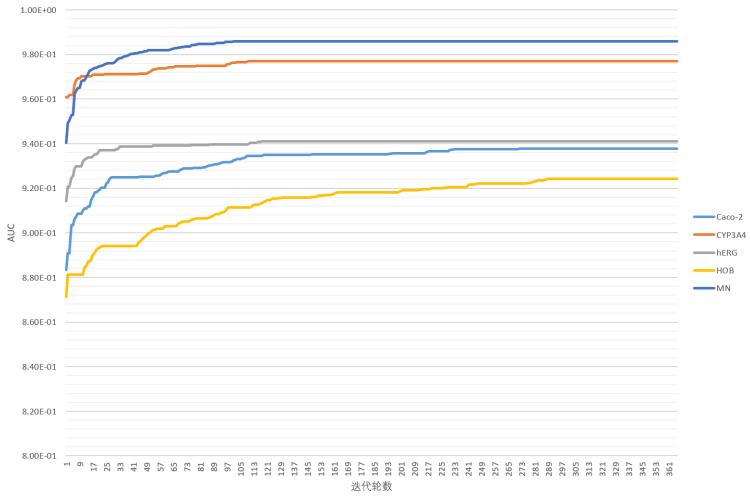

图 **14**　基于五个 **LightGBM** 分类模型的 **AUC** 曲线

7.3 多标签分类

在此问题中，可将对五个标签（ADMET 五个性质）的分类问题看作是多标签二分类问题，即相同的数据特征对五个标签都进行二分类预测。而多标签分类问题最大的特性在于多个标签之间的相关性。如果将 ADMET 分类问题当作五个单标签分类问题处理，难免会丢失标签之间的关联性信息，在实际问题中模型可能得不出较优的结果并且没有较好的泛化性。更一般地，多标签分类问题处理策略可以分为：

- 一阶策略: 忽略和其他标签的相关性，比如把多标签分类分解为多个独立的二分类问题。比如将多标签学习分解为若干个独立的二分类问题。一阶策略的主要价值在于概念上简单、效率高。但另一方面, 结果的有效性和泛化性不是最优的。
- 二阶策略：用标签之间的成对关系来处理。比如在相关和不相关的标签之间排序，或者任意两个标签之间的互动。结果可以取得不错的泛化表现。然而，有一些现实世界的问题的标签相互关系超越了二阶假设。
- 高阶策略：考虑标签之间的高阶关系，比如将所有其他标签的影响施加到每个标签上，或寻找随机标签子集之间的关系等。

7.3.1 多标签相关性分析

为了证明在此问题上多标签分类模型的可行性，我们进行了 ADMET 标签间的相关性分析。由于标签是 0-1 变量，常用的相关性分析方法表现不好，因此我们采用克莱姆 V（Cramer’s V）相关性分析，又称为克莱姆相关系数、独立系数等，是双变量相关分析的一种方法，专门用于衡量分类数据与分类数据之间相关程度。该系数取值范围为 0 到 1，0 表示两个变量无关，1 表示完全相关。

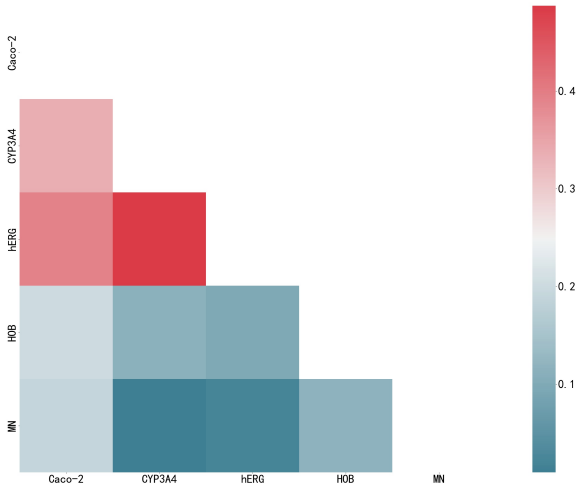

图 **15** **ADMET** 五个性质的 **Cramer’s V** 相关性分析

如图 15所示，可以看出 ADMET 性质中某些变量存在着不弱的相关性。比如 Caco-2 与 CYP3A4,hERG 的相关性系数分别为 0.34 和 0.40，hERG 和 CYP3A4 之间的相关性系数为 0.49。但是 MN 与其它标签之间的相关性较弱。总体而言，这五个标签存在着一定的相关性，对其进行多标签分类建模也存有合理性。不仅如此，在关于 ADMET 的文献 [4] 中，作者研究了 ADMET 中部分性质的关联性。此外，在拜尔医药于 2020 年所发表的论文 [5] 中，考虑了 ADMET 各个性质的相关性并采用了深度神经网络进行了多任务的 ADMET 的预测，多标签分类也是属于多任务学习的一种。

因此我们可直接将 ADMET 的分类问题建模成多标签分类问题，这样处理的好处是可以挖掘 ADMET 五个标签之间的关联性。为了处理该问题，我们采用了多元概率比变分自编码器方法（Multivariate Probit Variational AutoEncoder , MPVAE），该方法于 2020 年由 [6] 提出，当时在各个数据集与其他方案对比中都取得了最优的结果，并且这些数据集包含标签维度有小至 6 标签的数据集，同时也有小样本数据集。所以我们认为此方法可以应用在我们的 ADMET 五标签分类问题上。

7.3.2 多元概率比变分自编码器

MPVAE 算法主要由多元概率比模型（MP）和变分自编码器（VAE）两部分所组成。MP 是一个经典的关于数据关联的潜变量模型。与大多数深度方法中的典型 softmax 变换不同，该模型将样本从多元正态分布映射到其自身的累积分布（CDF），以将输出范围限制在 [0,1] 范围内。VAE 是一类重要的生成模型。首先，自编码器（AE）是一种数据压缩方式，它把一个数据点 x 有损编码为低维的隐向量 z，通过 z 可以解码重构回 x。这是一个确定性过程，但我们实际无法拿它来生成任意数据，因为我们要想得到 z，就

必须先用 x 编码。VAE 可以解决这个问题，它可以直接通过模型生成隐向量 z，并且生成的 z 是既包含了数据信息又包含了噪声，因此用各不相同的 z 可以生成无穷无尽的新数据。

接下来以我们的数据集为例，我们对于此算法进行简要推导。首先给定数据集 $\mathcal{D}_{ADMET} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, 其中 N 为数据集样本数量，$\mathbf{x}_i$ 为第 i 个样本决策变量的值，$\mathbf{y}_i \in \{0,1\}^L$ 为第 i 个样本的 L 个标签值且 $L = 5$。

- 变分自编码器: VAE 通过引入潜在变量 z，为观测数据点 X : $P(X) = \int p_\theta(X \mid z;\theta)P(z)dz$ 假设一个生成过程。由于大多数 z 对 $P(X)$ 的贡献很小，常见的蒙特卡罗抽样方法的有效性很低。而变分法通过有效的采样学习得到一个分布 $q_\phi(z \mid X)$ 去逼近难以处理的 $P(z \mid X)$。KL 散度（Kullback-Leibler divergence）($\mathcal{D}$) 可用于衡量两个分布的距离，即 $\mathcal{D}\left(q_\phi(z \mid X) \| P(z \mid X)\right) = \mathbb{E}_{z\sim q_\phi}\left[\log q_\phi(z \mid X) - \log P(z \mid X)\right]$。采用贝叶斯准则可得到：$\log P(X) - \mathcal{D}[Q(z \mid X) \| P(z \mid X)] = \mathbb{E}_{z\sim q_\phi}\left[\log p_\theta(X \mid z)\right] - \mathcal{D}\left[q_\phi(z \mid X) \| P(z)\right]$。等式的右侧是要最大化的可处理证据下限（ELBO）。$P(z)$ 是一个标准多元正太分布，ELBO 中的第一项专注于 X，第二项惩罚近似分布和建立在潜在空间上的先验之间的 KL 散度。p_θ 和 q_ϕ 都可以通过神经网络进行参数化。借助于重新参数化方式，可以使用反向传播对整个模型进行训练。
- 多元概率比模型: 便于理解，考虑单个样本 $(\mathbf{x}, \mathbf{y})$，其中 $\mathbf{x}$ 是输入特征向量，$\mathbf{y} \in \{0,1\}^L$ 是标签向量，MPVAE 引入了隐变量 $\mathbf{y}^* \in \mathbb{R}^L$，其服从一个多元正太分布 $\mathcal{N}(\mathbf{x}\gamma, \Sigma)$，其中 γ 是权重因子，Σ 是协方差矩阵。$\mathbf{y}$ 表示为 $\mathbf{y}^*$ 是否为正的指示符：$y_i = \mathbb{1}\left\{y_i^* > 0\right\}, i = 1, \ldots, L$。观测概率 $\mathbf{y}$ 由 $P(\mathbf{y} \mid \mathbf{x}\gamma, \Sigma) = \int_{A_L} \ldots \int_{A_1} p\left(\mathbf{y}^* \mid \mathbf{x}\gamma, \Sigma\right) dy_1^* \ldots dy_L^*$ 所给定，其中如果 $y_j = 0$，$A_j = (-\infty, 0]$，反之, 则 $A_j = (0, \infty)$。$p(\cdot)$ 表示正太分布的概率密度函数（PDF），只要用神经网络 $f(\mathbf{x})$ 替换平均值 $\mathbf{x}\gamma$，就可以将该框架推广到深度模型。在 MPVAE 中，平均值由 VAE 的解码器给出。

整个模型可以看作是一个两阶段的生成过程，第一阶段将特征和标签映射到高斯子空间，其中均值和方差由多层感知器学习。这一阶段的关键任务是匹配这两个子空间。第二阶段对每个子空间中的样本进行解码，并将输出作为平均值输入到多元概率模块中，然后分别学习全局协方差矩阵。最终，第二阶段的输出给出了预测的标签值。

如图 x 所示，特征编码器使用 ψ 参数化的神经网络将 x 编码到概率潜在子空间。类似地，另一个带有参数 ϕ 的标签编码器将 y 映射到另一个具有相同维数的概率潜在子空间。来自子空间的两个样本 z_x, z_y 被送入共享解码器，并在多元概率比模型中被破译为 m_x, m_y。在全局协方差矩阵 Σg 的信息协助下，我们从 $\mathcal{N}\left(m_x, \Sigma_g\right), \mathcal{N}\left(m_y, \Sigma_g\right)$ 中对 s_x, s_y 进行采样，以得出最终的 $0-1$ 预测值 $\hat{y}_f, \hat{y}_l$。在测试输出过程中，仅需考虑 $\hat{y}_f$ 为预测输出值。

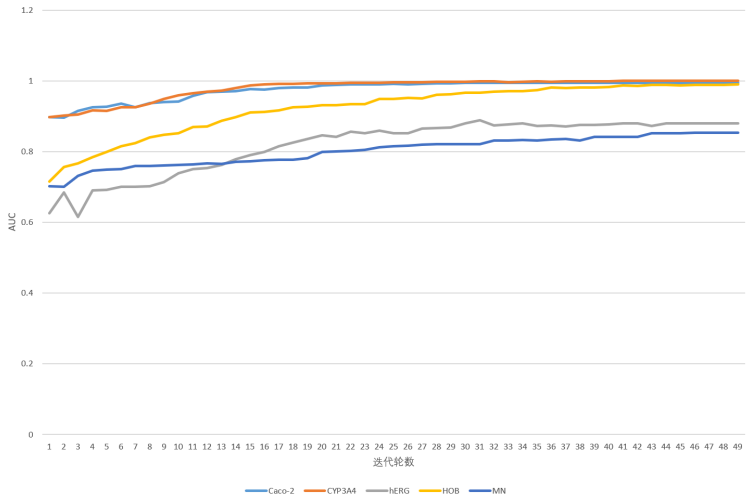

图 **16**　**MPVAE** 多标签分类 **AUC** 性能曲线

如图 16所示，其中对于三类标签有着较好的预测性能，分别是 CYP3A4,MN 和 Caco-2。但是对于 MN 和 hERG，表现得不够好，这可以呼应上文 ADMET 五个性质相关性分析得出的结论，即 MN 与其它标签相关性低，因此在联合预测时挖掘不到隐层信息，故性能较差。

7.4 分类结果分析

首先考虑样本集参数：我们将总样本划分为百分之八十的训练集和百分之二十的验证集，验证集需要用于评估两类方案各种性能，以及两种方案的对比。我们这里没有考虑到离线训练模型与在线使用模型的时间复杂度对比。我们的对比主要是根据 AUC 性能。

八、 问题 4：分子描述符优化

该问的目的在于寻找合适的分子描述符变量以及取值使化合物对抑制 ERα 具有更好的生物活性，同时具有更好的 ADMET 性质（在给定的五个 ADMET 性质中，至少三个性质较好）。

首先，为了选取合适的分子描述符，我们结合了问题 2 和问题 3 的决策变量选择过程，然后构建出此问所需的决策变量。然后，我们在此基础上进行对变量的优化来最大化生物活性并满足更好 ADMET 性质的约束条件，需要注意的是，我们可以发现分子描述符分为连续型变量和整数型变量。所以此优化问题属于混合整数规划问题，即其中部分决策变量限制为整数。对于这种混合整数规划问题，智能优化算法可以提供很好的帮助，因此我们采用了麻雀搜索来进行参数寻优。另外，此优化问题还有着 ADMET 性质中至少三个性质较好的非线性约束，也就是说在利用智能优化算法优化自变量时，需要

满足非线性约束。一般对于这样的情况，智能优化算法的一般的解决策略是采用丢弃法(不满足约束条件的解直接丢弃，并寻找新解)，这里，我们采用基于惩罚函数法的策略，其核心思想是将约束条件转化为惩罚函数，附加在原有的目标函数上构造新的目标函数；当不满足约束条件时，通过惩罚函数使新的目标函数变差而被舍弃。因此，我们采用外点法将非线性约束转化为无约束，此时原优化问题只存在决策变量的取值约束 (自变量取值约束无需加入惩罚项转化为无约束，因为自变量范围在智能优化算法中可以直接设置)，然后再利用麻雀搜索求解转化后的混合整数规划问题。整体流程如图 17所示。

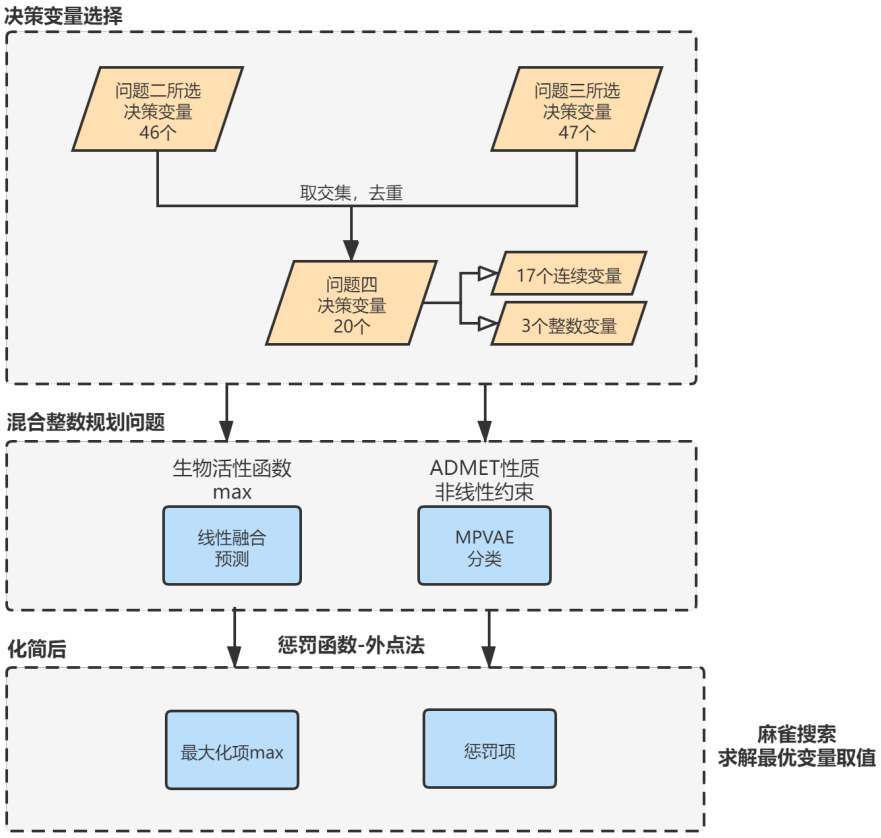

图 **17**　问题 **4** 处理流程图

8.1 决策变量选择

为了寻找同时满足两个要求的决策变量，我们需要利用到问题 2 和问题 3 的决策变量选择过程，并选出符合题意的决策变量。我们的核心思想是利用对生物活性相关的特征和对 ADMET 性质相关的特征取交集，如此可以得到既与生物活性相关又与 ADMET 性质相关的特征。

我们具体的做法是：在本文的问题 2 和问题 3 部分中，我们已经利用了问题 1 构建的集成降维算法，分别是筛选出了对生物活性具有显著影响的变量集合，和对生物 ADMET 性质具有显著影响的变量集合，两个集合中分别包含 20 个变量和 51 个变量。

为了更好地搜寻对化合物的两方面性质均能产生显著影响的分子描述符及其能够产生良性影响的所处范围，我们需要综合问题 2 及问题 3 的变量筛选结果，来对我们的变量优化范围做进一步的精确化搜索。在我们对问题 2 中的对生物活性的定量预测模型的构建过程中，已经得到初步分析结果，对生物活性影响显著的分子描述符变量不超过 20 个。基于问题 4 中所选择的分子描述符变量需要同时对两方面性质均可产生显著影响，我们可以得出的判断是我们在需要优化的变量个数在 20 个以内，否则将与我们在问题 2 中得出的结论相悖。由此，我们分别将在问题 1 与问题 2 中，加权得分分列前 50 名的变量，进行交集筛选，即只有对抑制 ERα 具有更好的生物活性且总体上能够对化合物的 AMDET 性质产生正面影响的分子描述符变量，才能够被划入本问题的优化建模范围内。我们全面考虑了分子描述符的数据特性，及其在集成降维算法中的表现后，筛选出了 20 个具有优化空间的分子描述符变量，如下所示：minsOH，maxsOH ，SsOH ，nHsOH ，apol ，fragC ，hmin ，McGowan_Volume ，VABC ，VP-0 ，maxHsOH ，WTPT-1 ，ETA_Eta_R ， VP-1 ， SP-1 ， CrippenMR ， VAdjMat ， ATSp3 ， nBonds ， nHBAcc。

8.2 变量取值优化

从此问题的变量取值优化角度来看，这是根据上述的所解决的问题，对于我们构建取值优化问题，我们先定义所要用到的参数和模型，如下所示：

- 决策变量：分为连续变量和整数变量，分别记作 $\mathbf{v}^r = \{v_i^r\}_{i=1}^{N^r}, \mathbf{v}^z = \{v_j^z\}_{j=1}^{N^z}$，总变量集为 $\mathbf{v} = \mathbf{v}^r \cup \mathbf{v}^z$，其中 N^r 和 N^z 分别为连续变量和整数变量的个数。
- ERα 生物活性值：记为 E，根据问题 2 以此问所选变量建立生物活性预测模型 $\mathcal{M}_p$，满足 $E = \mathcal{M}_p(\mathbf{v})$。
- ADMET 性质：记为 $\mathbf{h} = \{h_l\}_{l=1}^5$，即 $\{h_1, h_2, h_3, h_4, h_5\}$，分别表示 Caco-2，CYP3A4，hERG，HOB，MN 的分类预测值。根据问题 3 构建多标签分类模型 $\mathcal{M}_c$，且满足 $\mathbf{h} = \mathcal{M}_c(\mathbf{v})$。由题意得，我们的优化问题需要满足 ADMET 性质中至少三个性质较好，因此我们根据表 6的描述来定义每个性质“好”的时候所取的 0-1 值，记为 $\{\hat{h}_1, \hat{h}_2, \hat{h}_3, \hat{h}_4, \hat{h}_5\} = \{1, 0, 0, 1, 0\}$。

我们以最大化生物活性函数 $\mathcal{M}_p(\mathbf{v})$ 为目标，并以满足 ADMET 至少三个性质较好的限制为非线性约束，以及考虑决策变量 $\mathbf{v}$ 的取值范围，则我们的优化问题可以写成：

$$
\begin{aligned}
&\arg\max_{\mathbf{v}} \mathcal{M}_p(\mathbf{v}) \\
&\text{s.t.} \sum_{l=1}^{5} (h_l \odot \hat{h}_l) \geq 3 \\
&v_i^r \in [l_i, u_i], \\
&v_j^z \in \left\{z_j^1, z_j^2, \cdots, z_j^M\right\}
\end{aligned}
\tag{10}
$$

表 **6**　标准三线表格

指标	1 的含义	0 的含义	性质为优时数值
Caco-2	渗透性好	渗透性差	1
CYP4503A4	被 CYP3A4 代谢好	被 CYP3A4 代谢差	0
hERG	有心脏毒性	无心脏毒性	0
HOB	生物利用度好	生物利用度差	1
MN	遗传毒性	无遗传毒性	0

其中 l_i 和 u_i 分别表示第 i 个连续型变量 v_i^r 的下界和上界, $\{z_j^m\}_{m=1}^M$ 表示第 j 个整数型变量的 M 个取值，$\odot$ 为同或运算，此处的物理意义为当某预测分类值与其相应性质较好时所取的值相同时，则记 1，反之，为 0。

从上述优化问题可以看出，各个变量除了有各自的取值范围外，还需满足关于由 $\mathcal{M}_c(\mathbf{v})$ 得出的关于 $\mathbf{h}$ 的非线性不等式。对于此问题，我们采用惩罚函数法来解决。惩罚函数法是一类常用的处理约束条件的技术，方法的思想是将约束条件转化为惩罚函数，附加在原有的目标函数上构造新的目标函数；当不满足约束条件时，通过惩罚函数使新的目标函数变差而被舍弃。惩罚函数法有外点法和内点法。外点法对可行域外的点（即不满足约束的点）施加惩罚，对可行域内部的点不惩罚，从而使迭代点向可行域 D 逼近。内点法是在可行域内部进行搜索，约束边界起到类似围墙的作用，使目标函数无法穿过，就把搜索点限制在可行域内了，因此只适用于不等式约束。

8.2.1 外点法

首先，我们由表已知 $\hat{\mathbf{h}} = \{1, 0, 0, 1, 0\}$, 所以我们可以对于同或运算进行化简：

$$A \odot B = AB + \bar{A}\bar{B} \tag{11}$$

其中 A, B 分别为二进制 0-1 值，$\bar{A}, \bar{B}$ 分别表示为 A 非，B 非。则有

$$\begin{aligned}\sum_{l=1}^{5}(h_l \odot \hat{h}_l) =& h_1 \odot 1 + h_2 \odot 0 + h_3 \odot 0 + h_4 \odot 1 + h_5 \odot 0 \\ =& h_1 - h_2 - h_3 + h_4 - h_5 + 3\end{aligned} \tag{12}$$

然后，我们采用外点法将式 (10) 中的非线性约束转化为惩罚项加入目标函数：

$$\begin{aligned} \arg\min_{\mathbf{v}} \ & -\mathcal{M}_p(\mathbf{v})+\sigma[\max\{0,-h_1+h_2+h_3-h_4+h_5\}]^2 \\ \text{s.t. } & v_i^r \in [l_i,u_i], \\ & v_j^z \in \{z_j^1, z_j^2, \cdots, z_j^M\} \end{aligned} \tag{13}$$

其中 σ 为惩罚因子，在实际优化过程中，σ 的设置会逐渐变大。

对于以上化简后的混合整数规划问题，只存在决策变量的取值约束，此类约束条件在启发式算法中比较容易处理，只要设定初始解、新解在决策变量取值的上下限之间就可以解决。我们采用一种较新颖的智能优化算法——麻雀搜索进行优化决策变量取值。

8.2.2 麻雀搜索

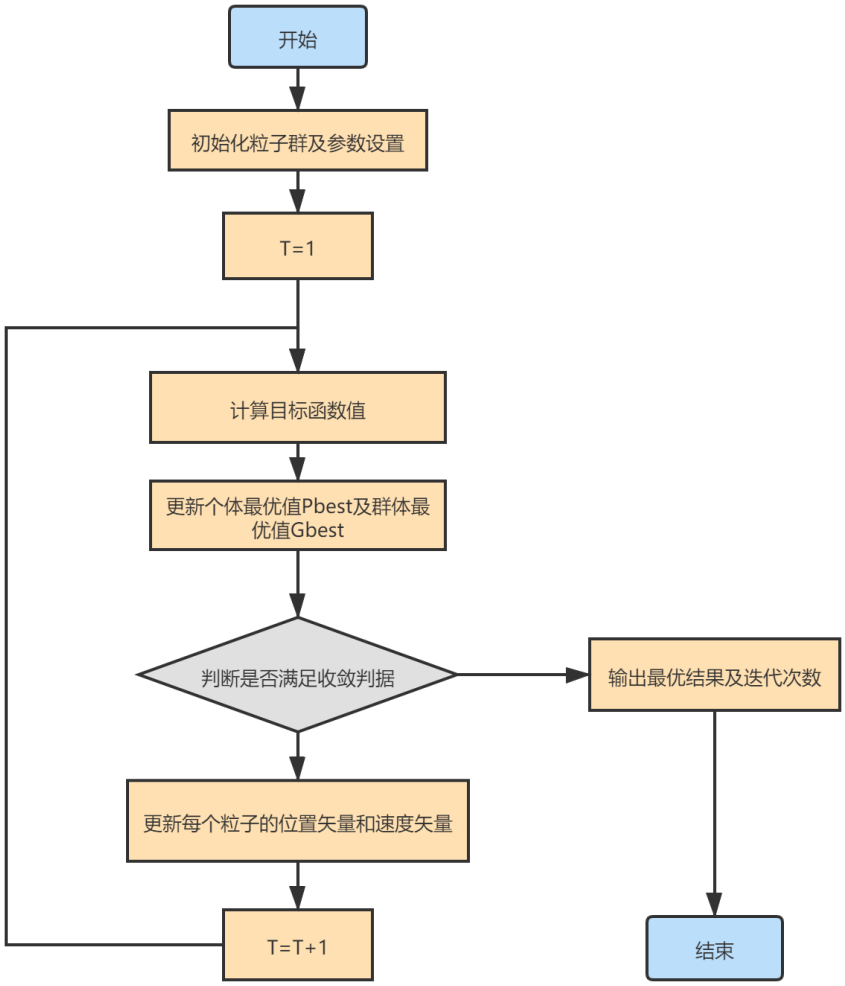

图 **18**　麻雀搜索算法流程图

麻雀搜索算法 (Sparrow Search Algorithm, SSA) 是一种新型的群智能优化算法，在 2020 年由 [7] 提出，主要是受麻雀的觅食行为和反捕食行为的启发。其算法流程如图

18所示，在 SSA 中，具有较好造应度值的发现者在援家过程中会优先获取食物。此外，因为发现者负责为整个麻雀种群寻找食物并为所有加入者提供觅食的方向。因此，发现者可以获得比加入者更大的觅食搜索范围。在每次迭代的过程中，发现者的位置更新描述如下：

$$X_{i,j}^{t+1}=\begin{cases}X_{i,j}^{t}\cdot\exp\left(\frac{-i}{\alpha\cdot \text{iter}_{\max}}\right) & \text{if} \quad R_2<ST\\ X_{i,j}^{t}+Q\cdot L & \text{if} \quad R_2\geq ST\end{cases} \tag{14}$$

其中，t 代表当前迭代数，$iter_{max}$ 是一个常数，表示最大的迭代次数。X_{ij} 表示第 i 个麻雀在第 j 维中的位置值息，$0\in(0,1]$ 是一个随机数。$R2(R2\in[0,1])$ 和 $ST(ST\in[0.5,1])$ 分别表示预值和安全值。Q 是服从正态分布的随机数。L 表示一个 $1\times d$ 的矩阵，其中该矩阵内每个元素全部为 1. 当 $R_2<ST$ 时，这意味着此时的觅食环境周围没有捕食者，发现者可以执行广泛的搜索操作。当 $R_2{\geq}ST$，这表示种群中的一些麻雀已经发现了捕食者，并向种群中其它麻雀发出了警报，此时所有麻都需要迅速飞到其它安全的地方进行觅食。加入者（追随者）的位置更新描述如下：

$$X_{i,j}^{t+1}=\begin{cases}Q\cdot\exp\left(\frac{X_{\text{worst}}^{t}-X_{i,j}^{t}}{i^2}\right) & \text{if} \quad i>n/2\\ X_{P}^{t+1}+\left|X_{i,j}^{t}-X_{P}^{t+1}\right|\cdot A^{+}\cdot L & \text{otherwise}\end{cases} \tag{15}$$

其中，X_p 是目前发现者所占据的最优位置，X_{worst} 则表示当前全局最差的位置。A 表示一个 1xd 的矩阵，其中每个元素随机赋值为 1 或-1，并且 $A^{+}=A^{T}\left(AA^{\top}\right)^{-1}$。当 $i>n/2$ 时，这表明，适应度值较低的第 i 个加入者没有获得食物，处于十分饥饿的状态，此时需要飞往其他地方觅食，以获得更多的能量。当意识到危险时，麻雀种群会做出反捕食行为，其数学表达式如下：

$$X_{i,j}^{t+1}=\begin{cases}X_{\text{best}}^{t}+\beta\cdot\left|X_{i,j}^{t}-X_{b\,\text{bst}}^{t}\right| & \text{if} \quad f_i>f_g\\ X_{i,j}^{t}+K\cdot\left(\frac{\left|X_{i,j}^{t}-X_{\text{worst}}^{t}\right|}{(f_i-f_w)+\varepsilon}\right) & \text{if} \quad f_i=f_9\end{cases} \tag{16}$$

其中，X_{best} 是当前的全局最优位置。β 作为步长控制参数，是服从均值为 0，方差为 1 的正态分布的随机数。$K\in[-1,1]$ 是一个随机数，f_i 则是当前麻雀个体的适应度值。f_g 和 f_w 分别是当前全局最佳和最差的适应度值。ε 是最小的常数，以避免分母出现零。为简单起见，当 $f_i>f_g$ 表示此时的麻雀正处于种群的边缘，极其容易受到捕食者的攻击。$f_j=f_g$ 时，这表明处于种群中间的麻雀意识到了危险，需要靠近其他的麻雀以此尽量减少它们被捕食的风险。K 表示麻雀移动的方向同时也是步长控制参数。

8.3 优化结果分析

8.3.1 参数说明

对于 20 个决策变量的取值范围如表 7所示，其值的选取为附件二中对应变量的最小值和最大值。需要注意的是，其中有 3 个是整型变量。惩罚因子 σ 初始为 1，随着麻

表 **7**　决策变量取值范围

决策变量	最小值	最大值
minsOH	0	11.73233644
maxsOH	0	12.47085005
SsOH	0	65.62532567
nHsOH	0	6
apol	30.66193	359.66274
fragC	274.06	91243.68
hmin	-0.588781415	0.388364158
McGowan_Volume	1.554	18.2733
VABC	182.5471121	2284.171248
VP-0	8.194431597	95.33733465
maxHsOH	0	0.852772151
WTPT-1	27.17055246	318.0768446
ETA_Eta_R	18.67717	628.15255
VP-1	4.699004209	55.92431431
SP-1	6.630228955	76.7668832
CrippenMR	56.1508	615.6786
VAdjMat	4.807354922	8.348728154
ATSp3	811.2269849	15128.22323
nBonds	14	163
nHBAcc	0	66

表 **8**　决策变量取值范围

决策变量	最优值
minsOH	9.2321
maxsOH	9.6231
SsOH	18.9218
nHsOH	2
apol	46.9831
fragC	1320.2341
hmin	0.2172
McGowan_Volume	2.3312
VABC	310.2329
VP-0	9.9290
maxHsOH	0.6218
WTPT-1	41.2873
ETA_Eta_R	30.2412
VP-1	7.9821
SP-1	9.3788
CrippenMR	85.1299
VAdjMat	5.2712
ATSp3	2230.2873
nBonds	23
nHBAcc	0

雀搜索每次迭代增加 1。对于麻雀搜索算法最大迭代次数，我们设置 50 次。

8.3.2 优化结果

我们定义每次迭代所展示的 PCI50 值为当前迭代及之前所得到的最优结果。如图19所示，经过约 30 次的迭代，PCI50 值收敛，收敛至 9.36。此时所对应的决策变量参数如表 8所示。

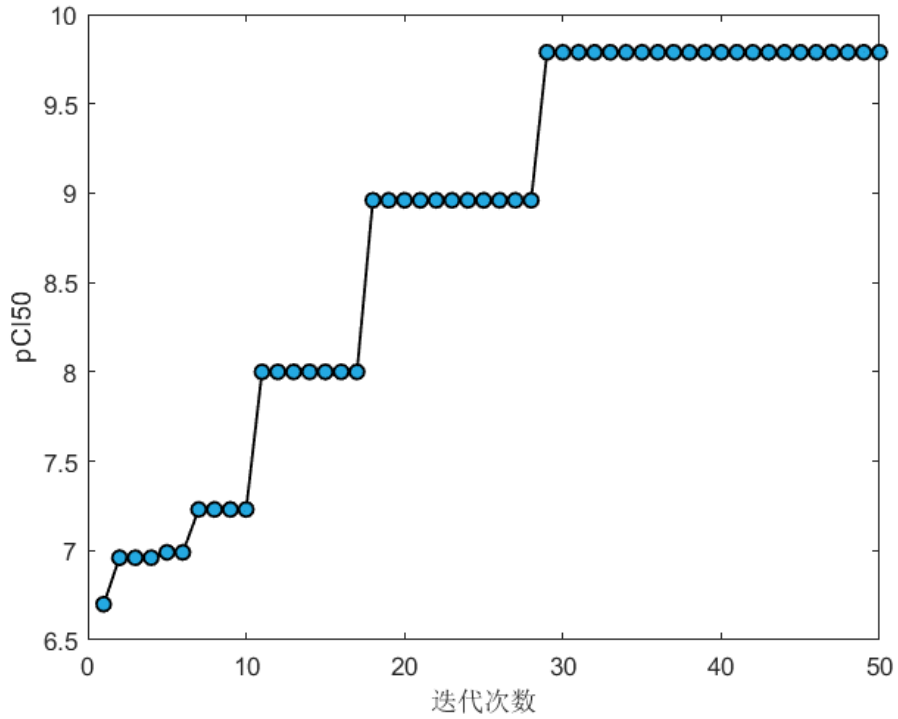

图 **19**　麻雀搜索迭代过程

九、 模型的评价和推广

9.1 模型评价

9.1.1 模型优点

- 集成评分降维法：通过结合嵌入法、包装法等多种降维方法进行特征筛选，同时对每个特征都有打分权值输出，避免单个学习器的偏差，针对不同模型的打分筛选结果进行交叉评估，极大地保证了预测结果的合理性与完整性。
- 线性融合预测法：多次实验证明，线性融合预测法具有良好的鲁棒性，预测评分准确性均略高于或等于其中最优学习器的预测结果。
- 多标签分类：预测模型保留了标签之间的相关性，深入挖掘 ADMET 五方面性质之间的关联性。

- 基于限制条件的决策变量选择：定位到同时满足多方面有约束的决策变量，更加贴近实际应用场景。
- 混合整数规划：适用于多种数据类型，贴近实际应用中的整数和非整数混合优化的问题。
- 利用惩罚函数法解决非线性约束：化简目标优化函数，可通过惩罚函数使新的目标函数变差而被舍弃，将带约束的优化函数转换为一个优化目标。

9.1.2 模型缺点

- 模型较为复杂，需要较长调参时间。
- 由于专业及时间限制，本问尚未对制药过程中各种变量的生物学本质做深入分析。大多数工作基于数据本身进行，后续若能深度融合专业相关知识，可以对该问题进行更加深入的理解及分析。
- 训练样本数较少，后续需要更多数据对其进行修正。

9.2 模型推广

- 本文提出的方法和模型可推广应用于其他生物或化学研究流程中的回归预测、分类问题，以及优化问题。
- 数据预处理及特征降维等通用数据处理步骤给出了清晰的脉络及参考思路，对后续深入进行相关研究有较强的改进价值。

参考文献

[1] Swersky K , Duvenaud D , Snoek J , et al. Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces[J]. Statistics, 2014.

[2] James Bergstra, R. Bardenet, Yoshua Bengio, Balázs Kégl. Algorithms for Hyper-Parameter Opti- mization. 25th Annual Conference on Neural Information Processing Systems (NIPS 2011), Dec 2011, Granada, Spain.

[3] Hutter F , Hoos H H , Leytonbrown K . Sequential model-based optimization for general algorithm configuration. , 2012.

[4] Ludovic Le Hégarat, Sylvie Huet, Valérie Fessard, A co-culture system of human intestinal Caco-2 cells and lymphoblastoid TK6 cells for investigating the genotoxicity of oral compounds, Mutagenesis, Volume 27, Issue 6, November 2012, Pages 631–636.

[5] Gller A H , Kuhnke L , Montanari F , et al. Bayer's in silico ADMET platform: a journey of machine learning over the past two decades[J]. Drug Discovery Today, 2020, 25(9):1702-1709.

[6] Bai, J. , S. Kong , and C. Gomes . "Disentangled Variational Autoencoder based Multi-Label Classification with Covariance-Aware Multivariate Probit Model." Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence, IJCAI-PRICAI-20 2020.

[7] Xue J , Shen B . A novel swarm intelligence optimization approach: sparrow search algorithm[J]. Systems science Control Engineering An Open Access Journal, 2020, 8(1):22-34.

附录 A 主要源代码

相关性分析代码：

```
梁：
# 互信息
result_mic = MIC(X,y.astype(str))
np.savetxt('MIC.csv', result_mic, delimiter=',')

梁：

# 正态性检验
for i in range(X_.shape[1]):
print(stats.kstest(X_.iloc[:,i], 'norm'))

# p值都很小，说明6个指标均不符合正态分布pvalue

# # 正态性检验画图分析
fig = plt.figure(figsize = (30,15))
for i in range(6):
ax2 = fig.add_subplot(2,3,i+1)
X_.iloc[:,i].hist(bins=50,ax = ax2)
X_.iloc[:,i].plot(kind = 'kde', secondary_y=True,ax = ax2)
plt.grid()
ax2.set_title(X_.columns[i])

plt.savefig("前6个特征正态检验绘图.jpg")
plt.show()

梁：
# 相关系数
# 因为所有指标均未通过正态性检验，即不服从正态分布，故计算斯皮尔曼相关系数
# stats.spearmanr(data)
plt.figure(figsize=(250, 120))
_X=data.loc[:,"nAcid":"PCI50"]
column = _X.columns.tolist()
mcorr = _X[column].corr(method="spearman")
mask = np.zeros_like(mcorr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g = sns.heatmap(mcorr,
mask=mask,
cmap=cmap,
square=True,
annot=False,
fmt='0.2f')
```

```
plt.savefig("热力图.jpg")
plt.show()

梁:
"""随机森林特征打分"""
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.preprocessing import MinMaxScaler

file_path = r"E:\华为杯\2021年D题\第四问筛特征.xlsx"
data = pd.read_excel(file_path)
# data=data.dropna(axis=1)

# list = ["Caco_2","CYP3A4","hERG","HOB","MN"]
# for i in range(len(list)):

# 划分特征和目标
x = data.loc[:,"nAcid":"Zagreb"]
y = data.loc[:,"PCI50"]



estimator = RFR(n_estimators=500,
min_samples_split=5,
max_features=10,
random_state=42)


estimator.fit(x,y)

cval = cross_val_score(estimator, x, y, scoring='roc_auc', cv=10)


importance = estimator.feature_importances_
importance = MinMaxScaler().fit_transform(importance.reshape(-1, 1))
np.savetxt('NO4_RF.csv', importance, delimiter=',')



梁:
"""XGBOOST特征打分"""
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import r2_score
import pandas as pd
```

```
from sklearn.metrics import mean_squared_error as MSE
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import csv


# 准备数据集

file_path = r"E:\华为杯\2021年D题\第三问数据.xlsx"
data = pd.read_excel(file_path)
list = data.loc[:,"Caco_2":"MN"].columns



for i in range(len(list)):
# data=data.dropna(axis=1)

# 划分特征和目标
x = data.loc[:,"nAcid":"Zagreb"]
y = data.loc[:,list[i]]

# 根据比例划分训练集测试集
# X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.3)

# 使用类DMatrix读取数据
# dtrain = xgb.DMatrix(X_train,Y_train)
# dtest = xgb.DMatrix(X_test,Y_test)
dtotal = xgb.DMatrix(x,y)

# 参数
param = {"silent":False,
   "objective":"binary:logistic", # 用于fenlei
   "eta":0.1, # 学习率
   "subsample":1,
   "max_depth":4,
   "gamma":0,
   "lamda":1,
   "alpha":0,
   "colsample_bytree":1,
   "cosample_bylevel":1,
   "colsample_bynode":1,
   "nfold":5
}
num_round = 200 #树的个数
n_fold = 5 #n折交叉训练

# 训练
```

```
bst = xgb.train(param,dtotal,num_round)

# 测试并查看R方和MSE
# print(r2_score(Y_test,bst.predict(dtest)))
# print(MSE(Y_test,bst.predict(dtest)))

# n_fold折交叉验证
cv_result = xgb.cv(params=param,dtrain=dtotal,num_boost_round=num_round)
# print(cv_result)

XGBOOST_importance_gain = bst.get_score(importance_type='gain')
XGBOOST_importance_gain = pd.DataFrame([XGBOOST_importance_gain])

XGBOOST_importance_gain.to_csv("XG__{}.csv".format(list[i]))
```

线性融合预测以及贝叶斯优化代码：

```
"""线性融合+贝叶斯调参"""
import pandas as pd
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.ensemble import RandomForestRegressor as RFR
from bayes_opt import BayesianOptimization
import numpy as np
import joblib
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import accuracy_score



# 读取训练集和测试集的文件
train_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\第二问数据.xlsx"
test_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\test_use.xlsx"
train_data = pd.read_excel(train_file_path)
test_data = pd.read_excel(test_file_path)

# 划分特征和目标
x = train_data.iloc[:,:-1]
y = train_data.iloc[:,-1]

# 根据比例划分训练集验证集
X_train,X_eval,Y_train,Y_eval = train_test_split(x,y,test_size=0.2,shuffle=True)

X_train.to_excel("train_dataset.xlsx")
X_eval.to_excel("eval_dataset.xlsx")
Y_train.to_excel("train_target.xlsx")
```

```python
Y_eval.to_excel("eval_target.xlsx")

# 数据读取
train_data_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\train_dataset.xlsx"
eval_data_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\eval_dataset.xlsx"
train_target_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\train_target.xlsx"
eval_target_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\eval_target.xlsx"
test_data_file_path = r"E:\learn_pytorch\learn_pytorch\XGboost_Bayesian\test_use.xlsx"


train_data = pd.read_excel(train_data_file_path,index_col=0)
eval_data = pd.read_excel(eval_data_file_path,index_col=0)
train_target = pd.read_excel(train_target_file_path,index_col=0)
eval_target = pd.read_excel(eval_target_file_path,index_col=0)
test_data = pd.read_excel(test_data_file_path)



def get_bast_parameters(ret):
"""这个函数用来获取调参输出的最佳超参数"""
target_max_axis = np.argmax(np.max(ret.space.target.reshape(len(ret.space.target), -1),
    axis=1)) # 最大target所在轮数
best_parameters = ret.space.params[target_max_axis, :]



return best_parameters



"""———————————————————————————————————————————————————————————此处是随机森林部分———
def rfr_cv_object(n_estimators, min_samples_split, max_features, data,
targets):
"""随机森林交叉验证

关注的超参空间:
n_estimators, min_samples_split, and max_features
目标:
最大化指标roc_auc，交叉验证的平均作为最终的模型效果
"""
estimator = RFR(n_estimators=n_estimators,
min_samples_split=min_samples_split,
max_features=max_features,
random_state=42)
cval = cross_val_score(estimator, data, targets, scoring='neg_mean_squared_error', cv=10)
return cval.mean()
```

```
def optimize_rfr(data, targets):
def rfr_crossval(n_estimators, min_samples_split, max_features):
"""rfc_cv的二次封装，保证:
1.n_estimators和min_samples_split为整数
2.避免max_features在(0, 1)范围之外
"""
return rfr_cv_object(
n_estimators=int(n_estimators),
min_samples_split=int(min_samples_split),
max_features=int(max_features),
data=data,
targets=targets,
)

optimizer = BayesianOptimization(
f=rfr_crossval,
# 超参空间
pbounds={
   "n_estimators": (100, 500),
   "min_samples_split": (2, 25),
   "max_features": (5, 20),
},
random_state=42,
verbose=2)
# 2个初始化点和10轮优化，共12轮
optimizer.maximize(init_points=2, n_iter=1)

print("rf Final result:", optimizer.max)
return optimizer


ret_rfr = optimize_rfr(train_data, train_target)


# 获得调参结果
rfr_best_params = get_bast_parameters(ret_rfr)
rfr_best_max_features = int(rfr_best_params[0])
rfr_beat_min_samples_split = int(rfr_best_params[1])
rfr_best_n_estimators = int(rfr_best_params[2])

# 用以上参数再训练模型
Final_RFR =
    RFR(n_estimators=rfr_best_n_estimators,min_samples_split=rfr_beat_min_samples_split,max_features=rfr_best_m
Final_RFR.fit(train_data, train_target)

# 再训练模型在验证集上的表现
predict_rf = Final_RFR.predict(eval_data)
```

```
# 在验证集上的表现
RMSE_RFR = np.sqrt(mean_squared_error(eval_target,predict_rf))



# 将训练的模型保存到磁盘（value=模型名），默认当前文件夹下
joblib.dump(filename="RFR_model_test",value=Final_RFR)

# 下载本地模型
test_model_rfr = joblib.load(filename="RFR_model_test")

# 用下载下来的模型进行预测
rf_pred = test_model_rfr.predict(test_data)



"""————————————————————————————————————————————————————————————————————————————————————————————————————————————————此处是XGboost部分——

# 使用类DMatrix转换数据
dtrain = xgb.DMatrix(train_data,train_target)
deval = xgb.DMatrix(eval_data,eval_target)
dtest = xgb.DMatrix(test_data)

def xg_cv_object(eta,subsample,gamma, num_round,data):
"""XGBoost交叉验证

关注的超参空间：
eta,subsample,gamma,max_depth, num_round
目标：
最小化MSE，交叉验证的平均作为最终的模型效果
"""



# 参数
param = {"silent": False,
   "objective": "reg:linear", # 用于回归
   "eta": eta, # 学习率
   "subsample": subsample,
   "max_depth": 5,# 在该数据集下的最高表现？
   "gamma": gamma,

   "alpha": 0,
   "colsample_bytree": 1,
   "cosample_bylevel": 1,
   "colsample_bynode": 1,
   "nfold": 5
```

```
}
num_round = num_round # 树的个数



# estimator = xgb.train(param,dtrain,num_round)
cval = xgb.cv(params=param,dtrain=data,num_boost_round=num_round)
return -1*cval.loc[num_round-1,"test-rmse-mean"]

def optimize_xg(dtotal):
def xg_crossval(eta,subsample,gamma, num_round):
"""bst_cv的二次封装，保证：
1.eta,sunsample在（0, 1）之间
2.gamma和max_depth都很重要，最好两个只调一个，一起调效果反而差
"""
return xg_cv_object(
eta=max(min(eta, 0.999), 1e-3),
subsample=max(min(subsample, 0.999), 1e-3),
# max_depth=int(max_depth),
gamma=gamma,
num_round=int(num_round),
data=dtotal
)

optimizer = BayesianOptimization(
f=xg_crossval,
# 超参空间
pbounds={
   "eta": (0.001, 0.2),
   "subsample": (0.6, 1),
   # "max_depth": (2, 20),
   "gamma": (0,50),
   "num_round": (150,300)
},
random_state=42,
verbose=2)
# 2个初始化点和10轮优化，共12轮
optimizer.maximize(init_points=2, n_iter=1)

print("xgboost Final result:", optimizer.max)
return optimizer



ret_xg = optimize_xg(dtrain)

# 获得调参结果
xg_best_params = get_bast_parameters(ret_xg)
```

```python
xg_best_eta = xg_best_params[0]
xg_beat_gamma = xg_best_params[1]
xg_best_num_round = int(xg_best_params[2])
xg_best_subsample = xg_best_params[3]

# 再训练
param_xg = {"silent": False,
   "objective": "reg:linear", # 用于回归
   "eta": xg_best_eta, # 学习率
   "subsample": xg_best_subsample,
   "max_depth": 5,# 在该数据集下的最高表现?
   "gamma": xg_beat_gamma,
   "lamda": 1,
   "alpha": 0,
   "colsample_bytree": 1,
   "cosample_bylevel": 1,
   "colsample_bynode": 1,
   "nfold": 5
}
num_round = xg_best_num_round # 树的个数


Final_xg = xgb.train(param_xg,dtrain,num_round)
# 再训练的模型在验证集上的表现
predict_xg = Final_xg.predict(deval)

# 在验证集上的RMSE
RMSE_XG = np.sqrt(mean_squared_error(eval_target,predict_xg))



# 将训练的模型保存到磁盘（value=模型名），默认当前文件夹下
joblib.dump(filename="xg_model_test",value=Final_xg)


# 下载本地模型
test_model_xg = joblib.load(filename="xg_model_test")

"""用下载下来的模型进行预测，在此之前先将其转成dtest"""
xg_pred = test_model_xg.predict(dtest)




"""——————————————————————————————————————————————————————————此处是lightGBM部分——
```

```
# 数据格式转换

lgb_train = lgb.Dataset(train_data,train_target)
lgb_eval = lgb.Dataset(eval_data,eval_target)

def gbm_cv_object(learning_rate,num_leaves,feature_fraction,bagging_fraction):
"""lightGBM交叉验证

关注的超参空间:
learning_rate,num_leaves,feature_fraction,lambda_l1,lambda_l2, bagging_fraction
目标:
最小化RMSE, 交叉验证的平均作为最终的模型效果
"""



# 参数
params = {
   "boosting_type": "gbdt", # 设置提升类型
   "objective": "regression", # 目标函数, 此处是回归
   "metric": {"l2_root"}, # 评估函数, 支持使用多个, l2是MSE
   "num_leaves": num_leaves, # 叶子节点数
   "learning_rate": learning_rate,
   "feature_fraction": feature_fraction, # 建树的特征选择比例
   "bagging_fraction": bagging_fraction, # 建树的样本采集比例
   "bagging_freq": 5, # 意味着每k次迭代执行bagging
   # "reg_alpha": l1,
   # "reg_lambda": l2,
   "verbose": -1

}
boost_round = 1000 # 迭代次数
early_stop_rounds = 50 # 验证集如果在early_stop_rounds轮中未能提高, 则提前停止



# estimator = xgb.train(param,dtrain,num_round)
cval = lgb.cv(params=params,train_set=lgb_train,num_boost_round=boost_round,
nfold=10,stratified=False,early_stopping_rounds=early_stop_rounds,
verbose_eval=20,seed=0,metrics="rmse")
return -1*cval["rmse-mean"][-1]



def optimize_gbm(lgb_total):
def gbm_crossval(learning_rate,num_leaves,feature_fraction,bagging_fraction):
"""bst_cv的二次封装, 保证:
1.learning_rate越小越好
2.num_leaves控制树的复杂程度, 越打大越复杂
```

```
3.feature_fraction特征选择比例，控制过拟合，别太小了
4.bagging_fraction提速度，并降低过拟合

"""
return gbm_cv_object(
learning_rate=max(min(learning_rate, 0.999), 1e-3),
num_leaves=int(num_leaves),
feature_fraction=max(min(feature_fraction, 0.999), 1e-3),
# l1=max(min(l1, 0.999), 1e-3),
# l2=max(min(l2, 0.999), 1e-3),
bagging_fraction=max(min(bagging_fraction, 0.999),1e-3)



)

optimizer = BayesianOptimization(
f=gbm_crossval,
# 超参空间
pbounds={
   "learning_rate": (0.001, 0.05),
   "num_leaves": (30, 50),
   "feature_fraction": (0.8, 1),
   # "l1":(0,1),
   # "l2":(0,1)
   "bagging_fraction": (0.8,1)


},
random_state=42,
verbose=2)
# 2个初始化点和10轮优化，共12轮
optimizer.maximize(init_points=2, n_iter=1)

print("Final result:", optimizer.max)
return optimizer

ret_gbm = optimize_gbm(lgb_train)

# 获得调参结果
gbm_best_params = get_bast_parameters(ret_gbm)
gbm_best_bagging_fraction = gbm_best_params[0]
gbm_beat_feature_fraction = gbm_best_params[1]
# best_l1 = best_params[2]
# best_l2 = best_params[3]
gbm_best_learning_rate = gbm_best_params[2]
gbm_best_num_leaves = int(gbm_best_params[3])
```

```
# 再训练
params_gbm = {
   "boosting_type": "gbdt", # 设置提升类型
   "objective": "regression", # 目标函数，此处是回归
   "metric": {"l2_root"}, # 评估函数，支持使用多个，l2是MSE
   "num_leaves": gbm_best_num_leaves, # 叶子节点数
   "learning_rate": gbm_best_learning_rate,
   "feature_fraction": gbm_beat_feature_fraction, # 建树的特征选择比例
   "bagging_fraction": gbm_best_bagging_fraction, # 建树的样本采集比例
   "bagging_freq": 5, # 意味着每k次迭代执行bagging
   # "reg_alpha": l1,
   # "reg_lambda": l2,
   "verbose": -1

}
boost_round = 1000 # 迭代次数
early_stop_rounds = 50 # 验证集如果在early_stop_rounds轮中未能提高，则提前停止

# 再训练模型
results = {}
final_gbm = lgb.train(params_gbm,lgb_train,num_boost_round=boost_round,
evals_result=results)

# 再训练模型在验证集上的表现
predict_gbm= final_gbm.predict(eval_data,num_iteration=final_gbm.best_iteration)
# cval = lgb.cv(params=params,train_set=lgb_train,num_boost_round=boost_round,
#                  nfold=10,stratified=False,early_stopping_rounds=early_stop_rounds,
#                  verbose_eval=20,seed=0,metrics="rmse")
RMSE_gbm = np.sqrt(mean_squared_error(eval_target,predict_gbm))


# 将训练的模型保存到磁盘（value=模型名），默认当前文件夹下
joblib.dump(filename="test_gbm_model",value=final_gbm)

# 下载本地模型
test_model_gbm = joblib.load(filename="test_gbm_model")


# 模型预测
gbm_pred= test_model_gbm.predict(test_data,num_iteration=test_model_gbm.best_iteration)


print("随机森林在验证集上的RMSE: ",RMSE_RFR)
print("xgboost在验证集上的RMSE: ",RMSE_XG)
print("lightGBM在验证集上的RMSE:",RMSE_gbm)
```

```
print("*"*200)

print("随机森林模型预测结果为: ",rf_pred)
print("xgboost模型预测结果为: ",xg_pred)
print("lightGBM的预测结果为: ",gbm_pred)

"""————————————————————————————————————————————————————————————————————————————开始线性融合了————————

# 检验一下
linear_fusion_predict = (predict_rf + predict_xg + predict_gbm)/3
RMSE_linear_fusion = np.sqrt(mean_squared_error(eval_target,linear_fusion_predict))

print("*"*200)
print("融合模型在验证集上的RMSE:",RMSE_linear_fusion)




linear_fusion_result = (rf_pred + xg_pred +gbm_pred)/3
print("融合模型的预测结果:",linear_fusion_result)
```

MPVAE 分类代码:

```
import argparse
from train import train
from test import test


parser = argparse.ArgumentParser()
parser.add_argument('-dataset', "--dataset", default='ours', type=str, help='dataset name')
parser.add_argument('-cp', "--checkpoint_path",
    default='./model/model_ours/lr-0.00075_lr-decay_0.50_lr-times_3.0_nll-0.10_l2-1.00_c-200.00/vae-15',
    type=str, help='The path to a checkpoint from which to fine-tune.')

parser.add_argument('-dd', "--data_dir", default='./data/ours/ours.npy', type=str, help='The
    path of input observation data')
parser.add_argument('-train_idx', "--train_idx", default='./data/ours/train_index.npy',
    type=str, help='The path of training data index')
parser.add_argument('-valid_idx', "--valid_idx", default='./data/ours/valid_index.npy',
    type=str, help='The path of validation data index')
parser.add_argument('-test_idx', "--test_idx", default='./data/ours/test_index.npy',
    type=str, help='The path of testing data index')

'''
parser.add_argument('-dd', "--data_dir", default='./data/mirflickr/mirflickr_data.npy',
    type=str, help='The path of input observation data')
parser.add_argument('-train_idx', "--train_idx",
```

```
    default='./data/mirflickr/mirflickr_train_idx.npy', type=str, help='The path of training
    data index')
parser.add_argument('-valid_idx', "--valid_idx",
    default='./data/mirflickr/mirflickr_test_idx.npy', type=str, help='The path of
    validation data index')
parser.add_argument('-test_idx', "--test_idx",
    default='./data/mirflickr/mirflickr_val_idx.npy', type=str, help='The path of testing
    data index')
'''
'''
parser.add_argument('-dd', "--data_dir", default='./data/scene/scene_data.npy', type=str,
    help='The path of input observation data')
parser.add_argument('-train_idx', "--train_idx", default='./data/scene/scene_train_idx.npy',
    type=str, help='The path of training data index')
parser.add_argument('-valid_idx', "--valid_idx", default='./data/scene/scene_test_idx.npy',
    type=str, help='The path of validation data index')
parser.add_argument('-test_idx', "--test_idx", default='./data/scene/scene_val_idx.npy',
    type=str, help='The path of testing data index')
'''

parser.add_argument('-bs', "--batch_size", default=100, type=int, help='the number of data
    points in one minibatch')
parser.add_argument('-tbs', "--test_batch_size", default=128, type=int, help='the number of
    data points in one testing or validation batch')
parser.add_argument('-lr', "--learning_rate", default=0.00075, type=float, help='initial
    learning rate')
parser.add_argument('-epoch', "--max_epoch", default=20, type=int, help='max epoch to train')
parser.add_argument('-wd', "--weight_decay", default=1e-5, type=float, help='weight decay
    rate')
parser.add_argument('-lrdr', "--lr_decay_ratio", default=0.2, type=float, help='The decay
    ratio of learning rate')
parser.add_argument('-lrdt', "--lr_decay_times", default=3.0, type=float, help='The number
    of times learning rate decays')
parser.add_argument('-ntest', "--n_test_sample", default=10000, type=int, help='The sampling
    times for the testing')
parser.add_argument('-ntrain', "--n_train_sample", default=100, type=int, help='The sampling
    times for the training')
parser.add_argument('-z', "--z_dim", default=1000, type=int, help='z dimention: the number
    of the independent normal random variables in DMS the rank of the residual covariance
    matrix')

parser.add_argument('-label_dim', "--label_dim", default=5, type=int, help='the number of
    labels in current training')
parser.add_argument('-latent_dim', "--latent_dim", default=3, type=int, help='the number of
    labels in current training')
parser.add_argument('-meta_offset', "--meta_offset", default=0, type=int, help='the offset
    caused by meta data')
```

```
parser.add_argument('-feat_dim', "--feature_dim", default=51, type=int, help='the
    dimensionality of the features')

parser.add_argument('-se', "--save_epoch", default=1, type=int, help='epochs to save the
    checkpoint of the model')
parser.add_argument('-max_keep', "--max_keep", default=3, type=int, help='maximum number of
    saved model')
parser.add_argument('-check_freq', "--check_freq", default=120, type=int, help='checking
    frequency')

parser.add_argument('-nll_coeff', "--nll_coeff", default=0.1, type=float, help='nll_loss
    coefficient')
parser.add_argument('-l2_coeff', "--l2_coeff", default=1.0/40, type=float, help='l2_loss
    coefficient')
parser.add_argument('-c_coeff', "--c_coeff", default=200., type=float, help='c_loss
    coefficient')
parser.add_argument('-scale_coeff', "--scale_coeff", default=1.0, type=float,
    help='mu/logvar scale coefficient')
parser.add_argument('-keep_prob', "--keep_prob", default=0.5, type=float, help='drop out
    rate')
parser.add_argument('-resume', "--resume", action='store_true', help='whether to resume a
    ckpt')
parser.add_argument('-write_to_test_sh', "--write_to_test_sh", action='store_true',
    help='whether to modify test.sh')
parser.add_argument('-mode', "--mode",default="train", type=str, help='training/test mode')
parser.add_argument('-r_sigma', "--residue_sigma", default='', type=str, help='what sigma r
    to use')

args = parser.parse_args()

if __name__ == "__main__":
if args.mode == 'train':
train(args)
elif args.mode == 'test':
test(args)
else:
raise ValueError("mode %s is not supported." % args.mode)
import torch
import torch.nn.functional as F
from torch import nn, optim
from torch.utils.tensorboard import SummaryWriter
import numpy as np
import sys
import os
import datetime
from copy import copy, deepcopy
import evals
```

```
from utils import build_path, get_label, get_feat
from model import VAE, compute_loss
import pandas as pd
import csv
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
sys.path.append('./')
THRESHOLDS =
    [0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.10,0.15,0.20,0.25,0.30,0.35,0.40,0.45,0.50,0.55,0.60,0.6

METRICS = ['ACC', 'HA', 'ebF1', 'miF1', 'maF1', 'meanAUC', 'medianAUC', 'meanAUPR',
    'medianAUPR', 'meanFDR', 'medianFDR', 'p_at_1', 'p_at_3', 'p_at_5']

def train(args):
print('reading npy...')
np.random.seed(4) # set the random seed of numpy
data = np.load(args.data_dir) #load data from the data_dir
train_idx = np.load(args.train_idx) #load the indices of the training set
valid_idx = np.load(args.valid_idx) #load the indices of the validation set
test_idx = np.load(args.test_idx)
labels = get_label(data, train_idx, args.meta_offset, args.label_dim) #load the labels of
    the training set

print("data:",data)
print(data.shape)
print("train_i:",train_idx)
print("test_i:",test_idx)
print("valid_i:", valid_idx)
#print("label_shape", data.train.size)
print("label:", labels)
print("label_shape",torch.from_numpy(labels).shape)
print("positive label rate:", np.mean(labels)) #print the rate of the positive labels in the
    training set
param_setting =
    "lr-{}_lr-decay_{:.2f}_lr-times_{:.1f}_nll-{:.2f}_l2-{:.2f}_c-{:.2f}".format(args.learning_rate,
    args.lr_decay_ratio, args.lr_decay_times, args.nll_coeff, args.l2_coeff, args.c_coeff)
build_path('summary/{}/{}'.format(args.dataset, param_setting))
build_path('model/model_{}/{}'.format(args.dataset, param_setting))
summary_dir = 'summary/{}/{}'.format(args.dataset, param_setting)
model_dir = 'model/model_{}/{}'.format(args.dataset, param_setting)

one_epoch_iter = np.ceil(len(train_idx) / args.batch_size) # compute the number of
    iterations in each epoch
n_iter = one_epoch_iter * args.max_epoch
print("one_epoch_iter:", one_epoch_iter)
print("total_iter:", n_iter)
```

```
print("showing the parameters...")
print(args)

writer = SummaryWriter(log_dir=summary_dir)

print('building network...')

#building the model
vae = VAE(args).to(device)
vae.train()

#log the learning rate
writer.add_scalar('learning_rate', args.learning_rate)

#use the Adam optimizer
optimizer = optim.Adam(vae.parameters(), lr=args.learning_rate, weight_decay=1e-5)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, one_epoch_iter * (args.max_epoch /
    args.lr_decay_times), args.lr_decay_ratio)

if args.resume:
vae.load_state_dict(torch.load(args.checkpoint_path))
current_step = int(args.checkpoint_path.split('/')[-1].split('-')[-1])
print("loaded model: %s" % args.label_checkpoint_path)
else:
current_step = 0

# smooth means average. Every batch has a mean loss value w.r.t. different losses
smooth_nll_loss=0.0 # label encoder decoder cross entropy loss
smooth_nll_loss_x=0.0 # feature encoder decoder cross entropy loss
smooth_c_loss = 0.0 # label encoder decoder ranking loss
smooth_c_loss_x=0.0 # feature encoder decoder ranking loss
smooth_kl_loss = 0.0 # kl divergence
smooth_total_loss=0.0 # total loss
smooth_macro_f1 = 0.0 # macro_f1 score
smooth_micro_f1 = 0.0 # micro_f1 score
#smooth_l2_loss = 0.0

best_loss = 1e10
best_iter = 0
best_macro_f1 = 0.0 # best macro f1 for ckpt selection in validation
best_micro_f1 = 0.0 # best micro f1 for ckpt selection in validation
best_acc = 0.0 # best subset acc for ckpt selction in validation

temp_label=[]
temp_indiv_prob=[]
```

```
best_test_metrics = None


# training the model
for one_epoch in range(args.max_epoch):
print('epoch '+str(one_epoch+1)+' starts!')
np.random.shuffle(train_idx) # random shuffle the training indices

for i in range(int(len(train_idx)/float(args.batch_size))+1):
optimizer.zero_grad()
start = i*args.batch_size
end = min(args.batch_size*(i+1), len(train_idx))
input_feat = get_feat(data,train_idx[start:end], args.meta_offset, args.label_dim,
    args.feature_dim) # get the NLCD features

input_label = get_label(data,train_idx[start:end], args.meta_offset, args.label_dim) # get
    the prediction labels
input_feat, input_label = torch.from_numpy(input_feat).to(device),
    torch.from_numpy(input_label)
input_label = deepcopy(input_label).float().to(device)

label_out, label_mu, label_logvar, feat_out, feat_mu, feat_logvar = vae(input_label,
    input_feat)

#print(label_out)
#train the model for one step and log the training loss
if args.residue_sigma == "random":
r_sqrt_sigma = torch.from_numpy(np.random.uniform(-np.sqrt(6.0/(args.label_dim+args.z_dim)),
    np.sqrt(6.0/(args.label_dim+args.z_dim)), (args.label_dim, args.z_dim))).to(device)
total_loss, nll_loss, nll_loss_x, c_loss, c_loss_x, kl_loss, indiv_prob =
    compute_loss(input_label, label_out, label_mu, label_logvar, feat_out, feat_mu,
    feat_logvar, r_sqrt_sigma, args)
else:

total_loss, nll_loss, nll_loss_x, c_loss, c_loss_x, kl_loss, indiv_prob =
    compute_loss(input_label, label_out, label_mu, label_logvar, feat_out, feat_mu,
    feat_logvar, vae.r_sqrt_sigma, args)




total_loss.backward()
optimizer.step()
scheduler.step()

train_metrics = evals.compute_metrics(indiv_prob.cpu().data.numpy(),
    input_label.cpu().data.numpy(), 0.5, all_metrics=False)
```

```
macro_f1, micro_f1 = train_metrics['maF1'], train_metrics['miF1']

smooth_nll_loss += nll_loss
smooth_nll_loss_x += nll_loss_x
#smooth_l2_loss += l2_loss
smooth_c_loss += c_loss
smooth_c_loss_x += c_loss_x
smooth_kl_loss += kl_loss
smooth_total_loss += total_loss
smooth_macro_f1 += macro_f1
smooth_micro_f1 += micro_f1

temp_label.append(input_label) #log the labels
temp_indiv_prob.append(indiv_prob) #log the individual prediction of the probability on each
    label

current_step += 1
lr = optimizer.param_groups[0]['lr']
writer.add_scalar('learning_rate', lr, current_step)

if current_step % args.check_freq==0: #summarize the current training status and print them
    out
nll_loss = smooth_nll_loss / float(args.check_freq)
nll_loss_x = smooth_nll_loss_x / float(args.check_freq)
#l2_loss = smooth_l2_loss / float(args.check_freq)
c_loss = smooth_c_loss / float(args.check_freq)
c_loss_x = smooth_c_loss_x / float(args.check_freq)
kl_loss = smooth_kl_loss / float(args.check_freq)
total_loss = smooth_total_loss / float(args.check_freq)
macro_f1 = smooth_macro_f1 / float(args.check_freq)
micro_f1 = smooth_micro_f1 / float(args.check_freq)

# temp_indiv_prob = np.reshape(np.array(temp_indiv_prob), (-1))
# temp_label = np.reshape(np.array(temp_label), (-1))

#temp_indiv_prob = np.reshape(temp_indiv_prob,(-1, args.label_dim))
#temp_label = np.reshape(temp_label,(-1, args.label_dim))

time_str = datetime.datetime.now().isoformat()
print("step=%d %s\nlr=%.6f\nmacro_f1=%.6f,
    micro_f1=%.6f\nnll_loss=%.6f\tnll_loss_x=%.6f\nc_loss=%.6f\tc_loss_x=%.6f\tkl_loss=%.6f\ntotal_loss=%.6f
    % (current_step, time_str, lr, macro_f1, micro_f1, nll_loss*args.nll_coeff,
    nll_loss_x*args.nll_coeff, c_loss*args.c_coeff, c_loss_x*args.c_coeff, kl_loss,
    total_loss))
#print("step=%d %s\nlr=%.6f\nmacro_f1=%.6f,
    micro_f1=%.6f\nnll_loss=%.6f\tnll_loss_x=%.6f\tl2_loss=%.6f\nc_loss=%.6f\tc_loss_x=%.6f\tkl_loss=%.6f\nt
    % (current_step, time_str, lr, macro_f1, micro_f1, nll_loss*args.nll_coeff,
```

```
    nll_loss_x*args.nll_coeff, l2_loss*args.l2_coeff, c_loss*args.c_coeff,
    c_loss_x*args.c_coeff, kl_loss, total_loss))
temp_indiv_prob=[]
temp_label=[]

smooth_nll_loss = 0
smooth_nll_loss_x = 0
#smooth_l2_loss = 0
smooth_c_loss = 0
smooth_c_loss_x = 0
smooth_kl_loss = 0
smooth_total_loss = 0
smooth_macro_f1 = 0
smooth_micro_f1 = 0

if current_step % int(one_epoch_iter*args.save_epoch)==0: #exam the model on validation set
print("--------------------------------")
# exam the model on validation set
current_loss, val_metrics = valid(data, vae, writer, valid_idx, current_step, args)
macro_f1, micro_f1 = val_metrics['maF1'], val_metrics['miF1']

# select the best checkpoint based on some metric on the validation set
# here we use macro F1 as the selection metric but one can use others
if val_metrics['maF1'] > best_macro_f1:
print('macro_f1:%.6f, micro_f1:%.6f, nll_loss:%.6f, which is better than the previous best
    one!!!'%(macro_f1, micro_f1, current_loss))

best_loss = current_loss
best_iter = current_step

print('saving model')
torch.save(vae.state_dict(), model_dir+'/vae-'+str(current_step))
print('have saved model to ', model_dir)
print()

if args.write_to_test_sh:
test_sh_path = "script/run_test_%s.sh" % args.dataset
if os.path.exists(test_sh_path):
ckptFile = open(test_sh_path, "r")
command = []
for line in ckptFile:
arg_lst = line.strip().split(' ')
for arg in arg_lst:
if 'model/model_{}/lr-'.format(args.dataset) in arg:
command.append('model/model_{}/{}/vae-{}'.format(args.dataset, param_setting, best_iter))
else:
command.append(arg)
```

```
ckptFile.close()
else:
command = ("python main.py --data_dir %s --test_idx %s --label_dim %d --z_dim %d
    --feature_dim %d --nll_coeff %s --c_coeff %s --batch_size 64 --mode test -cp %s" %
    (args.data_dir, args.test_idx, args.label_dim, args.z_dim, args.feature_dim,
    args.nll_coeff, args.c_coeff, 'model/model_{}/{}/vae-{}'.format(args.dataset,
    param_setting, best_iter))).strip().split(' ')

ckptFile = open(test_sh_path, "w")
ckptFile.write(" ".join(command)+"\n")
ckptFile.close()
best_macro_f1 = max(best_macro_f1, val_metrics['maF1'])
best_micro_f1 = max(best_micro_f1, val_metrics['miF1'])
best_acc = max(best_acc, val_metrics['ACC'])

print("--------------------------------")

def valid(data, vae, summary_writer, valid_idx, current_step, args):
vae.eval()
print("performing validation...")

all_nll_loss = 0
all_l2_loss = 0
all_c_loss = 0
all_total_loss = 0

all_indiv_prob = []
all_label = []

real_batch_size=min(args.batch_size, len(valid_idx))
for i in range(int((len(valid_idx)-1)/real_batch_size)+1):
start = real_batch_size*i
end = min(real_batch_size*(i+1), len(valid_idx))

input_feat = get_feat(data,valid_idx[start:end], args.meta_offset, args.label_dim,
    args.feature_dim)
input_label = get_label(data,valid_idx[start:end], args.meta_offset, args.label_dim)
input_feat, input_label = torch.from_numpy(input_feat).to(device),
    torch.from_numpy(input_label)
input_label = deepcopy(input_label).float().to(device)

with torch.no_grad():
label_out, label_mu, label_logvar, feat_out, feat_mu, feat_logvar = vae(input_label,
    input_feat)
total_loss, nll_loss, nll_loss_x, c_loss, c_loss_x, kl_loss, indiv_prob =
    compute_loss(input_label, label_out, label_mu, label_logvar, feat_out, feat_mu,
    feat_logvar, vae.r_sqrt_sigma, args)
```

```
all_nll_loss += nll_loss*(end-start)
#all_l2_loss += l2_loss*(end-start)
all_c_loss += c_loss*(end-start)
all_total_loss += total_loss*(end-start)

for j in deepcopy(indiv_prob).cpu().data.numpy():
all_indiv_prob.append(j)
for j in deepcopy(input_label).cpu().data.numpy():
all_label.append(j)

# collect all predictions and ground-truths
all_indiv_prob = np.array(all_indiv_prob)
all_label = np.array(all_label)

Auc=[]
for j in range(0,5):
label_p1 = []
label_r1 = []
for i in all_indiv_prob:
label_p1.append(i[j])
for i in all_label:
label_r1.append(i[j])
fpr, tpr, thresholds = roc_curve(label_r1, label_p1, pos_label=1)
Auc.append(auc(fpr, tpr))
print("aucccccc:",Auc)
nll_loss = all_nll_loss/len(valid_idx)
l2_loss = all_l2_loss/len(valid_idx)
c_loss = all_c_loss/len(valid_idx)
total_loss = all_total_loss/len(valid_idx)

best_val_metrics = None
for threshold in THRESHOLDS:
val_metrics = evals.compute_metrics(all_indiv_prob, all_label, threshold, all_metrics=True)

if best_val_metrics == None:
best_val_metrics = {}
for metric in METRICS:
best_val_metrics[metric] = val_metrics[metric]
else:
for metric in METRICS:
if 'FDR' in metric:
best_val_metrics[metric] = min(best_val_metrics[metric], val_metrics[metric])
else:
best_val_metrics[metric] = max(best_val_metrics[metric], val_metrics[metric])

time_str = datetime.datetime.now().isoformat()
```

```
    acc, ha, ebf1, maf1, mif1 = best_val_metrics['ACC'], best_val_metrics['HA'],
        best_val_metrics['ebF1'], best_val_metrics['maF1'], best_val_metrics['miF1']

    # nll_coeff: BCE coeff, lambda_1
    # c_coeff: Ranking loss coeff, lambda_2
    print("*********************************************")
    print("valid results: %s\nacc=%.6f\tha=%.6f\texam_f1=%.6f, macro_f1=%.6f,
        micro_f1=%.6f\nnll_loss=%.6f\tc_loss=%.6f\ttotal_loss=%.6f" % (time_str, acc, ha, ebf1,
        maf1, mif1, nll_loss*args.nll_coeff, c_loss*args.c_coeff, total_loss))
    print("*********************************************")


    summary_writer.add_scalar('valid/nll_loss', nll_loss, current_step)
    summary_writer.add_scalar('valid/l2_loss', l2_loss, current_step)
    summary_writer.add_scalar('valid/c_loss', c_loss, current_step)
    summary_writer.add_scalar('valid/total_loss',total_loss, current_step)
    summary_writer.add_scalar('valid/macro_f1', maf1, current_step)
    summary_writer.add_scalar('valid/micro_f1', mif1, current_step)
    summary_writer.add_scalar('valid/exam_f1', ebf1, current_step)
    summary_writer.add_scalar('valid/acc', acc, current_step)
    summary_writer.add_scalar('valid/ha', ha, current_step)

    vae.train()

    return nll_loss, best_val_metrics
```

麻雀搜索-线性融合预测-MPVAE 分类优化代码:

```
import numpy as np
import torch
import sys
import datetime
from copy import deepcopy
import evals
from utils import build_path, get_label, get_feat
from model import VAE, compute_loss
import pandas as pd
import argparse
import matplotlib.pyplot as plt
import xgboost as xgb
import lightgbm as lgb
import joblib

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
parser = argparse.ArgumentParser()
parser.add_argument('-cp', "--checkpoint_path",
    default='./model/model_ours/lr-0.00075_lr-decay_0.20_lr-times_3.0_nll-0.10_l2-0.03_c-200.00/vae-361',
```

```
    type=str, help='The path to a checkpoint from which to fine-tune.')

parser.add_argument('-bs', "--batch_size", default=100, type=int, help='the number of data
    points in one minibatch')
parser.add_argument('-tbs', "--test_batch_size", default=128, type=int, help='the number of
    data points in one testing or validation batch')
parser.add_argument('-lr', "--learning_rate", default=0.00075, type=float, help='initial
    learning rate')
parser.add_argument('-epoch', "--max_epoch", default=3, type=int, help='max epoch to train')
parser.add_argument('-wd', "--weight_decay", default=1e-5, type=float, help='weight decay
    rate')
parser.add_argument('-lrdr', "--lr_decay_ratio", default=0.5, type=float, help='The decay
    ratio of learning rate')
parser.add_argument('-lrdt', "--lr_decay_times", default=3.0, type=float, help='The number of
    times learning rate decays')
parser.add_argument('-ntest', "--n_test_sample", default=10000, type=int, help='The sampling
    times for the testing')
parser.add_argument('-ntrain', "--n_train_sample", default=100, type=int, help='The sampling
    times for the training')
parser.add_argument('-z', "--z_dim", default=1000, type=int, help='z dimention: the number of
    the independent normal random variables in DMS the rank of the residual covariance matrix')

parser.add_argument('-label_dim', "--label_dim", default=5, type=int, help='the number of
    labels in current training')
parser.add_argument('-latent_dim', "--latent_dim", default=3, type=int, help='the number of
    labels in current training')
parser.add_argument('-meta_offset', "--meta_offset", default=0, type=int, help='the offset
    caused by meta data')
parser.add_argument('-feat_dim', "--feature_dim", default=20, type=int, help='the
    dimensionality of the features')

parser.add_argument('-se', "--save_epoch", default=1, type=int, help='epochs to save the
    checkpoint of the model')
parser.add_argument('-max_keep', "--max_keep", default=3, type=int, help='maximum number of
    saved model')
parser.add_argument('-check_freq', "--check_freq", default=120, type=int, help='checking
    frequency')

parser.add_argument('-nll_coeff', "--nll_coeff", default=0.1, type=float, help='nll_loss
    coefficient')
parser.add_argument('-l2_coeff', "--l2_coeff", default=1.0, type=float, help='l2_loss
    coefficient')
parser.add_argument('-c_coeff', "--c_coeff", default=200., type=float, help='c_loss
    coefficient')
parser.add_argument('-scale_coeff', "--scale_coeff", default=1.0, type=float, help='mu/logvar
    scale coefficient')
parser.add_argument('-keep_prob', "--keep_prob", default=0.5, type=float, help='drop out rate')
```

```
parser.add_argument('-resume', "--resume", action='store_true', help='whether to resume a
    ckpt')
parser.add_argument('-write_to_test_sh', "--write_to_test_sh", action='store_true',
    help='whether to modify test.sh')
parser.add_argument('-mode', "--mode",default="test", type=str, help='training/test mode')
parser.add_argument('-r_sigma', "--residue_sigma", default='', type=str, help='what sigma r to
    use')
args = parser.parse_args()
vae = VAE(args).to(device)
vae.load_state_dict(torch.load(args.checkpoint_path))
vae.eval()

def label_prediction(input_label,input_feat):
with torch.no_grad():

input_feat, input_label = torch.Tensor(input_feat).to(device), torch.Tensor(input_label)

input_label = deepcopy(input_label).float().to(device)
label_out, label_mu, label_logvar, feat_out, feat_mu, feat_logvar = vae(input_label,
    input_feat)
total_loss, nll_loss, nll_loss_x, c_loss, c_loss_x, kl_loss, indiv_prob =
    compute_loss(input_label, label_out,
label_mu, label_logvar,
feat_out, feat_mu,
feat_logvar,
vae.r_sqrt_sigma, args)
label_predict = indiv_prob.cpu().data.numpy()
for i in label_predict:
for j in range(0,5):
if i[j]<0.5:
i[j]=0
else:
i[j]=1
return label_predict

def compute_label_sum(label_predict):
y=-label_predict[0][0]+label_predict[0][1]+label_predict[0][2]-label_predict[0][3]+label_predict[0][4]
return y
def Penalty_1(sigma,label_predict):
y=sigma*(max(0,compute_label_sum(label_predict)))**2
return y

def linear_fusion(test_data):
"""————————————————————————————————————————————————————————随机森林————————————————————————————————————
# 下载随机森林模型模型
data_small = pd.read_excel(r"test_use.xlsx")
list = [data_small.columns]
```

```
print("asd",test_data[0][:20])
test_data=np.array(test_data[0][:20])
test_data = pd.DataFrame(test_data.reshape(1, 20), columns=list)
test_model_rfr = joblib.load(filename="RFR_model_test")

# 用下载下来的模型进行预测
rf_pred = test_model_rfr.predict(test_data)

"""————————————————————————————————————————————————————————————xgboost————————————————————————————

# 下载本地模型
test_model_xg = joblib.load(filename="xg_model_test")

"""用下载下来的模型进行预测，在此之前先将其转成dtest"""
dtest = xgb.DMatrix(test_data)
xg_pred = test_model_xg.predict(dtest)

"""————————————————————————————————————————————————————————————lightGBM————————————————————————————
# 下载本地模型
test_model_gbm = joblib.load(filename="test_gbm_model")


# 模型预测
gbm_pred = test_model_gbm.predict(test_data,num_iteration=test_model_gbm.best_iteration)

"""————————————————————————————————————————————————————————————线性融合————————————————————————————
linear_fusion_result = (rf_pred + xg_pred + gbm_pred)/3

return linear_fusion_result
def object_f(sigma,input_label,input_feat):
input_label=input_label.tolist()

input_feat = input_feat.tolist()
input_feat=[input_feat]
label_predict=label_prediction(input_label,input_feat)
y=linear_fusion(input_feat)#-Penalty_1(sigma,label_predict)
y=-y
return y

def SSA(pop, M, lb, ub, dim):
# global fit
P_percent = 0.2
pNum = round(pop * P_percent)


#print('lllllllllllll:', lb)
```

```
X = np.zeros((pop, dim))
X_rl= np.random.rand(1, 5)
fit = np.zeros((pop, 1))
sigma=1
for i in range(pop):
X[i, :] = lb + (ub - lb) * np.random.rand(1, dim)

fit[i, 0] = object_f(sigma,X_rl,X[i,:])



pFit = fit
pX = X
fMin = np.min(fit[:, 0])
bestI = np.argmin(fit[:, 0])
bestX = X[bestI, :]
Convergence_curve = np.zeros((1, M))
for t in range(M):
sigma=sigma+1
sortIndex = np.argsort(pFit.T)
fmax = np.max(pFit[:, 0])
B = np.argmax(pFit[:, 0])
worse = X[B, :]

r2 = np.random.rand(1)
if r2 < 0.8:
for i in range(pNum):
r1 = np.random.rand(1)
X[sortIndex[0, i], :] = pX[sortIndex[0, i], :] * np.exp(-(i) / (r1 * M))
X[sortIndex[0, i], :] = Bounds(X[sortIndex[0, i], :], lb, ub)
fit[sortIndex[0, i], 0] = object_f(sigma,X_rl,X[sortIndex[0, i], :])
elif r2 >= 0.8:
for i in range(pNum):
X[sortIndex[0, i], :] = pX[sortIndex[0, i], :] + np.random.rand(1) * np.ones((1, dim))
X[sortIndex[0, i], :] = Bounds(X[sortIndex[0, i], :], lb, ub)
fit[sortIndex[0, i], 0] = object_f(sigma,X_rl,X[sortIndex[0, i], :])
bestII = np.argmin(fit[:, 0])
bestXX = X[bestII, :]

for ii in range(pop - pNum):
i = ii + pNum
A = np.floor(np.random.rand(1, dim) * 2) * 2 - 1
if i > pop / 2:
X[sortIndex[0, i], :] = np.random.rand(1) * np.exp(worse - pX[sortIndex[0, i], :] /
    np.square(i))
else:
X[sortIndex[0, i], :] = bestXX + np.dot(np.abs(pX[sortIndex[0, i], :] - bestXX),
1 / (A.T * np.dot(A, A.T))) * np.ones((1, dim))
```

```
X[sortIndex[0, i], :] = Bounds(X[sortIndex[0, i], :], lb, ub)
fit[sortIndex[0, i], 0] = object_f(sigma,X_rl,X[sortIndex[0, i], :])
arrc = np.arange(len(sortIndex[0, :]))
# c=np.random.shuffle(arrc)
c = np.random.permutation(arrc)
b = sortIndex[0, c[0:20]]
for j in range(len(b)):
if pFit[sortIndex[0, b[j]], 0] > fMin:
X[sortIndex[0, b[j]], :] = bestX + np.random.rand(1, dim) * np.abs(
pX[sortIndex[0, b[j]], :] - bestX)
else:
X[sortIndex[0, b[j]], :] = pX[sortIndex[0, b[j]], :] + (2 * np.random.rand(1) - 1) * np.abs(
pX[sortIndex[0, b[j]], :] - worse) / (pFit[sortIndex[0, b[j]]] - fmax + 10 ** (-50))
X[sortIndex[0, b[j]], :] = Bounds(X[sortIndex[0, b[j]], :], lb, ub)
fit[sortIndex[0, b[j]], 0] = object_f(sigma,X_rl,X[sortIndex[0, b[j]]])
for i in range(pop):

if fit[i, 0] < pFit[i, 0]:
pFit[i, 0] = fit[i, 0]
pX[i, :] = X[i, :]
if pFit[i, 0] < fMin:
fMin = pFit[i, 0]
bestX = pX[i, :]
Convergence_curve[0, t] = fMin
# print(fMin)
# print(bestX)
return fMin, bestX, Convergence_curve




def Bounds(s, Lb, Ub):
temp = s
for i in range(len(s)):
if temp[i] < Lb[0, i]:
temp[i] = Lb[0, i]
elif temp[i] > Ub[0, i]:
temp[i] = Ub[0, i]

return temp



print(linear_fusion([T]))
SearchAgents_no = 1
Max_iteration = 10
dim=20
```

```
lb=np.zeros((1,dim))
ub=np.ones((1,dim))
[fMin, bestX, SSA_curve] = SSA(SearchAgents_no, Max_iteration, lb, ub, dim)
print(['最优值为: ', fMin])
print(['最优变量为: ', bestX])
thr1 = np.arange(len(SSA_curve[0, :]))

plt.plot(thr1, -SSA_curve[0, :])

plt.xlabel('num')
plt.ylabel('object value')
plt.title('line')
plt.show()
```