


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	华东师范大学
参赛队号	No.21102690109
队员姓名	1. 罗昕欣
	2. 陈炳生
	3. 刁鹏

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:

本文通过分析 1975 个化合物的 729 个分子描述符以及其对 ERα 生物活性影响，利用统计方法以及机器学习算法进行了大批量的数据处理和分析，主要完成了以下工作:

问题一中，化合物对 ERα 的生物活性的影响可以参考 pIC_{50} 指标，先对各化合物的分子描述符的数据进行预处理，剔除了部分子在全部 1975 个化合物中处于相同水平的分子描述符。通过随机森林的特征选择方法对剩余的分子描述符以及 pIC_{50} 指标构建回归模型并选择出最具有显著影响的 20 个分子描述符变量['BCUTc-1l','SHsOH','SsOH','minHsOH','maxHsOH','maxsOH','minsOH', 'SssO', 'MLFER_A','MDEC-23','WTPT-5','maxssO','MLogP','MDEC-22', 'nC', 'nBase' , 'minssO', 'LipoaffinityIndex','minsssN','BCUTc-1h'] 。

问题二中，我们使用了 stacking 模型，通过分子描述式的 training 集和化合物的 pIC_{50} 值构建了一个基于弹性网络（ElasticNet）回归，梯度提升回归（Gradient Boosting Regression），随机森林回归，LASSO 回归的带有 5 折交叉验证的集成模型，然后与 XGboost 和 LightGBM 进行模型融合，利用各模型结果的加权和作为最终结果，对化合物对 ERα 生物活性的进行定量预测，在 training 集上 MSE 可以低至 0.1096，准确率较高。对 testing 集中 50 个化合物的对 ERα 生物活性的预测结果参考附件 ERα_activity.xlsx。

问题三中，各个化合物的 ADMET 性质由五个指标来展示，为对每个指标都进行较为精准的预测，构建专门针对该指标的模型。针对每个指标的指标值只分为 0-1，确认这是个二分类问题，所以考虑利用分类算法对其进行拟合。我们对每个 ADMET 性质都使用了八种模型进行拟合，分别是：逻辑回归、临近算法、支持向量机、决策树、随机森林、Adaboost、XGboost、LightGBM，然后根据每个模型在验证集的准确率和相应的 auc 曲线下方面积进行比较，得出最优的拟合模型。最终，Caco-2 指标用支持向量机模型，CYP3A4 指标用 LightGBM 模型，hERG 指标用 Adaboost 模型，HOB 指标用 LightGBM 模型，最后 MN 指标先通过 SMOTE 采样解决正负样本比例不一致问题，然后采用用随机森林模型，并对原数据表中 50 个 test 的样本集做出预测并将结果放在附件 ADMET.xlsx。

问题四中，我们考虑采取某种搜索方法：我们先依据题目要求找到满足 ADMET 性质有三个较好，且生物活性较高的化合物作为初始值。我们将分子描述式的值都设为前面寻找到的化合物的分子描述式的值，然后我们将其中部分固定，对剩下的进行搜索。在搜索的同时判断其 ADMET 性质和生物活性，将其中优于初始值的纳入取值范围。类似的，每次固定一些分子描述式值，对剩下的部分进行搜索，我们就可以得出所有的分子描述式的取值范围，最终就可以得到这 20 个分子描述式的取值范围。

关键字：随机森林　stacking 模型　特征提取　模型融合

目录

1. 问题重述

1.1 问题背景

乳腺癌是目前世界上最常见，致死率较高的癌症之一。乳腺癌的发展与雌激素受体密切相关，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。目前，抗激素治疗常用于 ERα 表达的乳腺癌患者，其通过调节雌激素受体活性来控制体内雌激素水平。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。比如，临床治疗乳腺癌的经典药物他莫昔芬和雷诺昔芬就是 ERα 拮抗剂。

目前，在药物研发中，为了节约时间和成本，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。具体做法是：针对与疾病相关的某个靶标（此处为 ERα），收集一系列作用于该靶标的化合物及其生物活性数据，然后以一系列分子结构描述符作为自变量，化合物的生物活性值作为因变量，构建化合物的定量结构-活性关系（Quantitative Structure-Activity Relationship, QSAR）模型，然后使用该模型预测具有更好生物活性的新化合物分子，或者指导已有活性化合物的结构优化。

一个化合物想要成为候选药物，除了需要具备良好的生物活性（此处指抗乳腺癌活性）外，还需要在人体内具备良好的药代动力学性质和安全性，合称为 ADMET（Absorption 吸收、Distribution 分布、Metabolism 代谢、Excretion 排泄、Toxicity 毒性）性质。其中，ADME 主要指化合物的药代动力学性质，描述了化合物在生物体内的浓度随时间变化的规律，T 主要指化合物可能在人体内产生的毒副作用。一个化合物的活性再好，如果其 ADMET 性质不佳，比如很难被人体吸收，或者体内代谢速度太快，或者具有某种毒性，那么其仍然难以成为药物，因而还需要进行 ADMET 性质优化。为了方便建模，本试题仅考虑化合物的 5 种 ADMET 性质，分别是：1）小肠上皮细胞渗透性（Caco-2），可度量化合物被人体吸收的能力；2）细胞色素 P450 酶（Cytochrome P450, CYP）3A4 亚型（CYP3A4），这是人体内的主要代谢酶，可度量化合物的代谢稳定性；3）化合物心脏安全性评价（human Ether-a-go-go Related Gene, hERG），可度量化合物的心脏毒性；4）人体口服生物利用度（Human Oral Bioavailability, 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（Micronucleus，MN），是检测化合物是否具有遗传毒性的一种方法。

1.2 问题提出

本试题针对乳腺癌治疗靶标 ERα，首先于”ERα_activity.xlsx” 文件中提供了 1974 个化合物对 ERα 的生物活性数据。该数据集包含了 1974 个化合物的结构式的一维表达式 SMILS 式，该化合物对 ERα 的生物活性值 IC_{50}，以及由 IC_{50} 取负对数的值 pIC_{50}。其次在文件“Molecular_Descriptor.xlsx”中给出了 1974 个化合物的 729 个分子描述符的信息（即自变量）。最后在关注化合物的生物活性的同时还考虑了它的 ADMET 性质，该性质提供在文件“ADMET.xlsx”中。需依据题目所给的 ERα 拮抗剂信息（1974 个化合物样本，每个样本都有 729 个分子描述符变量，1 个生物活性数据，5 个 ADMET 性质数据），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，从而为同时优化 ERα 拮抗剂的生物活性和 ADMET 性质提供预测服务。

问题一：根据文件提供的数据，对 1974 个化合物的 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，并给出前 20 个对生物

活性最具有显著影响的分子描述符并详细说明过程和合理性。

问题二：结合问题 1，选择不超过 20 个分子描述变量构建对 ERα 生物活性的定量预测模型，然后利用所构建的模型对所给的 test 集中的 50 个化合物进行 IC_{50} 值以及对应的 pIC_{50} 的值进行预测。

问题三：利用 729 个分子描述符以及 1974 个化合物的 ADMET 数据分别构建化合物的 Caco-2、CYP3A4、hERG、HOB、MN 的分类预测模型，并对给出的 test 表中的 50 个化合物的 ADMET 性质进行相应的预测。

问题四：寻找哪些分子描述符以及这些分子描述符在什么取值或者处于什么取值范围时能够使化合物对抑制 ERα 具有更好的生物活性同时具有更好的 ADMET 性质（至少三个性质较好）

2. 模型假设

为了建模过程的顺利开展，构建模型时考虑以下的假设条件：

假设一：每个化合物的 ADMET 性质相互独立，互不影响，在建立预测模型时可以每个性质单独考虑。

假设二：对各个化合物的分子描述符不考虑其生化上的性质，只考虑其数值上的意义，仅从数据上进行分析。

假设三：化合物对 ERα 的生物活性的影响以及各分子描述符值的测量操作规范，来源可靠，且化合物性质稳定，在每次实验以及数据采集过程中的没有变性。

3. 符号说明

符号	说明
$Gain(A)$	变量 A 的信息增益
$E(A)$	A 集的信息熵
X_j	第 j 个特征变量
d_j	第 j 个化合物的分子描述式变量
N	样本集总样本数
β	回归方程系数
$L(\theta)$	损失函数

4. 问题的模型建立与求解分析

4.1 问题一

4.1.1 问题描述和分析

依据文件“Molecular_Descriptor.xlsx”和“ERα_activity.xlsx”提供的数据，现 1974 个化合物的 729 个分子描述符指标，多指标所带来的问题是冗余信息过多，数据量过大不仅会给分析更加复杂，还会造成对模型的分类或者预测的精度下降。因此我们将进行变量选择，寻找其中最具有代表性的 20 个分子描述式，这便是一个变量的特征选择问题。机器学习方法是通过挖掘数据内在特点去发现各变量的影响的大小，在大量高维数据的分析中表现优异，本题将通过不同的机械学

习的方法去选择出多组最具有特征的 20 个变量，每组变量都结合问题二所建立的对 ERα 生物活性的拟合模型，以对 ERα 生物活性预测的准确率为标准，选择准确率最高的那组 20 个变量。

4.1.2 实验数据的预处理

在进行数据处理之前，先所有数据表进行缺失值和异常值的检查和筛选。通过一轮筛查并未发现各表中存在空白或者无意义的数据。在“ERα_activity.xlsx”表中审查了 IC_{50} 数值的范围，其范围从 0.046 至 3500000 nM, 跨度范围虽然较大，但仍在可解释的范围内：极高 IC_{50} 说明该化合物对抑制 ERα 的生物活性的作用微乎甚微，极低 IC_{50} 说明该化合物对抑制 ERα 的生物活性的作用很强，浓度并无上下限的限制，均可以从生物学上给出合理解释，故不作为异常值考虑。

在化合物的分子描述符指标集中，分别计算每个分子描述符的特征方差，其中共有 225 个分子描述符的方差为 0 ，这意味着该分子描述式在这 1974 个化合物中都保持同一个水平，没有差异，则其对抑制 ERα 的生物活性的作用应是一致的，故不纳入模型考虑的范围将其从 729 个分子描述符变量中剔除，仅以剩下的 504 个分子描述符作为变量纳入建立模型的考虑范围。

4.1.3 随机森林模型

在介绍随机森林之前我们先介绍随机森林内的基础构件——决策树 [1]。

决策树是一个树状预测模型，它是由节点和有向边组成的层次结构。每棵树中包含 3 种节点：根节点、内部节点、叶子节点。根节点是一棵树最开始的节点，也是训练数据的输入口，在一个棵树中是唯一的存在。树中每个内部节点都是一个分裂问题：它将到达该节点的样本按照某个特征变量的取值（也可以称作属性）进行分割，并且该节点的每一个后继分支对应于该特征变量的一个可能值或者区间。叶子节点是没有后继分支的节点，其代表着模型训练的结束，它的值是该树模型的输出值，比如分类模型则输出带有分类标签，回归模型则输出预测值。

训练一颗决策树的算法有很多，最常用的是自上而下的贪婪算法，每个内部节点将所有可能的属性进行遍历，选择分类效果最好的特征变量来分裂节点，可以分成两个或者更多的子节点，在每个节点上都重复该过程直到这棵决策树能够将全部的训练数据准确率较高的分类，或所有特征变量都被用到为止。具体步骤如下：

（1）：假设 T 为训练样本集。

（2）：从特征变量中选择一个最能区别 T 中样本的变量。

（3）：创建一个树节点，它的值为所选择的变量。创建此节点的子节点，每个子链代表所选变量的一个取值（或者区间），每个子链是不互相重叠的，使用子链的值进一步将样本细分为子类。

重复（2）（3）过程直到满足下列条件的其中一个：

(a): 所有的变量都在树里得到了划分

(b): 与这个节点关联的所有训练样本都具有相同的目标属性，比如属于同一个分类（熵为 0）。

在决策变量（属性）的选择上可利用信息增益（information gain）来选择属性，每次计算所有剩余候选属性的信息增益，然后根据信息增益最大的一个作为此次的分类属性。信息增益是用熵作为尺度，是衡量属性对训练数据分类能力的标准。对于某个变量 A ，它的信息增益计算表达式是：

$$Gain(A) = I(s_1, s_2, .., s_m) - E(A)$$

其中 $I(s_1, s_2, .., s_m) = -\sum_{i=1}^{m} p_i \log_2(p_i)$ ，其中 s_i 表示属于类 i 的样本数，p_i 为

任意样本属于类 i 的概率，属性 A 将样本集合划分为了 v 个子集 $\{s_1, s_2, ..., s_v\}$，s_{ij} 表示子集 s_j 中属于类 i 的样本数，E(A) 是根据 A 划分的子集的熵或者期望

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j}, ..., s_{mj}}{s} I(s_{1j}, ..., s_{mj})$$

。

随机森林（random forest）是用随机的方式构建一个许多二叉决策树结合的集成算法，通过训练多个弱模型来组合成一个强模型。单个决策树通常表现出很高的方差并容易导致过拟合的现象，在随机森林则是多个决策树的组合，每一个决策树都是都是根据训练集中随机抽取样本构建的，在构建树期间拆分每个节点时是进行随机的拆分，即每个决策树相互独立，互不干扰的。虽然森林中注入的随机性产生的决策树具有一定程度的解耦预测误差，但取这些预测的平均值，可以抵消一些错误。随机森林通过组合不同的树来减少方差，有时以略微增加偏差为代价。在实践中，方差减少通常是显着的，因此会产生一个整体更好的模型，它能提高预测精度和并防止过拟合，在小样本的回归以及分类上表现出色。

在模型训练阶段，随机森林使用 bootstrap 采样从输入的训练数据集中采集多个不同的子集来依次训练多个不同决策树，针对切分变量和切分变量的取值本可以采用穷举法，即遍历每个特征和每个特征的所有取值，最后从中找出最好的特征变量和取值的切分点；针对于选择变量和切分点的好坏，一般以切分后节点的各个子节点不纯度的加权和来衡量，其计算公式如下：

$$G(x_i, v_{ij}) = \frac{n_{left}}{N_s} H(X_{left}) + \frac{n_{right}}{N_s} H(X_{right}) \tag{1}$$

其中，x_i 为某一个切分变量，v_{ij} 为切分变量的一个切分值，n_{left}, n_{right}, N_s 分别为切分后左子节点的训练样本个数、右子节点的训练样本个数以及当前节点所有训练样本个数，X_{left}, X_{left} 分为左右子节点的训练样本集合，$H(X_m)$ 为衡量节点不纯度的函数，回归任务常采用均方误差 MSE = $\frac{1}{N_m} \sum_{i=1}^{N_m} (y - \bar{y}_m)^2$ 在预测阶段，随机森林将内部多个决策树的预测结果取平均得到最终的结果。决策树中某一节点的训练过程在数学等价于下面优化问题：

$$(x^*, v^*) = \arg\min_{x_i, v_{ij}} G(x_i, v_{ij}) \tag{2}$$

在得到了森林之后便可以用该森林进行预测，从每个决策二叉树的根结点处输入新的样本点，样本点与当前节点的切分变量以及切分值进行对比，若样本变量的值小于当前节点的切分值则访问当前节点的左节点，反之访问右节点，以此往复直到到达叶子结点处，返回该叶子结点的预测值。随机森林的预测结果则是内部所有决策树的预测结果求平均值得。

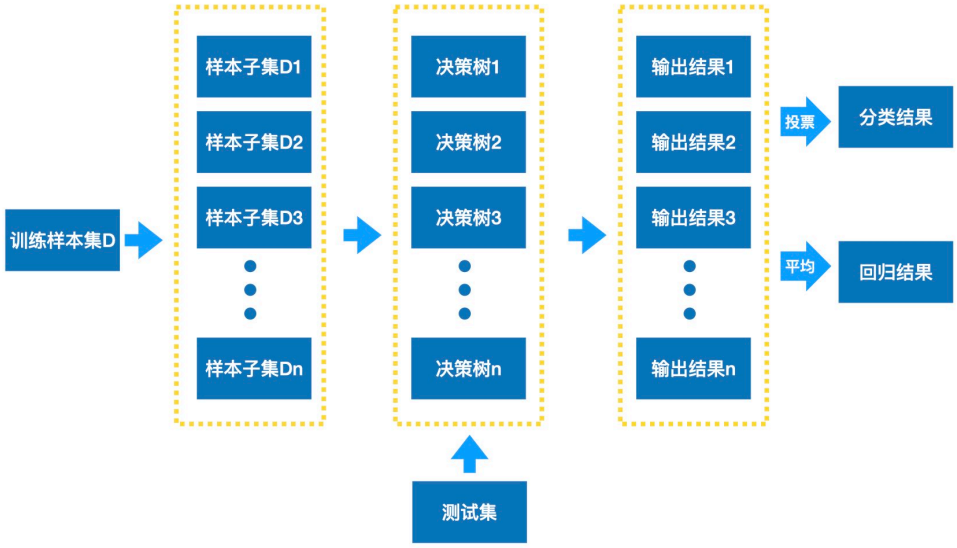

图 4-1　随机森林的简化分析图

在随机森林中某个特征变量 X 的重要性的计算方法如下：

1. 在随机森林中的每一颗决策树，使用 OOB 数据来计算它的袋外数据误差，记为 errOOB1。

2. 随机对袋外的数据 OOB 的所有样本的特征变量 X 进行一些变动，是它发生干扰，再次计算袋外数据误差 errOOB2。

3. 如果随机森林中有 Nt 颗树，那么特征 X 的重要性 = $\frac{errOOB2-errOOB1}{Nt}$，其可以解释成对该特征随机带来干扰之后，袋外数据的准确率大大降低了，说明这个特征对样本分类的结果影响很大，重要性就比较高。

特征选择的方法：

1. 对已构成随机森林中的特征变量按照变量的重要性进行排序

2. 按照删除比例从当前的特征变量中剔除重要性比较低的变量，得到一个新的特征集

3. 用新的特征集去建立新的随机森林，并重新计算每个特征变量的重要性并排序

重复以上步骤直到剩下变量的数目是我们提前所设定的数目，此时输出的变量即位我们所需的重要性最高的变量。

4.1.4 模型的求解以及结果分析

将 1975 个化合物的 504 个分子描述式向量计作 $d_i, i = 1, 2, ..., 1975$，每个 d_i 都是一个长度为 504 的向量，即 d_ij 表示第 i 个化合物的第 j 个分子描述式的值。记每个分子描述符变量为 $X_j, j = 1, 2, ..., 504$，每个 X_j 都为长度为 1975 的向量。以 504 个分子描述符的为自变量，化合物的 pIC_50 水平为因变量。建立 LASSO 回归模型（人工调整超参数 λ 的值使得最后输出结果中恰好有 20 个自变量的回归系数值显著不为 0，便可以得到前 20 个对生物活性最具有显著影响的分子描述符）和随机森林模型（设定二叉决策树的树木为 100，不控制决策树的深度）进行特征选择，具体求解过程利用 python 语言以及 scikit-learn 库中所带的函数，结果如下：

前 20 个对生物活性最具有显著影响的分子描述符（即变量）
['BCUTc-1l','SHsOH','SsOH','minHsOH','maxHsOH','maxsOH','minsOH', 'SssO','MLFER_A','MDEC-23','WTPT-5','maxssO','MLogP','MDEC-22', 'nC','nBase','minssO','LipoaffinityIndex','minsssN','BCUTc-1h']

4.2 问题二

4.2.1 问题描述和分析

利用问题一已求出来的 20 个变量建立一个化合物对 ERα 生物活性的定量预测模型，预测模型的构建可分为两步：模型训练以及预测，需先根据已有的数据建立起回归模型，再在未知因变量值的数据上进行预测。在实际建模过程中为了衡量模型的准确性，经常将已知的数据集再分为训练集和测试集（多数情况下训练集的样本个数应远大于测试集的样本个数），通过在测试集中预测的结果与真实结果的均方误差大小来作为评价一个模型准确性的标准。

4.2.2 弹性网络回归

LASSO 回归是在线性回归模型的基础上增加了一个 1-范数进行正则化，先介绍一下基础的线性回归模型：

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

在线性回归模型的众多自变量 X_i 中有的自变量是“关键因素”，对应的系数 $\beta_i \neq 0$，而有的自变量 X_j 与 Y 的关系不大，则其对应的系数 β_j 应是接近于 0 或者等于 0，LASSO 回归的作用就是稀疏估计筛选出回归系数接近于 0 的自变量。LASSO 回归系数的表达式为：

$$\begin{aligned} \hat{\beta}^{lasso} &= \arg\min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \\ &\text{subject to} \sum_{j=1}^{p} |\beta_j| \leq t \end{aligned} \tag{3}$$

等价于

$$\hat{\beta}^{lasso} = \arg\min_{\beta} \{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \} \triangleq \arg\min_{\beta} Q(\beta) \tag{4}$$

其中 N 为样本个数，λ 为常数系数，是一个需要进行调优的超参数，$\sum_{j=1}^{p} |\beta_j|$ 即为回归系数向量 β 的 L_1 范数，$Q(\beta)$ 为 LASSO 回归的损失函数。

简单来说，就是在线性回归的最小二乘估计参数的过程中添加了一个限制条件，当 t 越小，λ 则就越大，对估计参数的压缩作用也就越强，当对目标函数求最小时，一些不太重要的自变量系数就会被压缩至 0，从而达到了筛选变量的作用。以一元 LASSO 回归求解为例解释 LASSO 回归的解的意义：红色区域是目标函数值的“等高线”，每一圈都表示在此处的回归系数参数计算的目标函数值是相等的，红色的圆心是实际最优参数。但是由于我们对解空间做了限制，蓝色区域是由于 L_1 范数的约束，它是一个菱形或者多边形，则很容易切到坐标轴上，因此很容易产生稀疏的结果，这样就解释了为什么 lasso 可以进行特征选择。红色区域与蓝色区域的交点即为 LASSO 回归的在约束下的最优参数。

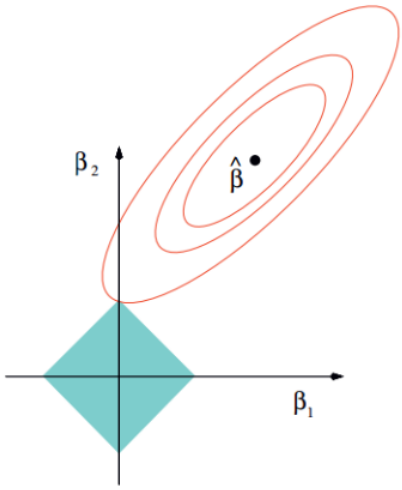

图 4-2　LASSO 回归的回归系数解空间与约束示意图

若将 L_1 范数更换为 L_2 范数，则又是一个经典的回归方法——岭回归（ridge regression），参考 LASSO 回归，可以直接给出其对应的目标函数为：

$$\hat{\beta}^{ridge} = \arg\min_{\beta}\{\frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}(\beta_j)^2\} \tag{5}$$

其中 N 为样本个数，λ 为常数系数，是一个需要进行调优的超参数，$\sum_{j=1}^{p}(\beta_j)^2$ 即为回归系数向量 β 的 L_2 范数。约束的范围由 LASSO 的多边形变成了一个以原点为圆心单位圆，如图：损失函数的等值点与单位元的切点很难会出现在坐标

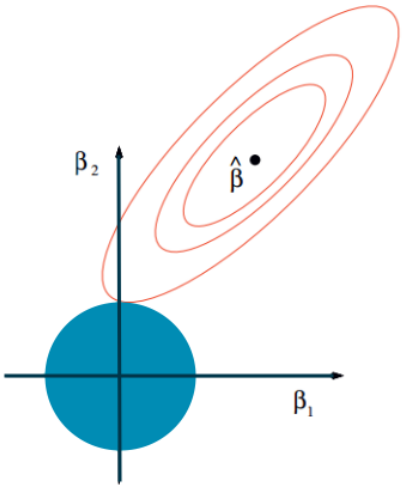

图 4-3　岭回归的回归系数解空间与约束示意图

轴上，因此不容易产生稀疏矩阵，适当的防止了过拟合的现象，增强了模型的泛化能力。

而 ElasticNet 回归，即弹性网络回归，则是结合了岭回归与 LASSO 回归，即结合了 L_1 和 L_2 正则化，其回归参数形式如下：

$$\hat{\beta}^{ElasticNet} = \arg\min_{\beta}\{\frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}x_{ij}\beta_j)^2 + \lambda_1\sum_{j=1}^{p}|\beta_j| + \lambda_2\sum_{j=1}^{p}(\beta_j)^2\} \tag{6}$$

ElasticnNet 回归继承了岭回归的稳定性，而且当有多个相关特征时，不同于 LASSO 回归只选择一组变量而将其他的压缩为 0，它可以产生群体效应保留多个变量。

求解常用的是坐标轴下降法（coordinate descent），即沿着坐标轴的方向去下降，这有些类似于大家熟知的梯度下降法，不同的是梯度下降的方向是沿着各分量的梯度（或者偏导数) 方向下降。由于 LASSO 回归的的损失函数由于 L_1 范数的存在，存在着不可导的点，固常用的梯度下降法以及牛顿迭代寻找最小值都不可用，因此我们选择了类似的坐标轴下降法，通过启发式的方法一步步迭代求得函数的最小值。

对于一个可微的凸函数 $f(\theta)$，其中 $\theta = (\theta_1, \theta_2, ..., \theta_n)$，若存在某一点 $\bar{\theta}$，使得 $J(\theta)$ 在 $\theta = \bar{\theta}$ 时，在各个坐标轴上都是最小值，那么 $J(\bar{theta})$ 则是一个全局的最小值。故 LASSO 回归的求解过程转化为一个优化目标，在 β 的 p 个坐标轴上对损失函数做迭代的下降，当所有的坐标轴上的 β_i 都达到收敛，我们的损失函数则达到最小，此时的 β 即为我们所要求的结果，即得到各个变量的回归系数。具体算法过程如下：

1. 对向量 β 随机取初始值，记为 $\beta^{(0)}$。

2. 在第 k 轮迭代中，从第 1 个系数 β_1^k 开始，依次求 β_i^k 。β_i^k 的表达式如下：

$$\beta_i^k \in \arg\min_{\beta_i}\{Q(\beta_1^{(k)}, \beta_2^{(k)}, ..., \beta_{i-1}^{(k)}, \beta_i, \beta_{i+1}^{(k-1)}, ...\beta_p^{(k-1)}\}. \tag{7}$$

由于 $Q(\beta)$ 此时只有 β_i 是变量，其余均为固定的值，因此可以通过容易通过二分法等简单方法寻找最小值。

3. 如果 $\beta^{(k)}$ 和 $\beta^{(k+1)}$ 在各维度上的变化都足够小，那么 $\beta^{(k)}$ 即为最终的结果，否则就进入下一轮的循环。

4.2.3 梯度提升回归

梯度提升回归，即 Gradient Boosting Regression（GBR），是由多个弱分类器的组合，在每次迭代的时引入一个弱分类器，最终结合成一个强分类器，而引入的弱分类器的标准，就是使得我们的模型在损失函数上持续下降，而梯度方向是是损失函数下降最快的方向，固我们从梯度方向出发去改进模型。接下来简要的介绍算法实现的过程：

1. 对于训练的自变量 X 以及因变量 Y，先选择一个弱分类器如决策树构建模型 F(·)，得到的损失函数为 $L(X) = 1/2(Y - F(X))^2$；

2. 确定损失函数的梯度方向：$\frac{-\partial L(X)}{\partial F(X)}$

3. 更新回归模型 $F^{(m)}(X) = F^{(m-1)}(X) - \rho\frac{-\partial L(X)}{\partial F^{(m-1)}(X)}$，其中 ρ 为提前设定学习率。

梯度提升回归经常与决策树联系在一起，以决策树这个弱分类器作为基不断地引入多个决策树，与随机森林的构建有些许相似。梯度提升回归通常能提供更好的精度，且对于数据预处理的需求较低，适合在本问题中引入。

4.2.4 Stacking 模型与模型融合

利用子模型的预期性能，加权各子模型对组合预测的贡献，可以改善模型平均。通过训练一个全新的模型来学习如何最好地组合来自每个子模型的贡献，这种方法被称为 Stacked Generalization(堆栈泛化)，或简称 Stacking，可以产生比任何单个贡献模型更好的预测性能。Stacking 模型本质上是一种分层的结构，此次以简单的二层 Stacking 进行分析。第一层由多个基模型组成，在每个基模型中输入为原始训练集 training1 进行训练，然后对 training1 和 testing 集进行预测，

输出预测结果 train_prediction1 和 test_prediction1；第二层的模型以第一层基模型的输出 train_prediction1 作为训练集进行再利用其他的模型进行训练，然后对 test_prediction1 进行预测，从而得到完整的 stacking 模型的本质。stacking 的方法在各大数据挖掘比赛上都很风靡，模型融合之后能够小幅度的提高模型的预测准确度。但是在实际情况下 train_prediction1 的得到是有问题的，用整个训练集来训练得到的模型，固需要引入一个常用的解决过拟合现象的方法——K 折交叉验证。我们以一个经典的 stacking 图 [2] 再详细解释 stacking 过程，上层是控制基模型的个数，下层需要控制 K 折交叉验证的个数 K：

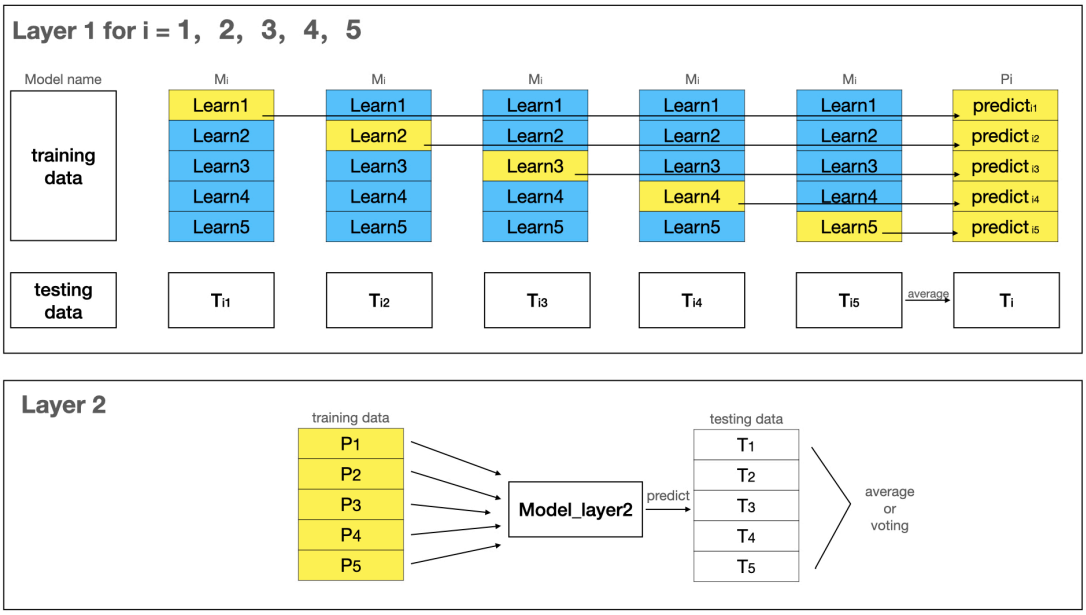

图 4-4　5 折交叉验证的双层 stacking 过程

假设我们有 5 个基模型 M1、M2、M3、M4、M5（可是不同的模型），训练集 training data，测试集 testing data，采用 5 折交叉验证：

第一层：由于采用 5 折交叉验证，我们将 training data 分为 $Learn_1$，$Learn_2$，$Learn_3$，$Learn_4$，$Learn_5$ 5 个子集（通常采用随机分配的方法）。对于每个基模型 $M_i, i = 1, 2., .., 5$，每次以 $Learn_j$ 作为训练集内的测试集，其余 Learn 子集的作为训练集进行模型训练并在 $Learn_j$ 集上进行测试，得到对 $Learn_j$ 预测结果 $Predict_{ij}$，对测试集 testing data 也进行预测得到结果记为 T_{ij}，求平均得到 T_i 即为第 i 个基模型的预测结果，结合所有的 Learn 集预测结果得到基模型 M_i 对 training data 的预测结果 $P_i, i = 1, 2, ..., 5$，

第二层：将第一层得到的 $P_1, P_2, ..., P_5$ 作为第二层的训练集 training data2，$T_1, T_2, ..., T_5$ 即为第二层的测试集，再利用第二层的模型（可于第一层的模型不同）进行训练和预测，预测得到的结果即为我们需要的对 testing data 的预测结果。

模型融合（Ensemble）是一种非常有效的技术，通过把多个单模型融合在一起，能够降低偏差和方差，控制模型的过拟合情况，提高模型的泛化能力的同时保证准确率。stacked model 是基于弱分类器的集成，将其与强分类器进行融合，在不损失模型精度的条件下，提升模型的泛化能力。

4.2.5 模型的求解以及结果分析

在本题中我们将采用问题一所得到的 20 个特征分子描述符变量与化合物的 pIC_{50} 值，以随机森林回归，梯度提升回归以及弹性网络回归作为 Stacking 过程的三个基模型，以 LASSO 回归为第二层的训练模型，构建一个 5 折交叉验证的二层的 Stacking 模型，具体代码参考附录 B，得到对 training 的回归模型，以回归模型对 training 集的训练结果与真实指标求均方误差 MSE 来作为评价一个模型模型精度的大小，得到 stacking 模型在 training 集上的 MSE 仅为 0.1144，精度较高。

之后我们对模型进行了改进，我们仍沿用之前建立的 stacking 模型，引入 xgboost 和 lightGBM 分类器，在 training 集中进行训练，人工调参数直到达到一个较小的 MSE 值。我们将 stacking 模型，xgboost 和 lightGBM 模型得到的结果的均方误差分别记作 $MSE1, MSE2, MSE3$，选择加权和的融合方法，为 MSE 比较大的模型分配较低的权重，对所有模型的结果求一个加权和，得到一个新的结果。4 个模型的 MSE 值如下表:

模型	MSE	权重
stacking	0.11439596778220512	0.7
xgboost	0.08676743051196513	0.25
lightGBM	0.35050901645970484	0.05
模型融合	0.10955402061292448	

以 stacking + ensemble 的模型去预测 testing 集中的 50 个化合物的 pIC_{50} 值并通过 pIC_{50} 和 IC_{50} 之间的转化公式

$$IC_{50} = 10^{-pIC_{50}+9}$$

来得到对应的 IC_{50} 的值，具体的预测结果详见附件‘ERα_activity.xlsx’, 此处仅作一个简单展示:

SMILES	Caco-2	CYP3A4	hERG	HOB	MN
COc1cc(OC)cc(\C=C\c2ccc(OS(=O)(=O)[C@@H]3C[C@@H]4O[C@H]3C(=C4c5ccc(O)cc5)c6ccc(O)cc6)cc2)c1	0	1	1	0	1
OC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3ccccc23)c4ccc(O)cc4	0	1	0	0	1
COc1ccc2C(=C(CCOc2c1)c3ccc(O)cc3)c4ccc(\C=C\C(=O)O)cc4	0	1	1	0	1
OC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
OC(=O)\C=C\c1ccc(cc1)C2=C(CCSc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
CC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(\C=C\c4ccccc4)cc3)c5ccc(F)cc5OCC2	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)c4ccccc4)cc3)c5ccc(F)cc5OCC2	0	1	1	0	1
OC(=O)\C=C\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
CCN(CC)C(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)N4CCCC4)cc3)c5ccc(F)cc5OCC2	0	1	1	0	1
CCN(CC)CCNC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)N4CCNCC4)cc3)c5ccc(F)cc5OCC2	0	1	1	0	1
CN1CCN(CC1)C(=O)\C=C\c2ccc(cc2)C3=C(CCOc4cc(F)ccc34)c5ccc(O)cc5	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)N4CCN(Cc5ccccc5)CC4)cc3)c6ccc(F)cc6OCC2	0	1	1	0	1
Cc1ccc(cc1)N2CCN(CC2)C(=O)\C=C\c3ccc(cc3)C4=C(CCOc5cc(F)ccc45)c6ccc(O)cc6	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)Nc4ccccc4)cc3)c5ccc(F)cc5OCC2	0	1	1	0	1
OC(=O)COc1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
Oc1ccc(cc1)C2=C(c3ccc(C=O)cc3)c4ccc(F)cc4OCC2	0	1	1	0	1
CCC(=C(c1ccc(O)cc1)c2ccc(\C=C\C(=O)O)cc2)c3ccccc3	0	0	1	0	1
OC(=O)CCCOc1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	0	1	1	0	1
COc1ccc2C(=C(CCOc2c1)c3ccc(O)cc3)c4ccc(OCC(=O)O)cc4	0	1	1	0	1
CCCC(CCC)(c1ccc(O)c(C)c1)c2cccs2	0	0	1	1	0
CCCC(CCC)(c1ccc(O)c(C)c1)c2cc(C)cs2	0	1	1	1	0
CCCC(CCC)(c1ccc(O)c(C)c1)c2ccc([nH]2)C(=O)OCC	0	0	1	0	0
CCCC(CCC)(c1ccc(O)c(C)c1)c2ccc(C(=O)OCC)n2C	0	1	1	1	0

图 4-5　附件 ERα_activity.xlsx 中部分数据

4.3 问题三

4.3.1 问题描述和分析

针对该问，我们选择将五个分类预测模型分别建立，针对不同的指标，选择最优的分类预测模型，下面是每个指标的选择过程。首先，对数据进行处理，由于部分方法对于数据是否标准化十分敏感，因此我们对所有数据都进行了标准化，但是，标准化有时反而不会改进模型的拟合效果，因此我们对未标准化、标准化的数据都进行了模型拟合，不过囿于篇幅所限，我们会直接给出较优结果。所以下文中我们在给出模型效果时，如果未加说明，则说明使用的是未标准化的数据；标准化的数据我们在介绍模型效果时即会特殊说明。

针对每个指标，我们都采用 8 种模型对数据进行拟合，之后通过比较模型的优劣来对每个指标选择适合的模型。对于每个拟合模型，我们将 ADMET.xlsx 中的 1480 个化合物当作训练集，494 个化合物当作验证集，以此进行模型拟合以及效果的评价。同时，我们认为如果模型在训练后，在验证集上的准确率越高，说明该模型拟合效果越好；此外，我们还考虑了 auc 曲线，该曲线下方面积衡量了模型对真阳率与伪阳率的辨别效果，auc 值越大说明辨别效果越好，所以下文中我们同时比较两个值来说明模型的优劣。

最终，想要预测 5 个 ADMET 性质值，对每个都建立 8 个模型拟合后，最终采用了 1 个 SVM 所建立模型，1 个 Adaboost 算法所建立模型，1 个随机森林所建立模型，2 个 LightGBM 算法所建立模型。而由于随机森林在上文中已经介绍过，这里就不再赘述，主要说明剩下的 7 个算法。

4.3.2 支持向量机方法

支持向量机是一种二分类模型，它将实例的特征向量映射为空间中的一些点，SVM 的目的就是想要画出一条线，以“最好地”区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。SVM 适合中小型数据样本、非线性、高维的分类问题。

将实例的特征向量（以二维为例）映射为空间中的一些点，如下图的实心点和空心点，它们属于不同的两类。SVM 的目的就是想要画出一条线，以“最好地”区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。

而在实际操作的时候，我们会发现给定超平面后，每一个类别，我们都能找到一个支持向量，支持向量上的点到超平面的距离都是一样的，所以我们只需考虑支持向量即可，这也是该方法名称的由来。

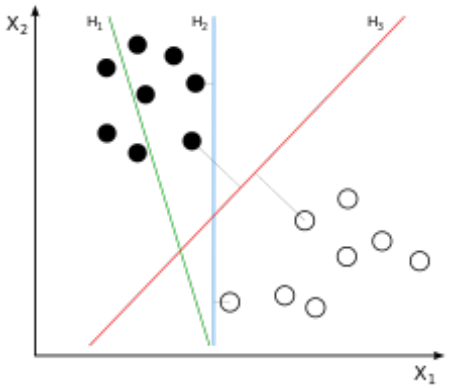

图 4-6　SVM 划分类别示意图

支持向量机方法有一些特点：

1、虽然可以找出无数条线将样本点进行区分，具体区别就是效果好不好，类似上图，蓝色和绿色线的分类效果就不是很好，而红线就是一个分类效果比较好的超平面。

2、支持向量机中，每做一个超平面，其两侧数据点都距离它有一个最小距离（垂直距离），这两个最小距离的和就是间隔。而我们的目的就是让间隔最大，这样我们分类的犯错概率才会更低。

3、训练好的 SVM 模型的算法复杂度是由支持向量的个数决定的，而不是由数据的维度决定的，所以 SVM 算法不太容易产生过拟合现象。

4、SVM 训练出来的模型完全依赖于支持向量，所以即使训练集中所有非支持向量的点都被去除，重复训练过程，我们同样会得到完全一样的模型。

在我们实现的时候，我们直接调用了 sklearn 库来拟合 SVM 模型，但是其中还涉及核函数的选择问题，下面简单说明一下核函数。

至于为什么要使用核函数，因为在线性 SVM 中，转化为最优化问题时求解的公式计算都是以内积形式出现的，而内积的算法复杂度非常大，所以需要使用核函数来取代计算非线性映射函数的内积。

我们所考虑的，除了线性内核之外，就是高斯径向核函数

$$K(X_i, X_j) = e^{-||X_i - X_j||^2} \tag{8}$$

以及 S 型核函数

$$K(X_i, X_j) = tanh(kX_i \cdot X_j - \delta) \tag{9}$$

4.3.3 XGboost 方法

XGboost 方法是一个基于决策树的分类算法，所以首先就要明确怎样的分类才算是合理的分类，什么才是关于分类的量化指标。所以在介绍该方法之前，首先我想说明关于量化分类效果的度量标准：信息增益。信息增益的核心思想就是以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂，那什么是信息增益呢？为了准确定义信息增益，我们先定义信息论中广泛使用的一个度量标准，就是熵，它刻画了任意样本集的纯度。通过熵，我们就能评估一个决策树的分类效果好坏了。

而在单个决策树基础上，我们就可以定义集成学习了。通过构建多个决策树对数据集进行预测，然后用某种策略将多个决策树预测的结果集成起来，作为最终预测结果，就是集成学习。而根据集成学习的各个决策树之间有无依赖关系，分为 Boosting 和 Bagging 两大类，Boosting 就是各分类器之间有依赖关系。XGboost 就属于 Boosting 流派，而想要详细说明，还需要介绍一下 GBDT，因为 XGboost 本质上还是 GBDT，只是把速度效率发挥到了极致。

GBDT 的原理很简单，就是把所有决策树的结果相加等于预测值，然后下一个决策树去拟合误差函数对预测值的残差，这样吧所有决策树的结论结合，就能得到最终结果。其中，GBDT 用负梯度来近似残差，然后把均方差损失函数作为损失函数。而到了 XGboost 算法，其核心思想在于不断添加决策树，不断的进行特征分裂来生长一棵树，每次添加一个树后，再拟合上次预测的残差。然后当我们训练完成后，只要给一个样本点，我们就可以根据这个样本的特征，在每棵树中找到一个对应的叶子节点，然后根据叶子节点的分数，得到该决策树对样本点的预测分数，最后将所有决策树的对应分数加起来就是该样本预测值。

显然，我们的目标是要让所有树的预测值尽量接近真实值，同时尽可能拥有较大的泛化能力，所以从数学角度看这是一个泛函最优化问题，那么就可以把目标函数简化如下：

$$L(\Phi) = \sum_i l(\hat{y}_i - y_i) + \sum_k \Omega(f_k) \tag{10}$$

这个目标函数就分为两部分：损失函数和正则化项，其中损失函数用来刻画预测分数与真实分数的差距，正则化定义复杂度。类似之前提到的 GBDT 的套路，XGboost 也是需要将多棵树的得分累加得到最终的预测得分，那么每轮加入的那棵树就十分关键。这颗树就是要让我们的目标函数尽量降低，因此需要考虑损失函数，而 XGboost 的优势就是它利用泰勒展开，将展开的各项对应到目标函数中，这样，我们就把损失函数部分解决了。

关于正则化项，我们得说明我们是利用叶子节点集合以及叶子节点得分来表示的，每个样本都得落在一个叶子节点上。因此，XGboost 对树的复杂度要求就包含了两部分：一个树里面的叶子节点个数和一个树上叶子节点得分的模平方，这样就把正则化部分解决了。

4.3.4 LightGBM 算法

LightGBM 是个快速的，分布式的，高性能的基于决策树算法的梯度提升框架。可用于排序，分类，回归以及很多其他的机器学习任务中。虽然 XGboost 算法十分优异，但是在实际应用时速度太慢，而 LightGBM 在不降低准确率的前提下，速度提升了 10 倍左右，占用内存下降了 3 倍左右。因为它是基于决策树算法的，它采用最优的叶明智策略分裂叶子节点，然而其它的提升算法分裂树一般采用的是深度方向或者水平明智而不是叶明智的。因此，在 LightGBM 算法中，当增长到相同的叶子节点，叶明智算法比水平明智算法减少更多的损失。因此导致更高的精度，而其他的任何已存在的提升算法都不能够达。实际应用时主要体现在两点：一是不使用所有的样本点来计算梯度，而是对样本进行采样来计算梯度；二是不是使用所有的特征来进行扫描获得最佳的切分点，而是将某些特征进行捆绑在一起来降低特征的维度，使寻找最佳切分点的消耗减少。

4.3.5 其他分类算法

除上述方法外本文还利用了一些其他的分类方法与上述三种方法做对比：

（1）Logistic 回归

该方法主要用于二分类问题，其核心思想在于寻找决策函数，只要将变量值带入决策函数后，超过了决策边界，我们就认为其属于一个类别；反之，就认为它属于另一个类别。而逻辑回归与回归的区别就在于需要寻找阶跃函数，使得决策函数的连续输出值变为不连续的类别分类值，这样给定分子描述式，我们就可以给出该化合物相应的 ADMET 性质。

（2）临近算法

KNN 是一个非常直观的分类算法，核心思想就是计算新样本点到过去样本的距离，然后找出 K 个距离最近的，我们认为新样本点与这 K 个点最相似，所以只要找出这 K 个点中哪个类别占比最大，我们就认为新样本点即为该类别。

（3）决策树

决策树也是分类方法中较为常见的一种，其核心思想就是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，并以此为根据进行分类。这种方法利用信息学理论中熵的概念，将对象属性与对象值之间构建一种映射关系。具体在本题中，可以将训练集中各个化合物的分子描述式看作对象属性，将已知的 ADMET 性质值看作对象值，那么就可以学习到一个分类器，通过该分类器就可以对新样本点进行预测。

（4）Adaboost 算法

Adaboost 算法是 boosting 系列算法中最著名的算法之一，既可以用作分类，也可以用作回归。Boosting 算法的工作机制是首先从训练集用初始权重训练出一个弱学习器 1，根据弱学习的学习误差率表现来更新训练样本的权重，使得之前

弱学习器 1 学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器 2 中得到更多的重视。然后基于调整权重后的训练集来训练弱学习器 2.，如此重复进行，直到弱学习器数达到事先指定的数目 T，最终将这 T 个弱学习器通过集合策略进行整合，得到最终的强学习器。其中 Adaboost 算法的主要特点在于其弱学习器权重系数的计算、更新样本权重和集合策略，尤其是 Adaboost 分类采用加权平均法的集合策略，使得其分类效果较好。

4.3.6 模型的建立与求解

下面是对 ADMET 性质的五个指标进行分类预测，我们将每个指标单独考虑，分别用多个分类算法建立多个分类模型，通过对比选择准确率最高的模型用于在之后在 testing 集上的预测。ROC 曲线全称为受试者工作特征曲线（receiver operating characteristic curve），它是根据一系列不同的二分类方式（分界值或决定阈），以真阳性率（敏感性）为纵坐标，假阳性率（1-特异性）为横坐标绘制的曲线。AUC 值，即为 ROC 曲线下与坐标轴围成的面积，是衡量一个学习器优劣的一种性能指标，AUC 越接近 1.0，检测方法真实性越高; 等于 0.5 时，则真实性最低，无应用价值。使用在 training 集上的准确率和 AUC 值作为模型优劣的指标 [3]。

1.**Caco-2 指标**

首先观察数据，所有化合物中，有 1215 个化合物在该指标取值为 0，759 个化合物在该指标取值为 1。同时，根据我们对该指标的生理意义理解，Caco-2 度量了化合物被人体吸收的能力，特别的，该指标值为 1 时代表该化合物的小肠上皮细胞渗透性较好，那此时该化合物更容易被人体吸收，所以我们认为指标值为 1 时属于具有更好的 ADMET 性质。

（1）逻辑回归方法: 该方法在验证集上的准确率为 87.44939271255061%，然后 auc 曲线下方的面积为 0.947381457891012，效果较为理想。

（2）标准化支持向量机方法: 在使用该方法时，一个可以调整的即为所使用的内核，所以我们针对三种不同内核：linear、rbf、sigmoid 分别计算了在 training 集上的准确率，auc 取值。

内核	验证集准确率	AUC 取值
linear	0.8825910931174089	0.9493984430290163
rbf	0.9149797570850202	0.9752653927813164
sigmoid	0.8299595141700404	0.896302193913659

显然，根据表格中数据，当使用 rbf 内核时，支持向量机模型在验证集上准确率和 auc 取值都较高，所以如果后续选择该模型，那么就应该使用 rbf 内核。

同时，在使用 SVM 时，参数 gamma 也是影响模型拟合效果的一个重要指标，所以我们考虑如何是 gamma 取到较优值，所以我们将 gamma 从 10 的-10 次方到 1，按照等比数列分 50 份，进行遍历，并通过 svc.score 函数来判断效果，最终 gamma=0.007196856730011514 时效果最好。

(3）标准化 KNN 方法: 该方法在验证集上准确率为 0.8724696356275303, auc 值为 0.9478060863411182，效果不如 rbf 内核的支持向量机方法。但是，在使用该方法时，仍然有参数可以调整的范围，KNN 方法中，K 就是一个可以调整的指标。同样，我们选择遍历的方法选择 K 最优的取值，但是由于 K 取值只能为整数，所以我们考虑范围就设置为为 2 到 12 之间的所有整数了。最终 K=2 时效果最好。

(4）决策树方法: 该模型在验证集上的准确率为 0.8785425101214575，相应的 auc 取值为 0.8677105449398443。

(5）随机森林方法: 该模型在验证集上的准确率为 0.8785425101214575，相应的 auc 取值为 0.970859872611465。

(6）Adaboost 算法: 该模型在验证集上的准确率为 0.9008097165991903，相应的 auc 取值为 0.8970806794055203。

(7）XGboost 算法: 该模型相应的 auc 取值为 0.96130。

(8）lightGBM 算法: 该模型相应的 auc 取值为 0.96518。

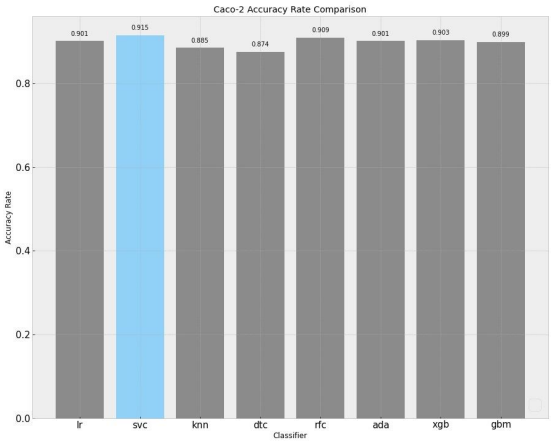

图 4-7　Caco-2 指标在多个分类模型下的准确率

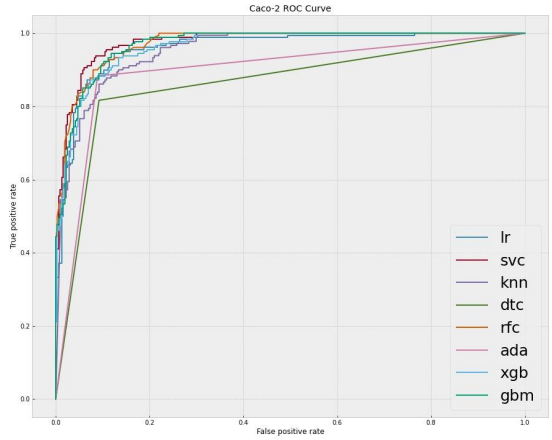

图 4-8　Caco-2 指标在多个分类模型下的 ROC 曲线

最终，针对该指标，我们选取的模型是支持向量机方法所建立模型，对应最高的验证集准确率为 0.9149797570850202。

对于之后的指标，模型构建方法类似，所以我们直接展示结果。

2.CYP3A4 指标

CYP3A4：‘1’代表该化合物能够被 CYP3A4 代谢，‘0’代表该化合物不能被 CYP3A4 代谢，而指标值为 1 就说明该化合物进入人体后会被代谢，而被代谢一般来说即说明其结构发生变化，进而其药理特性一般也会发生变化，最终导致药

物失效。所以该指标值为 1，我们认为是该 ADMET 性质不好，指标值为 0 时，我们认为其有较好的 ADMET 性质。

最终确定模型为 LightGBM 算法所建立模型，验证集准确率为 0.95344129554。

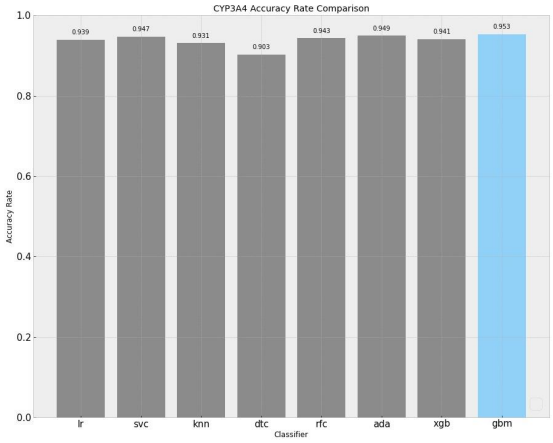

图 4-9　CYP3A4 指标在多个分类模型下的准确率

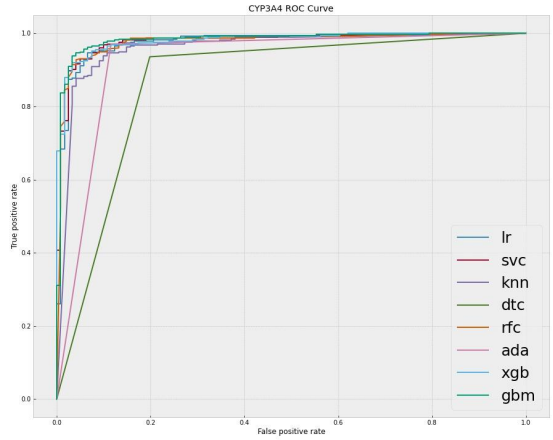

图 4-10　CYP3A4 指标在多个分类模型下的 ROC 曲线

3.hERG 指标

hERG：‘1’代表该化合物具有心脏毒性，‘0’代表该化合物不具有心脏毒性，显然，具有心脏毒性的化合物是应该减少使用的，所以该指标为 1 时，我们认为其 ADMET 性质不好；指标值为 0 时我们认为其拥有更好的 ADMET 性质。

最终确定模型为 Adaboost 方法所建立模型，验证集准确率为 0.9089068825910931。

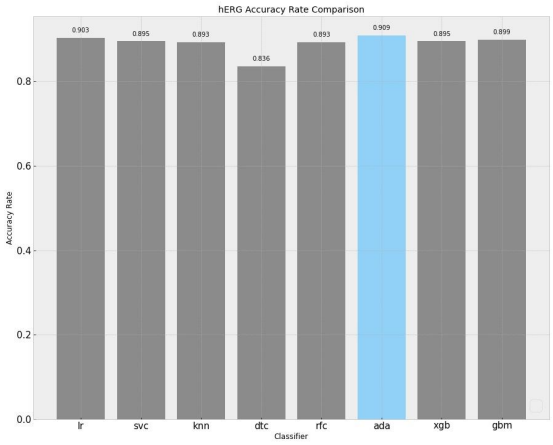

图 4-11　hERG 指标在多个分类模型下的准确率

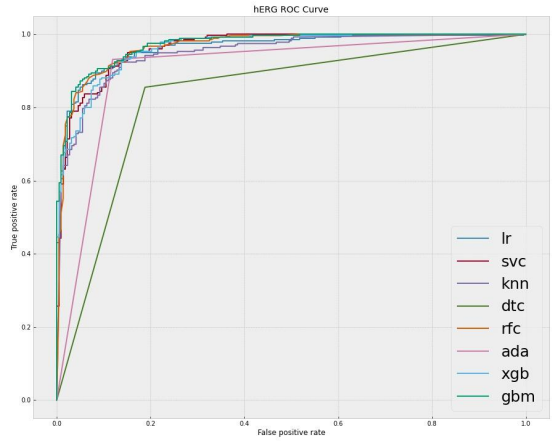

图 4-12　hERG 指标在多个分类模型下的 ROC 曲线

4.HOB 指标

HOB：‘1’代表该化合物的口服生物利用度较好，‘0’代表该化合物的口服生物利用度较差，显然，口服生物利用度较好的化合物更容易被病人使用，所以，该指标值为 1 时，我们认为其 ADMET 性质更好。

最终确定模型为 LightGBM 所建立模型，对应的验证集准确率为 0.896761133。

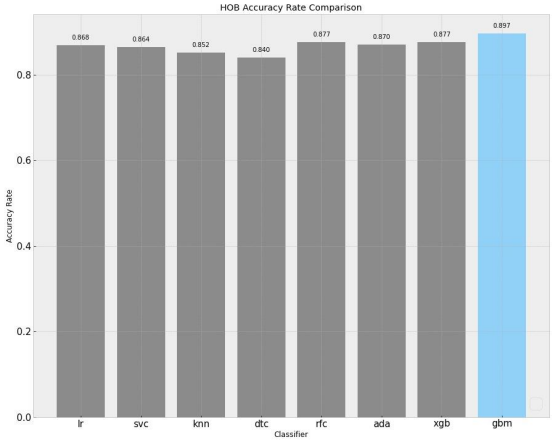

图 4-13　HOB 指标在多个分类模型下的准确率

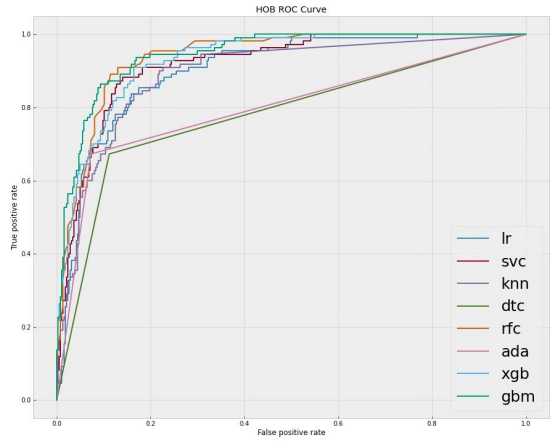

图 4-14　HOB 指标在多个分类模型下的 ROC 曲线

5.MN 指标

MN: ‘1’代表该化合物具有遗传毒性，‘0’代表该化合物不具有遗传毒性。显然，具有遗传毒性的药物病人不希望去使用，我们认为其药理特性不好，所以指标值为 1 时，我们认为其 ADMET 性质不好；指标值为 0 时，我们认为其 ADMET 性质较好。针对该指标，我们首先观察数据，一共 1974 个化合物，1514 个化合物取 1 值，460 个指标取 0 值，我们认为取 0 值的样本太少，存在样本不平衡问题，所以考虑增加样本。同时，由于样本本身数目就不多，所以使用过采样技术是较为合理的，在这里我们使用合成少数类过采样技术（SMOTE）。

SMOTE 算法是基于随机过采样算法的一种改进方案，由于随机过采样采取简单复制样本的策略来增加少数类样本，这样容易产生模型过拟合的问题，即使得模型学习到的信息过于特别 (Specific) 而不够泛化 (General)，SMOTE 算法的基本思想是对少数类样本进行分析并根据少数类样本人工合成新样本添加到数据集中，具体实现:

(1) 对于少数类中每一个样本 x，以欧氏距离为标准计算它到少数类样本集

中所有样本的距离，得到其 k 近邻。

(2) 根据样本不平衡比例设置一个采样比例以确定采样倍率 N，对于每一个少数类样本 x，从其 k 近邻中随机选择若干个样本，假设选择的近邻为 x_n。

(3) 对于每一个随机选出的近邻 x_n，分别与原样本按照如下的公式构建新的样本。如下：

$$x_{new} = x + rand(0,1) \times (\widetilde{x} - x) \tag{11}$$

对增加完样本后的数据，使用前文中提到的 8 个模型进行模拟，最终结果同样是采用随机森林所建立模型预测效果最好，验证集准确率为 0.971830985915493。

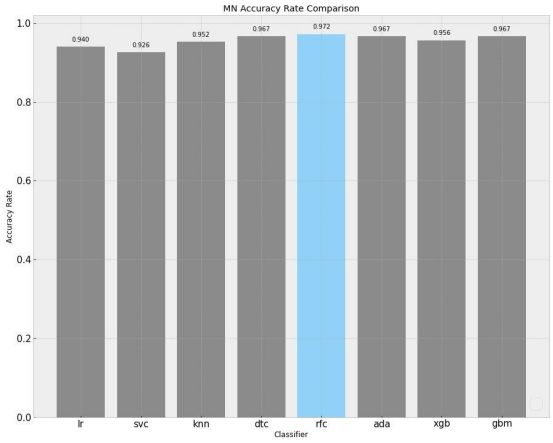

图 4-15　MN 指标在多个分类模型下的准确率

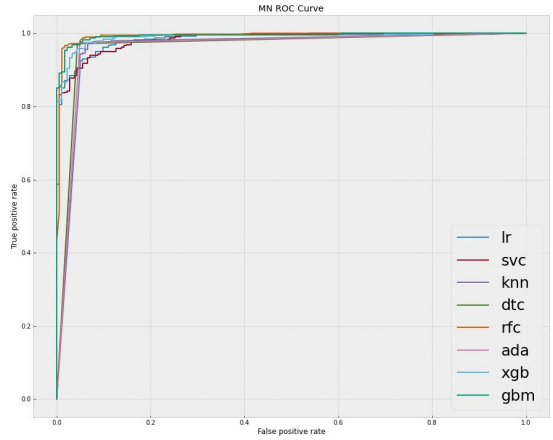

图 4-16　MN 指标在多个分类模型下的 ROC 曲线

以上就是我们对想要预测这五个 ADMET 性质所操作的具体流程，我们也是根据这五个拟合模型，对预测集—“ADMET.xlsx”的 test 表进行预测的，具体的结果已经补充在附件’ADMET.xlsx’ 中。

SMILES	IC50_nM	pIC50
COc1cc(OC)cc(\C=C\c2ccc(OS(=O)(=O)[C@@H]3C[C@@H]4O[C@H]3C(=C4c5ccc(O)	28.33260253	7.547713531
OC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3ccccc23)c4ccc(O)cc4	29.10627424	7.536013383
COc1ccc2C(=C(CCOc2c1)c3ccc(O)cc3)c4ccc(\C=C\C(=O)O)cc4	74.5128189	7.127769006
OC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	28.76984969	7.541062407
OC(=O)\C=C\c1ccc(cc1)C2=C(CCSc3cc(F)ccc23)c4ccc(O)cc4	31.29893761	7.504470404
CC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	131.4355395	6.881287188
Oc1ccc(cc1)C2=C(c3ccc(\C=C\c4ccccc4)cc3)c5ccc(F)cc5OCC2	27.28632274	7.564054988
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)c4ccccc4)cc3)c5ccc(F)cc5OCC2	25.85048389	7.587531323
OC(=O)\C=C\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	131.4427067	6.881263506
CCN(CC)C(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	35.36639225	7.451409241
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)N4CCCC4)cc3)c5ccc(F)cc5OCC2	21.55990995	7.666353058
CCN(CC)CCNC(=O)\C=C\c1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	66.53242701	7.176966634
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)N4CCNCC4)cc3)c5ccc(F)cc5OCC2	58.85652524	7.230205381
CN1CCN(CC1)C(=O)\C=C\c2ccc(cc2)C3=C(CCOc4cc(F)ccc34)c5ccc(O)cc5	40.71609161	7.390233917
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)N4CCN(Cc5ccccc5)CC4)cc3)c6ccc(F)cc6OCC2	20.38412465	7.690707934
Cc1ccc(cc1)N2CCN(CC2)C(=O)\C=C\c3ccc(cc3)C4=C(CCOc5cc(F)ccc45)c6ccc(O)cc6	21.77338567	7.662074035
Oc1ccc(cc1)C2=C(c3ccc(\C=C\C(=O)Nc4ccccc4)cc3)c5ccc(F)cc5OCC2	22.88773579	7.640397169
OC(=O)COc1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	123.8742907	6.907018819
Oc1ccc(cc1)C2=C(c3ccc(C=O)cc3)c4ccc(F)cc4OCC2	153.6897346	6.813355139
CCC(=C(c1ccc(O)cc1)c2ccc(\C=C\C(=O)O)cc2)c3ccccc3	6.369478776	8.195896105
OC(=O)CCCOc1ccc(cc1)C2=C(CCOc3cc(F)ccc23)c4ccc(O)cc4	133.545801	6.874369763
COc1ccc2C(=C(CCOc2c1)c3ccc(O)cc3)c4ccc(OCC(=O)O)cc4	172.6103841	6.762933081
CCCC(CCC)(c1ccc(O)c(C)c1)c2cccs2	172.4192165	6.763414333
CCCC(CCC)(c1ccc(O)c(C)c1)c2cc(C)cs2	72.95012421	7.136973964
CCCC(CCC)(c1ccc(O)c(C)c1)c2ccc([nH]2)C(=O)OCC	131.4899091	6.881107575
CCCC(CCC)(c1ccc(O)c(C)c1)c2ccc(C(=O)OCC)n2C	107.5566386	6.968362779
CCCC(CCC)(c1ccc(O)c(C)c1)c2c[nH]c3cc(OCc4ccccc4)ccc23	70.68743291	7.15065779

图 4-17　附件 ADMET.xlsx 中部分数据

4.4 问题四

4.4.1 问题描述和分析

针对该问，我们要同时满足两个条件：

1、要保证优良的 ADMET 性质，对此，我们考虑用数学公式将 ADMET 性质的五个指标值进行限制。在之前的分析中，我们可以知道，CYP3A4、hERG 和 MN 指标这三个指标为零时，说明相应的药理性质较差。我们将五个 ADMET 性质，相应的定义为自变量 K_1 到 K_5 ，那么此时，至少三个性质较好就等价于 $K_2 + K_3 + K_5 <= K_1 + K_4$ ，具体实现时也是这样限制的。

2、要尽可能搜索化合物的分子描述符的范围，使得相应的化合物生物活性尽可能高。

4.4.2 模型建立与求解

我们所考虑的方法是在局部最优解的范围进行搜索，寻找相应的分子描述符范围，即：我们将这 1974 个化合物的生物活性从高到低进行排序，之后对每个化合物，验证其是否满足至少三个 ADMET 性质良好这个条件，不行的话就寻找下一个，直到寻找到符合条件的化合物为止。这样，我们找到了既满足至少三个 ADMET 性质良好同时生物活性也较高的化合物，我们将其设定为初始值。之后进行小范围搜索。

考虑到第一二问我们已经将 749 个分子描述式通过变量选择减少到了 20 个，所以此时我们也仅考虑这 20 个选择后的分子描述式的应如何取值。下面介绍我们的搜索方法：

我们将这 20 个分子描述式的初始值都设为我们前面选择的化合物相应的分子描述式的值，然后我们将其中 19 个固定，对剩下的那一个进行搜索。若该分子描述式是一个连续性变量，我们以该分子描述式值的 1/10 作为步长分别向上向下进行搜索，若该分子描述式是一个离散型变量，我们以 1 作为步长分别向上向下进行搜索每走一个步长，搜索边界限定在该分子描述符在 training 集中的最

大值和最小值内。我们就利用第三问建立的 5 个模型，对新的 5 个 ADMET 性质值进行估计，然后判断其是否满足至少三个 ADMET 性质较好。如果条件满足，那么此时再利用第二问建立的生物活性回归模型，计算相应的生物活性，如果该生物活性不低于初始条件下的生物活性，我们就认为该分子描述式的取值是我们想要的，就纳入取值范围。类似的，每次固定 19 个分子描述式值，对剩下的那个进行搜索，我们就可以得出剩下的分子描述式的取值范围，最终就可以得到这 20 个分子描述式的取值范围:

```
"BCUTc-1l" found border: (-0.3975387594756116, -0.21683932335033354)
"SHsOH" found border: (0.11410088392948198, 2.7384212143075684)
"SsOH" found border: (1.3322676295501878e-15, 63.99262188130527)
"minHsOH" found border: (-0.11410088392948231, 0.7416557455416356)
"maxHsOH" found border: (9.71445146547012e-17, 0.7987061875063768)
"maxsOH" found border: (6.661338147750939e-16, 12.233883594955426)
"minsOH" found border: (6.661338147750939e-16, 11.2928156261127)
"SssO" found border: (0.0, 30.02512934925577)
"MLFER_A" found border: (-0.7623, 7.731900000000011)
"MDEC-23" found border: (2.1169006015595064, 52.92251503898773)
"WTPT-5" found border: (0.0, 125.73768843879981)
"maxssO" found border: (0.0, 6.659866009829617)
"MLogP" found border: (1.6099999999999945, 5.473999999999984)
"MDEC-22" found border: (1.3322676295501878e-15, 34.15356361193756)
"nC" found border: (7.0, 95.0)
"nBase" found border: (0.0, 26.0)
"minssO" found border: (0.0, 6.726366431788502)
"LipoaffinityIndex" found border: (-4.398448537853795, 22.725317445577947)
"minsssN" found border: (0.0, 2.734772117779864)
"BCUTc-1h" found border: (0.0738880396051771, 0.5246050811967577)
```

5. 模型的评价

5.1 模型的优点

（1）本文对所有分子描述符数据首先进行预处理，保证了数据的质量。

（2）针对问题 2，本文运用了目前经常用在提高模型精度的 stacking + ensemble 的组合，将多个弱分类器进行集成再与多个强分类器进行融合，比起使用一个单一模型，可以在保持模型准确度的情况下提高模型的泛化能力。

（3）针对问题 3，本文对比了先有的多种分类算法以选择针对一特定数据最合适的方法，比起统一用一个模型来做预测更加有针对性。除了常用的准确率作为模型评价标准，引入一个更为科学的标准 AUC 值。对于正负样本比例不一致的 MN 性质的样本进行了 SMOTE 抽样调整，去平衡正负样本的比例

（4）针对问题 4，本文设计了一个贪心算法，在一个既定的水平周围进行遍历，尽可能的搜索到足够大的范围。

5.2 模型的缺点

(1) 只考虑了题目所提供的所有数据，而没有去考虑分子结构式等生化性质，而实际中化合物的分子结构肯定是会对其性质造成影响的。

(2) 运用到强分类模型时，因为时间的限制没有做到特别合适的调参，只寻找了一个既定范围内的局部最优的参数

（3）由于本题属于小样本的数据分析，容易带来模型的过拟合问题，本文在防止过拟合的措施上研究不够多。

6. 参考文献

[1] 数据挖掘：概念与技术第三版 [M]. 机械工业出版社,2012.

[2] Kaggle-Ensemble-Guide[J/OL].https://mlwave.com/kaggle-ensembling-guide/, 2015-06-11
[3] 汪云云, 陈松灿. 基于 AUC 的分类器评价和设计综述 [J]. 模式识别与人工智能,2011,24(01):64-71.

附录 A 第一问程序代码

```
ssc = StandardScaler()
fX = mol.values
fX_trans = ssc.fit_transform(fX)
fy = erapi.values

# remove features with zero variance
n_zero_var = 0
zero_var_cols = []
for i in range(len(sel_vars)):
   if sel_vars[i] == 0:
      zero_var_cols.append(mol.columns[i])
      n_zero_var += 1
fil_cols = list(set(mol.columns).difference(set(zero_var_cols)))
mol_fil = mol[fil_cols].copy()

# randomforest selection
from sklearn.model_selection import cross_val_score, ShuffleSplit
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=2,
   max_depth=4,random_state=300)
rf_scores = []
# select individual feature and do cross-validation
for i in range(fX_fil.shape[1]):
   score = cross_val_score(rf,fX_fil[:, i:i+1], fy,
      scoring="r2", cv=ShuffleSplit(10, test_size=0.3))
   rf_scores.append((format(np.mean(score), '.3f'),
      mol_fil.columns[i]))
rf_columns = sorted(rf_scores, reverse=True)[:20]
rf_columns = [v[1] for v in rf_columns]
```

附录 B 第二问程序代码

```
from sklearn.linear_model import ElasticNet, Lasso,
   BayesianRidge, LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,
   GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin,
   RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score,
   train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb

# Validation function
```

```
n_folds = 5
train = train_X
def mse_cv(model):
    kf = KFold(n_folds, shuffle=True,
        random_state=100).get_n_splits(train_X)
    mse = -cross_val_score(model, train_X, train_y,
        scoring="neg_mean_squared_error", cv = kf)
    return mse

# utilize individual model
lasso = make_pipeline(StandardScaler(), Lasso(alpha =0.005,
    random_state=100))
ENet = make_pipeline(StandardScaler(),
    ElasticNet(max_iter=3000,alpha=0.0005, l1_ratio=.9,
    random_state=100))
KRR = GridSearchCV(KernelRidge(), param_grid=[
            {'alpha':[0.1,0.3,0.9,1.2,1.5,2.0],
                'degree':[2,3,4],'coef0':[1,2.5]}
         ],cv=3,scoring='neg_mean_squared_error')

RFR = RandomForestRegressor(n_estimators=100,criterion='mse',
    random_state=0)

GBoost = GradientBoostingRegressor(n_estimators=3000,
    learning_rate=0.05,
                           max_depth=4, max_features='sqrt',
                           min_samples_leaf=15,
                              min_samples_split=10,
                           loss='huber', random_state =5)

model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603,
    gamma=0.0468,
                       learning_rate=0.05, max_depth=3,
                       min_child_weight=1.7817, n_estimators=2200,
                       reg_alpha=0.4640, reg_lambda=0.8571,
                       subsample=0.5213, silent=1,
                       random_state =7, nthread = -1)

model_lgb =
    lgb.LGBMRegressor(objective='regression',num_leaves=5,
                        learning_rate=0.05, n_estimators=720,
                        max_bin = 55, bagging_fraction = 0.8,
                        bagging_freq = 5, feature_fraction =
                            0.2319,
                        feature_fraction_seed=9, bagging_seed=9,
                        min_data_in_leaf =6,
                            min_sum_hessian_in_leaf = 11)

score = mse_cv(lasso)
print("Lasso score: {:.4f} ({:.4f})\n".format(score.mean(),
    score.std()))
score = mse_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(),
    score.std()))
```

```
score = mse_cv(RFR)
print("RandomForestRegressor score: {:.4f}
    ({:.4f})\n".format(score.mean(), score.std()))
score = mse_cv(GBoost)
print("GBoost score: {:.4f} ({:.4f})\n".format(score.mean(),
    score.std()))
score = mse_cv(KRR)
print("Kernel Ridge score: {:.4f}
    ({:.4f})\n".format(score.mean(), score.std()))

class StackingAveragedModels(BaseEstimator, RegressorMixin,
    TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    # We again fit the data on clones of the original models
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True,
            random_state=156)

        # Train cloned base models then create out-of-fold
            predictions
        # that are needed to train the cloned meta-model
        out_of_fold_predictions = np.zeros((X.shape[0],
            len(self.base_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred

        # Now train the cloned meta-model using the out-of-fold
            predictions as new feature
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self

    #Do the predictions of all base models on the test data and
        use the averaged predictions as
    #meta-features for the final prediction which is done by the
        meta-model
    def predict(self, X):
        meta_features = np.column_stack([
            np.column_stack([model.predict(X) for model in
                base_models]).mean(axis=1)
            for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)

stacked_averaged_models = StackingAveragedModels(base_models =
```

```
    (ENet, RFR ,GBoost),
                                        meta_model = lasso)

score = mse_cv(stacked_averaged_models)
print("Stacking Averaged models score: {:.4f}
    ({:.4f})".format(score.mean(), score.std()))
# stack model performance
stacked_averaged_models.fit(train_X, train_y)
stacked_train_pred = stacked_averaged_models.predict(train_X)
stacked_pred = stacked_averaged_models.predict(test_X)

model_xgb.fit(train_X, train_y)
xgb_train_pred = model_xgb.predict(train_X)
xgb_pred = model_xgb.predict(test_X)

model_lgb.fit(train, train_y)
lgb_train_pred = model_lgb.predict(train_X)
lgb_pred = model_lgb.predict(test_X)

print('Stack model MSE', mean_squared_error(train_y,
    stacked_train_pred))
print('XGBoost regression MSE',mean_squared_error(train_y,
    xgb_train_pred))
print('LightGBM regression MSE: ', mean_squared_error(train_y,
    lgb_train_pred))
print('MSE score on train data:',
    mean_squared_error(train_y,stacked_train_pred*0.70 +
            xgb_train_pred*0.25 + lgb_train_pred*0.05 ))

ensemble = stacked_pred*0.70 + xgb_pred*0.15 + lgb_pred*0.15
# convert pIC50 to IC50_nM: 10^(-y+9)
result = np.power(10,(-ensemble + 9))
```

附录 C 第三问程序代码

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import xgboost as xgb
import lightgbm as lgb
from sklearn import metrics

def LR(X_train,y_train,X_test,y_test):
    # Standardization
    ssc = StandardScaler()
```

```
    X_train_trans = ssc.fit_transform(X_train)
    X_test_trans = ssc.transform(X_test)
    lr = LogisticRegression()
    lr.fit(X_train_trans,y_train)
    y_prob = lr.predict_proba(X_test_trans)[:,1]
    y_pred = lr.predict(X_test_trans)
    fpr_lr,tpr_lr,threshold_lr = metrics.roc_curve(y_test,y_prob)
    auc_lr = metrics.auc(fpr_lr,tpr_lr)
    score_lr = metrics.accuracy_score(y_test,y_pred)
    print([score_lr,auc_lr])
    return lr, fpr_lr,tpr_lr,auc_lr,score_lr

def SVC_(X_train,y_train,X_test,y_test):
    # SVM with standardization
    ssc = StandardScaler()
    X_train_trans = ssc.fit_transform(X_train)
    X_test_trans = ssc.transform(X_test)
    kernelList = ['linear','rbf','sigmoid']
    svc = None
    auc_svc = -1
    score_svc = -1
    fpr_svc = None
    tpr_svc = None
    for kernel in kernelList:
        svc_tmp = SVC(kernel=kernel).fit(X_train_trans,y_train)

        # decision border distance
        y_prob = svc_tmp.decision_function(X_test_trans)
        y_pred = svc_tmp.predict(X_test_trans)
        # false positive, true positive
        fpr_svc_tmp,tpr_svc_tmp,threshold_svc_tmp =
            metrics.roc_curve(y_test,y_prob)
        # auc curve
        auc_svc_tmp = metrics.auc(fpr_svc_tmp,tpr_svc_tmp)
        score_svc_tmp = metrics.accuracy_score(y_test,y_pred)
        print([score_svc_tmp,auc_svc_tmp])

        if not svc:
            svc = svc_tmp
        if auc_svc < auc_svc_tmp:
            auc_svc = auc_svc_tmp
            best_svc = svc
            score_svc = score_svc_tmp
            fpr_svc = fpr_svc_tmp
            tpr_svc = tpr_svc_tmp
    return svc, fpr_svc, tpr_svc, auc_svc, score_svc

def KNN(X_train,y_train,X_test,y_test):
    # KNN with standardization
    ssc = StandardScaler()
    X_train_trans = ssc.fit_transform(X_train)
    X_test_trans = ssc.transform(X_test)

    # find best K value
```

```
    score_K=[]
    KList=range(2,10)
    for k in KList:
        knn =
            KNeighborsClassifier(n_neighbors=k,weights='distance').fit(X_train_tran
        score_K.append(knn.score(X_test_trans,y_test))
    print('K={}, get highest
        score={}'.format(KList[score_K.index(max(score_K))],max(score_K)))

    knn =
        KNeighborsClassifier(n_neighbors=KList[score_K.index(max(score_K))],
                             weights='distance').fit(X_train_trans,y_train)

    y_prob = knn.predict_proba(X_test_trans)[:,1]
    y_pred = knn.predict(X_test_trans)
    fpr_knn,tpr_knn,threshold_knn =
        metrics.roc_curve(y_test,y_prob)
    auc_knn = metrics.auc(fpr_knn,tpr_knn)
    score_knn = metrics.accuracy_score(y_test,y_pred)
    print([score_knn,auc_knn])
    return knn, fpr_knn, tpr_knn, auc_knn, score_knn

def DTC(X_train,y_train,X_test,y_test):
    # decision tree
    dtc = tree.DecisionTreeClassifier()
    dtc.fit(X_train,y_train)
    y_prob = dtc.predict_proba(X_test)[:,1]
    y_pred = dtc.predict(X_test)
    fpr_dtc,tpr_dtc,threshod_dtc= metrics.roc_curve(y_test,y_prob)
    score_dtc = metrics.accuracy_score(y_test,y_pred)
    auc_dtc = metrics.auc(fpr_dtc,tpr_dtc)
    print([score_dtc,auc_dtc])
    return dtc, fpr_dtc, tpr_dtc, auc_dtc, score_dtc

def RFC(X_train,y_train,X_test,y_test):
    # random forest classifier
    rfc = RandomForestClassifier()
    rfc.fit(X_train,y_train)
    y_prob = rfc.predict_proba(X_test)[:,1]
    y_pred = rfc.predict(X_test)
    fpr_rfc,tpr_rfc,threshold_rfc =
        metrics.roc_curve(y_test,y_prob)
    auc_rfc = metrics.auc(fpr_rfc,tpr_rfc)
    score_rfc = metrics.accuracy_score(y_test,y_pred)
    print([score_rfc,auc_rfc])
    return rfc, fpr_rfc, tpr_rfc, auc_rfc, score_rfc

def ADA(X_train,y_train,X_test,y_test):
    # AdaBoost Classifier
    # use weak classifier
    adaclf = AdaBoostClassifier(
        DecisionTreeClassifier(max_depth=1),
        n_estimators=100
    )
```

```
    adaclf.fit(X_train, y_train)
    y_pred = adaclf.predict(X_test)
    # print('adaboost confusion matrix:')
    conf_mat = confusion_matrix(y_test, y_pred)
    fpr_ada,tpr_ada,threshold_ada =
       metrics.roc_curve(y_test,y_pred)
    auc_ada = metrics.auc(fpr_ada,tpr_ada)
    score_ada = metrics.accuracy_score(y_test,y_pred)
    print('[{},{}]'.format(score_ada,auc_ada))
    return adaclf, fpr_ada, tpr_ada, auc_ada, score_ada

def XGB(X_train,y_train,X_test,y_test):
    # XGBoost
    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest=xgb.DMatrix(X_test)

    params={'booster':'gbtree',
       'objective': 'binary:logistic',
       'eval_metric': 'auc',
       'max_depth':4,
       'lambda': 10,
       'gamma':0.1,
       'subsample':0.75,
       'colsample_bytree':0.75,
       'min_child_weight':3,
       'eta': 0.025,
       'seed':10,
       'nthread':-1,
        'silent':1,
       'verbosity':0}

    watchlist = [(dtrain,'train')]
    bst=xgb.train(params,dtrain,num_boost_round=100,evals=watchlist)

    ypred=bst.predict(dtest)
    y_pred = (ypred >= 0.5)*1

    fpr_xgb,tpr_xgb,threshold_xgb =
       metrics.roc_curve(y_test,ypred)
    auc_xgb = metrics.auc(fpr_xgb,tpr_xgb)
    score_xgb = metrics.accuracy_score(y_test,y_pred)
    print ('AUC: %.4f' % metrics.auc(fpr_xgb,tpr_xgb))
    print ('ACC: %.4f' % metrics.accuracy_score(y_test,y_pred))
    print(metrics.confusion_matrix(y_test,y_pred))
    return bst, fpr_xgb, tpr_xgb, auc_xgb, score_xgb

def LGBM(X_train,y_train,X_test,y_test):
    # LightGBM
    dtrain = lgb.Dataset(X_train, label=y_train)
    dtest=lgb.Dataset(X_test,label=y_test)

    params = {'num_leaves': 60,
            'min_data_in_leaf': 20,
            'objective': 'binary',
```

```
            'max_depth': -1,
            'learning_rate': 0.03,
            "min_sum_hessian_in_leaf": 6,
            "boosting": "gbdt",
            "feature_fraction": 0.9,
            "bagging_freq": 1,
            "bagging_fraction": 0.8,
            "bagging_seed": 11,
            "lambda_l1": 0,
            'lambda_l2': 0.001,
            "verbosity": 0,
            "nthread": -1,
            'metric': {'binary_logloss', 'auc'},
            "random_state": 2021,
            }

    gbm = lgb.train(params,
                    dtrain,
                    num_boost_round=100,
                    valid_sets=dtest,
                    early_stopping_rounds=50)
    ypred=gbm.predict(X_test)
    y_pred = (ypred >= 0.5)*1

    fpr_gbm,tpr_gbm,threshold_gbm =
        metrics.roc_curve(y_test,ypred)
    auc_gbm = metrics.auc(fpr_gbm,tpr_gbm)
    score_gbm = metrics.accuracy_score(y_test,y_pred)
    print ('AUC: %.4f' % metrics.auc(fpr_gbm,tpr_gbm))
    print ('ACC: %.4f' % metrics.accuracy_score(y_test,y_pred))
    return gbm, fpr_gbm, tpr_gbm, auc_gbm, score_gbm


def plot_acc_curve(prs,name,best_mn):
    # comparison between different approaches
    plt.style.use('bmh')
    plt.figure(figsize=(15,12))
    #设置坐标刻度值的大小以及刻度值的字体
    plt.tick_params(labelsize=15)

    name_items = []
    score_items = []
    for k,v in prs.items():
        name_items.append(k)
        score_items.append(v[2])
    colors = ['grey' if n!= best_mn else 'lightskyblue' for n in
        name_items]

    plt.bar(name_items,score_items,align='center',width=0.8,
          color=colors,alpha=0.9)
    indexs=np.arange(len(name_items))
    for x,y in zip(indexs,score_items):
        plt.text(x, y+0.01,'%.3f' % y, ha='center',va='bottom')
```

```
    plt.legend(loc='lower right',prop={'size':25})
    plt.xlabel('Classifier',fontsize=12)
    plt.ylabel('Accuracy Rate',fontsize=12)
    plt.title('{} Accuracy Rate Comparison'.format(name))
    plt.savefig('{}_acc_score.jpg'.format(name))
    plt.show()

def plot_roc_curve(prs,name):
    plt.style.use('bmh')
    plt.figure(figsize=(15,12))
    for k,v in prs.items():
        plt.plot(v[0],v[1],label=k)

    plt.legend(loc='lower right',prop={'size':25})
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('{} ROC Curve'.format(name))
    plt.savefig('{}_roc_curve.jpg'.format(name))
    plt.show()

def clf_train(X,y,name):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.25,random_state=100)
    model_params = (X_train,y_train,X_test,y_test)

    lr, fpr_lr,tpr_lr,auc_lr,score_lr = LR(*model_params)
    svc, fpr_svc ,tpr_svc,auc_svc,score_svc = SVC_(*model_params)
    knn, fpr_knn,tpr_knn,auc_knn,score_knn = KNN(*model_params)
    dtc, fpr_dtc,tpr_dtc,auc_dtc,score_dtc = DTC(*model_params)
    rfc, fpr_rfc,tpr_rfc,auc_rfc,score_rfc = RFC(*model_params)
    ada, fpr_ada,tpr_ada,auc_ada,score_ada = ADA(*model_params)
    bst, fpr_xgb,tpr_xgb,auc_xgb,score_xgb = XGB(*model_params)
    gbm, fpr_gbm,tpr_gbm,auc_gbm,score_gbm = LGBM(*model_params)

    ret = {
        'lr': (fpr_lr,tpr_lr,score_lr,auc_lr,lr),
        'svc':(fpr_svc,tpr_svc,score_svc,auc_svc,svc),
        'knn':(fpr_knn,tpr_knn,score_knn,auc_knn,knn),
        'dtc':(fpr_dtc,tpr_dtc,score_dtc,auc_dtc,dtc),
        'rfc':(fpr_rfc,tpr_rfc,score_rfc,auc_rfc,rfc),
        'ada':(fpr_ada,tpr_ada,score_ada,auc_ada,ada),
        'xgb':(fpr_xgb,tpr_xgb,score_xgb,auc_xgb,bst),
        'gbm':(fpr_gbm,tpr_gbm,score_gbm,auc_gbm,gbm)
    }

    return ret

def clf_predict(ret,Xtest,name):
    plot_roc_curve(ret,name)
    # AUC score comparison
    best_mn = ''
    best_m = None
    highest_score = -1
    for k,v in ret.items():
```

```
        # compare accuracy first
        if v[2] > highest_score:
            highest_score = v[2]
            highes_auc_score = v[-2]
            best_mn = k
            best_m = v[-1]
        elif v[2] == highest_score:
            if v[-2] > highes_auc_score:
                highest_score = v[2]
                highes_auc_score = v[-2]
                best_mn = k
                best_m = v[-1]
    print('{} get highest acc score({}), applied to compound: {},
        auc_score({})'
          .format(best_mn,highest_score,name,highes_auc_score))

    if best_mn == 'xgb':
        Dtest = xgb.DMatrix(Xtest)
        y_final_test = (best_m.predict(Dtest) >= 0.5) * 1
    elif best_mn == 'gbm':
        y_final_test = (best_m.predict(Xtest) >= 0.5) * 1
    else:
        y_final_test = best_m.predict(Xtest)
    best_item = {name:(best_mn,best_m)}
    plot_acc_curve(ret,name,best_mn)
    return y_final_test, best_item


# 比较各个分类器的效果，绘制准确度柱状图和 ROC曲线图
compound_names = ['Caco-2','CYP3A4','hERG','HOB']
compound_rets = []

# training different model in different compounds
for name in compound_names:
    compound_rets.append(clf_train(fX,
        adm_train[name].values,name))


from imblearn.over_sampling import SMOTE
from collections import Counter

# 对于样本不平衡的 ‘MN’化合物的负样本进行过采样处理
smo = SMOTE(sampling_strategy=0.5,random_state=100)
fX_smo, y_mn_smo = smo.fit_resample(fX,adm_train['MN'])
fX_smo.shape, y_mn_smo.shape

mn_ret = clf_train(fX_smo,y_mn_smo,'MN')
compound_names.append('MN')
compound_rets.append(mn_ret)
```

附录 D 第四问程序代码

```
#
    筛选出至少满足三种ADMET性质的样本，按照pIC50生物活性值逆序排列，找到活性最好那个样本
d_adm['pIC50'] = d_pic['pIC50']
d_adm['dis'] = d_adm.apply(lambda x: x[cols[1]] + x[cols[2]]
    +x[cols[4]] - x[cols[0]] - x[cols[3]],axis=1)
d_fil = d_adm[d_adm['dis'] <= 0]
suit_index =
    d_fil.sort_values(by='pIC50',ascending=False).index[0]
# suit_sample = d_mol[rf_columns].iloc[suit_index]
suit_sample = d_mol.iloc[suit_index]
suit_src_admet = d_adm.iloc[suit_index]

def sat_con(ap):
    return ap[1] + ap[2] + ap[4] - ap[0] - ap[3] <= 0
def get_pred_admet(sample):
    admet_pred = []
    for item in compound_best_items:
        item = list(item.values())[0]
        m_name = item[0]
        m = item[1]
        if m_name == 'xgb':
            Dtest = xgb.DMatrix(sample)
            pred = (m.predict(Dtest) >= 0.5) * 1
        elif m_name == 'gbm':
            pred = (m.predict(sample) >= 0.5) * 1
        else:
            pred = m.predict(sample)
        admet_pred.append(pred[0])
    return admet_pred

def find_border(col,x,ix,itval,ty):
    ori_val = x[0,ix]
    max_val = d_mol[col].max()
    min_val = d_mol[col].min()

    left = ori_val
    right = ori_val
    # when original value == 0
    if itval == 0.0:
        itval = abs(max_val / 100.0)
        if itval == 0: return {'column':col,'type':'连续',
            'value':0.0}

    # start looping to find border
    while x[0,ix] >= min_val and sat_con(get_pred_admet(x)) :
        x[0,ix] -= itval
    left = x[0,ix] + itval
    x[0,ix] = ori_val
    while x[0,ix] <= max_val and sat_con(get_pred_admet(x)) :
        x[0,ix] += itval
    right = x[0,ix] - itval
    print('"{}" found border: ({}, {})'.format(col,left,right))
    return {'column':col,'type':ty,'left':left,'right':right}
```

```
# x2 + x3 + x5 - x1 - x4 <= 0
ssam = suit_sample.values.reshape(1,-1)
areas_col = []
for col in fea_columns:
   x = ssam.copy()
   ix = d_mol.columns.to_list().index(col)
   # categorical or continuous
   if d_mol[col].dtype == int:
      itval = 1
      areas_col.append(find_border(col,x,ix,itval,'离散'))

   elif d_mol[col].dtype == float:
      itval = abs(x[0,ix] / 10.0)
      areas_col.append(find_border(col,x,ix,itval,'连续'))

print('最终的分子描述符及其取值范围: ', areas_col
```