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	华中科技大学
参赛队号	**21104870008**
队员姓名	**1.张　星**
	2.胡高阳
	3.杨　博

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：

在药物研发中，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，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。目前，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。本文通过分子描述符大数据的挖掘过程，阐述了基于 python 程序筛选拮抗 ERα 活性化合物的主要方法。主要包括以下几个部分：

(1) 首先从 1974 个化合物的 729 个分子描述符信息（即自变量）中，通过方差检验、皮尔逊相关性分析、p-value 值显著性检验等方法，筛选出 253 个分子描述符。以 ERα 的 pIC50 值作为目标函数，采用 Xgboost 算法对分子描述符进行决策分类训练，并与逐步回归线性回归方法的预测效果进行对比。发现 Xgboost 方法最优，其预测准确度为 91.7%。通过 Xgboost 计算 Shap 值，并对其排序选出前 20 个最重要的分子描述符。

(2) 其次，采用筛选出的 20 个分子描述符，通过 Xgboost 方法进行训练。通过机器学习参数调优、交叉验证等手段，得到最优模型，对目标函数的预测准确度为 91.2%。利用此模型对 test 数据进行预测。

(3) 然后，针对化合物的 ADMET 属性，采用 Xgboost 对目标函数进行二值分类训练，通过机器学习参数调优、交叉验证等手段，得到最优模型，并采用逻辑回归(Logistic Regression)进行效果验证，验证结果为 Xgboost 方法最优，且对目标函数的预测准确度为 90 %以上。利用此模型对 test 数据进行预测。

(4) 最后，针对化合物的综合 ADMET 属性，将其综合属性归纳为 1,2,3,4,5 类，对数据筛选后，分别采用 Xgboost 方法对 ERα 的 pIC50 值与综合 ADMET 属性进行分类决策训练。通过比较两个训练模型的前 40 个最大 Shap 绝对值，挑选出 15 个有交集的分子描述符。对 15 个分子描述符进行线性回归拟合，分别得到 pIC50 值与综合 ADMET 属性的数学模型；采用线性规划求出满足 ADMET 属性大于等于 3 与 pIC50 值最大时的分子描述符取值范围，即为分子描述符的最优取值范围。

通过以上方法，可为药物筛选与优化药物结构提供参考。

目录

1、背景简介

乳腺癌是女性最常见的恶性肿瘤之一，主要由乳腺导管内上皮或腺泡上皮细胞恶变而转移，致死率高[1,2]。研究表明，乳腺癌的发展与雌激素受体密切相关，其中雌激素受体 α 亚型 (Estrogen receptors alpha, ERα) 在正常乳腺上皮细胞中的表达不超过 10%，而在 50%-80%的乳腺肿瘤细胞中表达。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。

药物的特性通常是由原子的个数、种类、空间构型以及各原子或各基团之间的化学键性质所决定[3]。此外，合成药物的各类环境行为效应、生态毒理特性又和其性质密切相关，除了良好的生物活性外，还需要在人体内具备良好的药代动力学性质和安全性，合称为 ADMET (Absorption 吸收、Distribution 分布、Metabolism 代谢、Excretion 排泄、Toxicity 毒性) 性质。为节约时间和成本，通常采用化合物的定量结构-活性关系 (Quantitative Structure-Activity Relationship, QSAR) 模型[4]指导已有药物的结构优化[5-7]。随着计算机技术的发展，机器学习方法已经广泛应用于辅助药物设计[8-10]，为药物的研发提供了新思路。

Boosting 是最常用的集成学习算法之一，它能够把大量准确率不高的模型进行组合，然而在处理复杂数据集时，其迭代次数可能在数千万次以上，效率不高。与传统算法相比，XGBoost 算法在原损失函数的基础上添加了正则化项，类似于树进行了剪枝并限制了叶结点上的分数，有效地防止过拟合[11]。此外，XGBoost 算法在预测精度以及训练速度等方面有了较大的突破[12,13]，其高效、灵活且可移植、能进行大规模并行运算等特点，已被广泛应用于工程、图像处理、医学等领域[12-16]。

2、方法介绍

变量标准化处理：

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i \tag{1}$$

$$s=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i-\bar{x}\right)^2} \tag{2}$$

$$y_i=\frac{x_i-\bar{x}}{s} \tag{3}$$

式中，x_i 是数据变量，$\bar{x}$ 是数据 x_i 的平均值，n 是数据个数，s 是标准差，y_i 是标准化数据变量。

皮尔逊相关性分析：

$$r=\frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2\sum(y-\bar{y})^2}} \tag{4}$$

式中，r 是皮尔逊(Pearson)乘积矩相关系数，$\bar{x}$ 、$\bar{y}$ 分别是数据 x，y 的平均值

Xgboost：Xgboost 使用 K 个分类和回归树的集成[7]，最终的预测为每棵树的预测得分之和，集成算法表示为：

$$\hat{y}_i^{(t)}=\sum_{k=1}^{t}f_k\left(x_i\right)=\hat{y}_i^{(t-1)}+f_t\left(x_i\right) \tag{5}$$

式中，x_i 表示样本 i 对应的特征取值，$f_k(x_i)$表示叶子权重，表示第 k 棵树对于样本 i 的预测值，Xgboost 的预测结果就是 k 棵树的叶子权重之和。

Xgboost 的优势在于在模型构建初期就加入了正则项，使得模型更加的简单，而达到决策树剪枝的一个效果。因此 Xgboost 的目标函数由两部分构成，一是减少模型偏差的损失函数，二是抑制模型复杂度的正则项。Xgboost 算法的目标函数为：

$$L(\phi)=\sum_{i=1}^{n}l\left(y_i,\hat{y}_i\right)+\sum_{k=1}^{K}\Omega\left(f_k\right) \tag{6}$$

式中，$\sum_{i=1}^{n}l\left(y_i,\hat{y}_i\right)$ 为损失值，$\sum_{k=1}^{K}\Omega\left(f_k\right)$ 为正则项。

结合式 6 可以将目标函数改写为：

$$Obj^t=\sum_{i=1}^{n}l\left(y_i,\hat{y}^{(t-1)}+f_t\left(x_i\right)\right)+\Omega\left(f_t\right) \tag{7}$$

由泰勒展开得到：

$$Obj^t \approx \sum_{i=1}^{n}\left[l\left(y_i, \hat{y}^{(t-1)}\right) + g_i f_t\left(x_i\right) + \frac{1}{2} h_i f_t^2\left(x_i\right)\right] + \Omega\left(f_t\right) \tag{8}$$

式中，$g_i = \partial_{\hat{y}^{-1}} l\left(y_i, \hat{y}^{(t-1)}\right)$，$h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$移除常数项，简化为下式：

$$Obj^t \approx \sum_{i=1}^{n}\left[g_i f_t\left(x_i\right) + \frac{1}{2} h_i f_t^2\left(x_i\right)\right] + \Omega\left(f_t\right) \tag{9}$$

由样本上遍历转化为叶子结点上遍历：

$$Obj^t \approx \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T \tag{10}$$

式中，T 表示样子结点个数，g_i 表示第 i 个样本叶子结点的值，w_j 表示第 j 个叶子结点的权重值。令 $G_j = \sum_{i\in I_j} g_i$，$H_j = \sum_{i\in I_j} h_i$ 则：

$$Obj^t \approx \sum_{j=1}^{T}\left[\left(G_j w_j + \frac{1}{2}\left(H_j + \lambda\right) w_j^2\right] + \gamma T \tag{11}$$

由 $\frac{\partial J\left(f_t\right)}{\partial w_j} = G_j + \left(H_j + \lambda\right) w_j = 0$ 得到权重最优解：

$$w_j = -\frac{G_j}{H_j + \lambda} \tag{12}$$

将上式带入原函数：

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{13}$$

目标函数Obj主要用于对树的结构进行打分，分数越小效果就越好。由于不可能列举出所有可能的树结构进行打分，这里可以利用分割后的左L和右R节点的实例集的得分对分割候选节点进行评估，用以确定是否进行分割，采用如下分裂方式可以使得目标函数降低：

$$Obj_{split} = -\frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{\left(G_L + G_R\right)^2}{H_L + H_R + \lambda}\right] - \gamma \tag{14}$$

式中，$\frac{G_L^2}{H_L + \lambda}$ 为左子树分数，$\frac{G_R^2}{H_R + \lambda}$ 为右子树分数，$\frac{\left(G_L + G_R\right)^2}{H_L + H_R + \lambda}$ 不可分割的分数，γ 是新叶子节点引入的复杂度代价。

Xgboost 与传统的线性模型相比，在模型的可解释性上，几乎是一个黑箱模型。为量化每个特征对模型所做预测的贡献，可以引入 *SHAP* 值对 Xgboost 进行解释分析[18]。

考虑一个黑箱模型具有 *N* 个特征 *F*，为确定每个特征对模型预测结果的贡献度，将这 *N* 个特征进行排列组合，*SHAP* 需要为每个不同的组合训练一个不同的预测模型，得到 2^N 个预测结果 *S*，如图 1 所示。这些模型在涉及到它们的超参数和训练数据时是完全等价的，唯一改变的是模型中包含的一组特征。

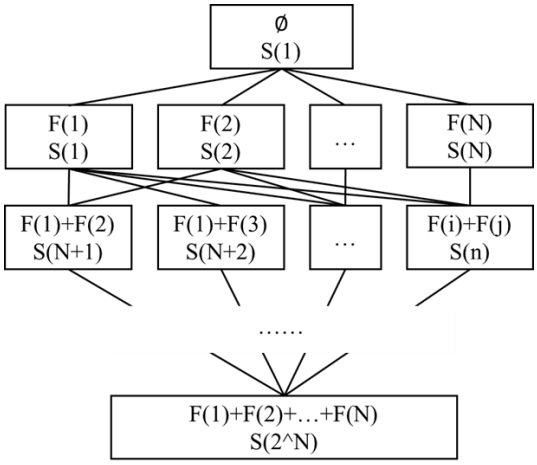


图 1、SHAP 原理示意图

图 1 中由一条边连接的两个节点只存在一个特征的差异，因此两个连接节点的预测之间的差距可以归结为该附加特征的影响，也被称为特性的“边际贡献”。

为计算 F(n)的边际贡献，需要找出所有差异为 F(n)的节点，将其预测结果的差值乘以对应边的权值：

$$\mathrm{SHAP}_{F(n)} = \sum w_n \cdot \left[S(d)_n - S(u)_n \right] \tag{15}$$

式中，w_n为边的权值，$S(d)_n$、$S(u)_n$分别为对应下部节点与上部节点代表的预测值。

对于同一特征而言，同一行上所有权值的和应该等于任意其它行上所有权值的和，同一行上的所有权值应该相等，且全部权值之和等于 1。即第 n 行上权值的计算公式为：

$$w_{n1} = w_{n2} = w_{n3} = \cdots = \left(N \cdot C_{N-1}^{n-1} \right)^{-1} \tag{16}$$

式中，w_n为边的权值，N 为行数，C_{N-1}^{n-1} 为从 N-1 个元素里面任选 n-1 个元素的组合值。

逻辑回归：考虑二分类任务，目标输出为 y∈{0,1}，而线性回归模型产生的预测值如式:

$$z = \boldsymbol{\omega}^T \mathbf{x} + b \tag{17}$$

其输出值 z 为连续实数，需要将其转换为 0/1 值。我们希望可以找到能够在一定程度上近似单位阶跃函数的代价函数，因此引入 Logistic 函数，它将 z 值转化为一个接近 0 或 1 的 y 值，并且输出值在 z=0 附近变化很陡，如下所示：

$$y = \frac{1}{1+e^{-z}} \tag{18}$$

将逻辑回归函数带入线性模型，可得：

$$y = \frac{1}{1+e^{-\left(\omega^T \mathbf{x} + b\right)}} \tag{19}$$

将式 17 转化为：

$$\ln \frac{y}{1-y} = \boldsymbol{\omega}^T \mathbf{x} + b \tag{20}$$

逻辑回归求解的目标函数是任意阶可导的凸函数，可将式 20 中的 y 视为类后验概率来估计 $p(y=1|\ \mathbf{x})$，则将对数变化后的式子重新写为：

$$\ln\frac{p(y=1|\ \mathbf{x})}{p(y=0|\ \mathbf{x})}=\boldsymbol{\omega}^T\mathbf{x}+b \tag{21}$$

可得到对于输入的向量 x ，其属于正例的概率如式：

$$p(y=1|\ \mathbf{x})=\frac{e^{\mathbf{w}^T\mathbf{x}+b}}{1+e^{\boldsymbol{\omega}^T\mathbf{x}+b}}=\sigma\left(\boldsymbol{\omega}^T\mathbf{x}+b\right) \tag{22}$$

属于反例的概率如式：

$$p(y=0|\ \mathbf{x})=\frac{1}{1+e^{\omega^T\mathbf{x}+b}}=1-\sigma\left(\boldsymbol{\omega}^T\mathbf{x}+b\right) \tag{23}$$

其中未知参数 ω 和 b 可以通过极大似然法来进行估计。给定数据集的逻辑回归模型最大化“对数似然”：

$$p\left(y_i|\ \mathbf{x}_i;\boldsymbol{\omega},b\right)=y_i p_1\left(\mathbf{x}_i;\boldsymbol{\beta}\right)+\left(1-y_i\right)p_0\left(\mathbf{x}_i;\boldsymbol{\beta}\right) \tag{24}$$

最大化似然函数等价于最小化式：

$$\ell(\boldsymbol{\beta})=\sum_{i=1}^{n}\left(-y_i\boldsymbol{\beta}^T\mathbf{x}_i+\ln\left(1+e^{\beta^T\mathbf{x}i}\right)\right) \tag{25}$$

最后通过经典的数值优化算法如牛顿法、梯度下降法等都可以求得其最优解：

$$\boldsymbol{\beta}^*=\arg\min_n\ell(\boldsymbol{\beta}) \tag{26}$$

分类问题建模时，最常用的性能评价指标为错误率与精度。但是在不平衡数据集中，由于多数类的数据量占比较高，其对结果的影响也更大，简单考虑最后样本中预测对了多少数据是非常不合理的。针对传统性能评估标准的缺陷，一些研究者提出可采用混淆矩阵(confusion matrix)、ROC 曲线、AUC 以及 G-mean 等评价指标来评估模型的分类性能。

针对二分类问题，可以将所有数据分为四类：样本为正类被预测为正类(TP)，样本为负类被预测为正类(FP)，样本为正类被预测为负类(FN)，样本为负类被预测为负类(TN)。以矩阵的形式展示出来即可得到混淆矩阵，如表 1 所示。

表 1、混淆矩阵

		预测值	
		正样本	负样本
真实值	正样本	TP (真正例)	FN (假反例)
	负样本	FP (假正例)	TN (真反例)

3、问题求解

问题 1

本文针对 1974 个化合物的 729 个分子描述符通过特征工程方法，基本思路与研究框架如图 2 所示。

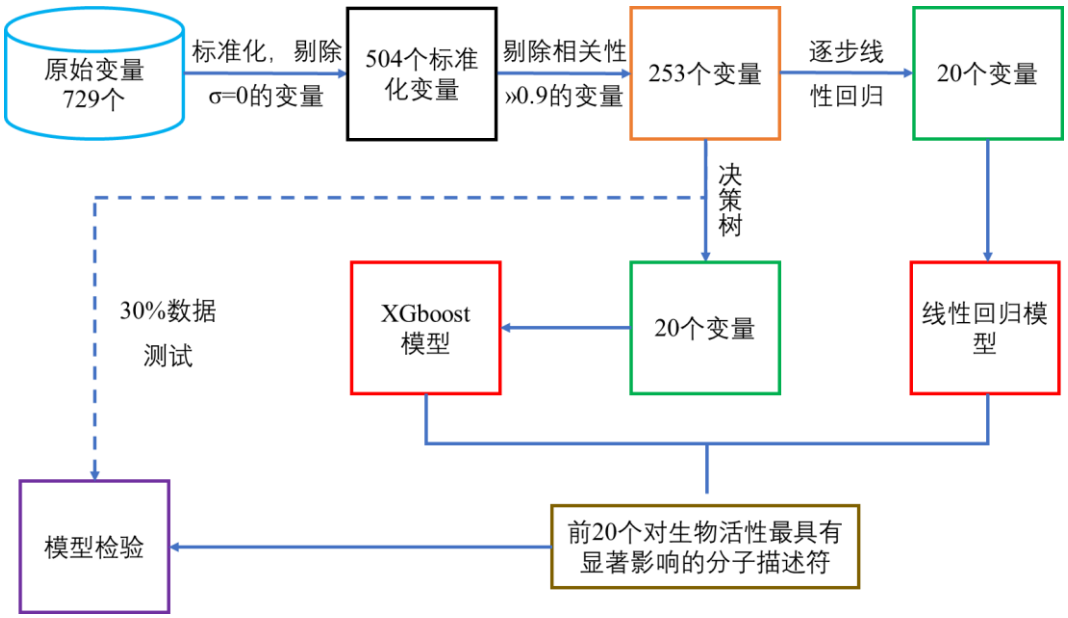


图 2、化合物分子特征提取思维导图

逐步回归：将对被解释变量有影响的变量指标因素逐个的引入方程，每添加一个变量指标后，就要对已选入的原有指标做显著性检验，如果先引入的指标因为新引入的指标而对被解释变量的影响不再显著时，就要剔除先引入的指标变量，在方程中引进新变量或者剔除影响不显著的变量称为逐步回归的一个步骤。在引进新变量之前，为了使回归方程中仅包含对被解释变量有显著影响的指标变量，需要对逐步回归的每一个步骤做 F 检验。这个过程要反复进行，直至没有显著的指标因素再引入方程，也没有不显著的变量指标从方程中剔除为止。

逐步回归法的基本步骤如下：

①对指标数据作标准化处理。

②从所有的指标因素中挑一个使构建的线性方程的回归平方和最大的变量，然后求其 F 统计量，若通过 F 检验，则引入该自变量。

③从剩余的变量中挑选一个变量，使方程加入该变量后具有最大的回归平方和，再对方程进行 F 检验。若通过 F 检验，则引入该变量，否则将不能引入该变量。

④将新变量引进回归方程后，要对原有变量的系数作 F 检验，若原有变量的系数对被解释变量的影响不再显著，则剔除原有变量，反之，则保留原有变量。

⑤重复步骤③和步骤④，直到没有新的变量引进方程，也没有变量被剔除。

最后通过多元线性回归方法计算变量的权系数并排序，结果如表 2 所示：

表 2、分子描述符对 pIC50 值的权系数排序

排序	变量名	权系数	排序	变量名	权系数
1	ECCEN	0.844991362	11	C1SP2	0.151271567
2	MLFER_A	0.393166595	12	minHBint10	0.147416509
3	BCUTp-1h	0.3049971	13	mindsCH	0.145815859
4	maxHBint2	0.277056741	14	maxsssCH	0.137151156
5	nHsOH	0.271660489	15	ETA_Shape_Y	0.13543088
6	nHBAcc_Lipinski	0.26401698	16	minHBd	0.13442896
7	nHBAcc	0.243652058	17	minHBa	0.119867955
8	MDEC-22	0.207499317	18	maxHBint6	0.107422473
9	minHsOH	0.193200154	19	MDEC-23	0.095018625
10	n6Ring	0.188283585	20	minHBint5	0.082672742

Xgboost：清除化合物中方差为零的变量，使得研究变量简化为 505 个。通过对数据预处理完成后的标准化数据排除相关系数高于 0.9 的标准化变量数据。而对于一个变量与其他多个变量相关性系数高于 0.9，则此变量删除，其他变量保存。同时进行 P-value 显著性检测，使得抽样误差所致的概率小于 0.05，使得研究变量简化为 254 个。再采用 Xgboost 对数据与目标函数进行训练，训练得到的分类决策树模型如图 3 所示。提取数据特征分布和 Shap 绝对值取平均值来代表该特征的重要性，分别如图 4、5 所示，最终 Shap 绝对值平均计算结果如表 3 所示。

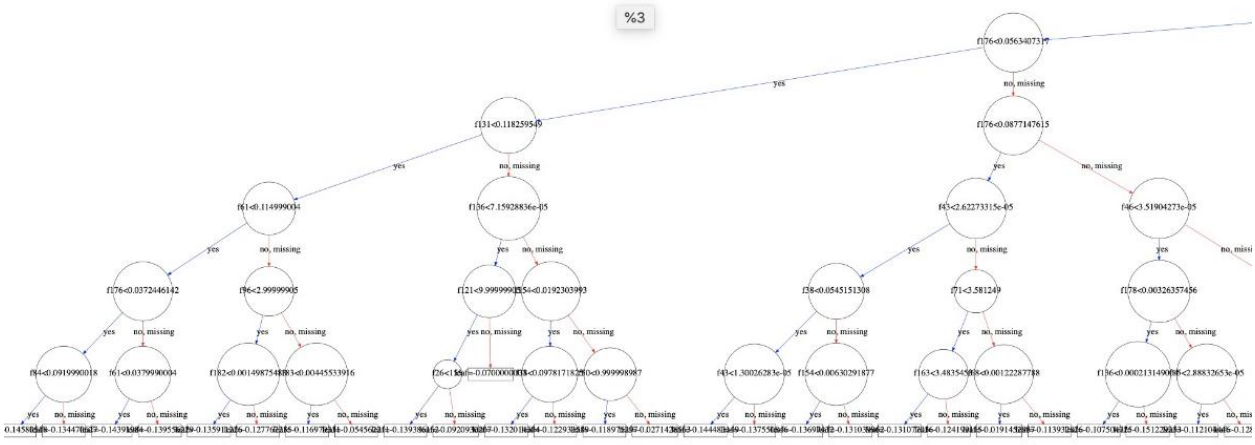


图 3、Xgboost 模型分类树的一部分

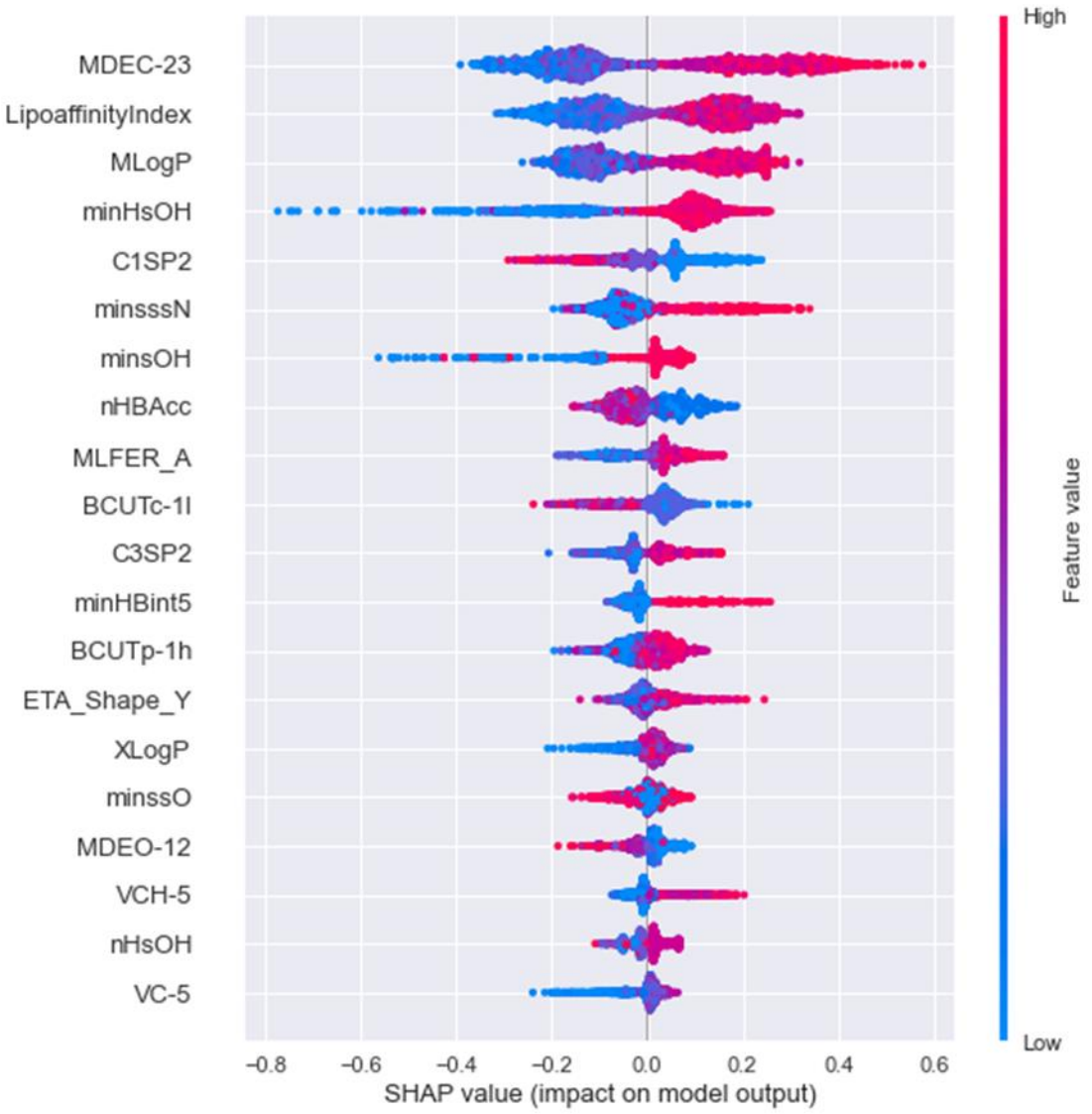


图 4、生物活性影响的特征分布

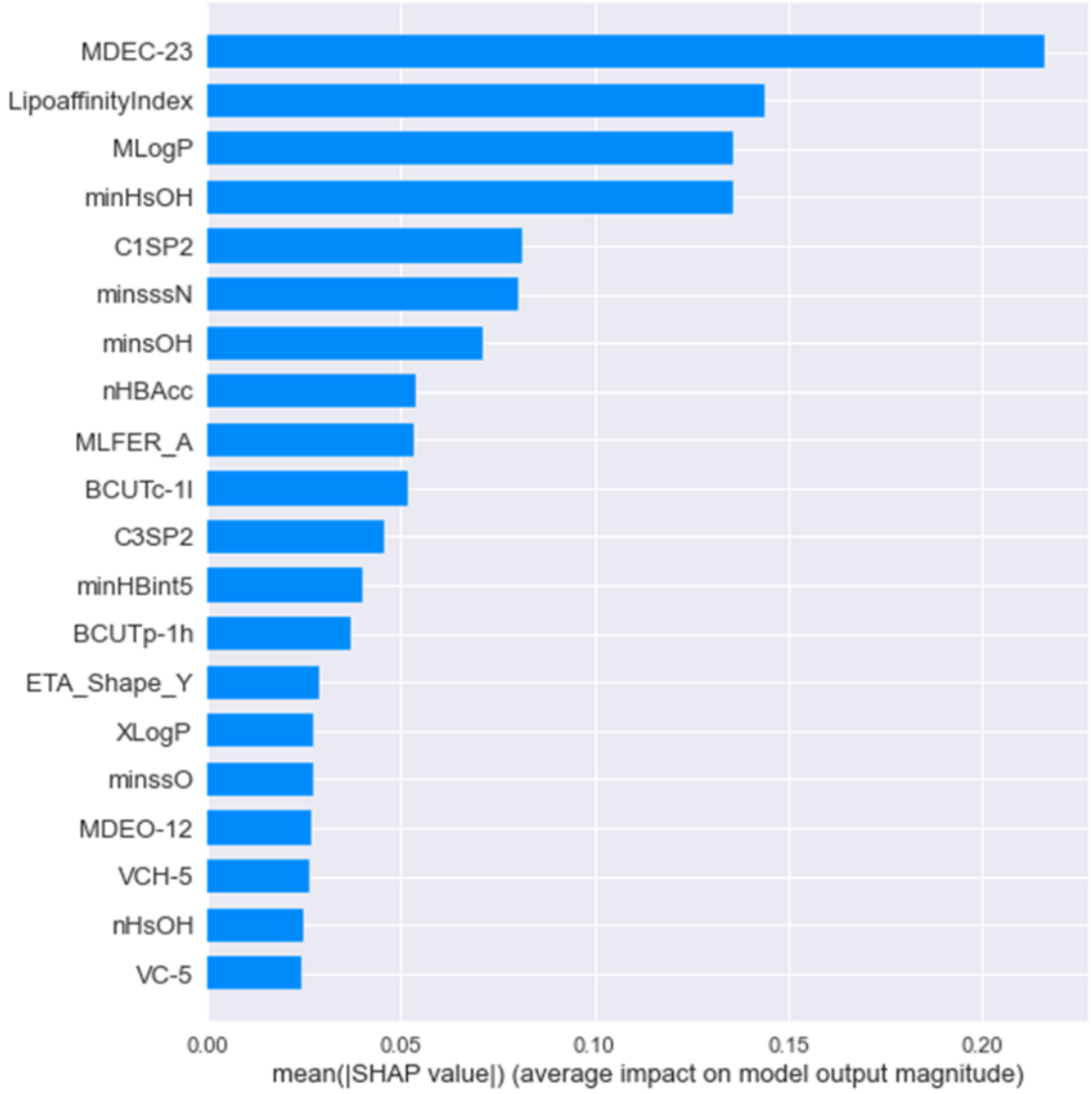


图 5、生物活性影响的 Shap 绝对值取平均值

表 3、分子描述符对 Shap 值的排序

排序	变量名	平均 Shap 值	排序	变量名	平均 Shap 值
1	MDEC-23	0.216354	11	C3SP2	0.045757
2	LipoaffinityIndex	0.14372	12	minHBint5	0.040156
3	MLogP	0.135957	13	BCUTp-1h	0.037007
4	minHsOH	0.135651	14	ETA_Shape_Y	0.028934
5	C1SP2	0.081136	15	XLogP	0.027433
6	minsssN	0.080248	16	minssO	0.027127
7	minsOH	0.071225	17	MDEO-12	0.026955
8	nHBAcc	0.053962	18	VCH-5	0.026505
9	MLFER_A	0.0534	19	nHsOH	0.024669
10	BCUTc-1l	0.051715	20	VC-5	0.024262

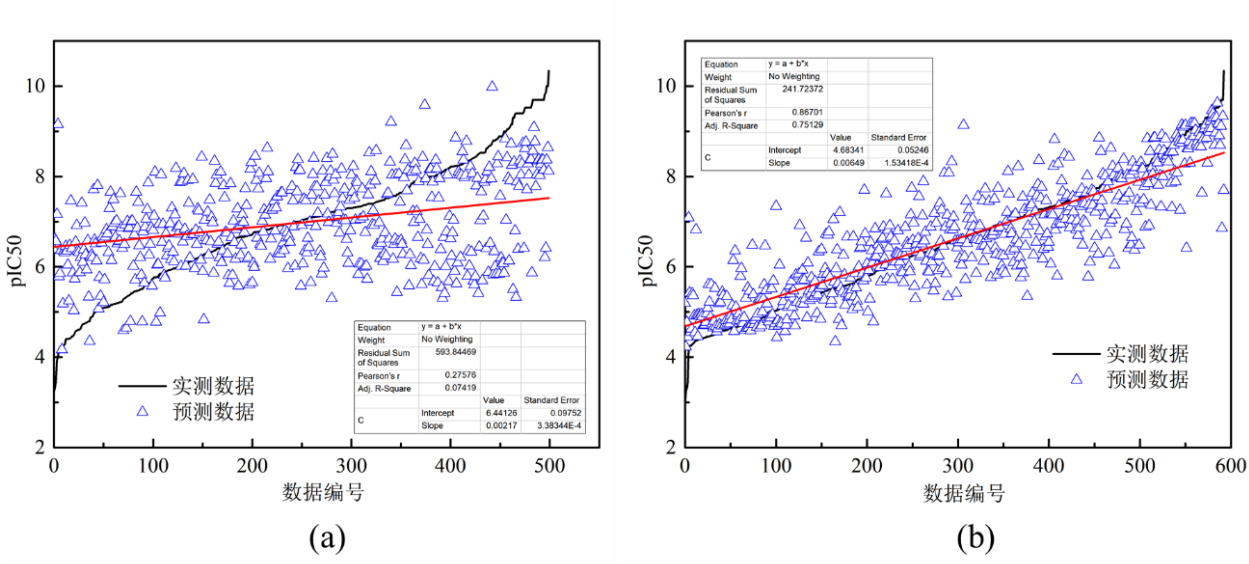


图 6、逐步回归方法和 XGboost 方法预测值与真实值对比

表 2 和表 3 中筛选的变量有九个相同，十一个变量存在差异，为进一步构建最优模型，通过选择 1974 组数据中的 30%数据集进行学习预测(如图 5 所示)，相较于逐步回归方法，基于决策树的 XGboost 表现出更好的拟合结果，因此，前 20 个对生物活性最具有显著影响的分子描述符(即变量)采用 XGboost 方法筛选。

问题 2

根据 ERα_activity.xlsx 测试集中的 IC50 值和对应的 pIC50 值数据通过 matlab 拟合如图 7 所示，得到 IC50=10^(9- pIC50)。

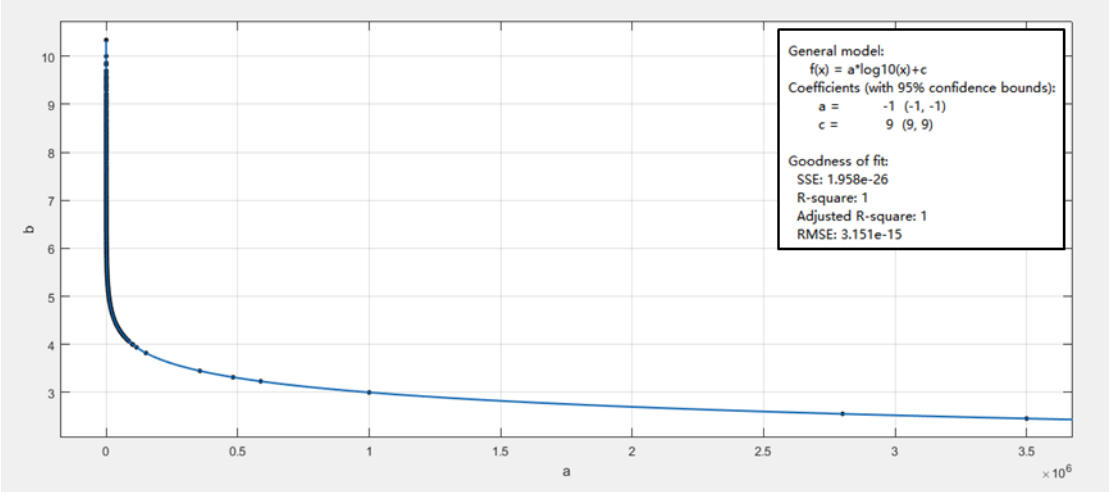


图 7、IC50 值和 pIC50 值曲线拟合

选取 XGboost 最终的 20 个特征值随机抽取 70%数据为训练集，30%为测试集，残差值 0.725，模型准确度 91.24%。如图 8 所示，Xgboost 模型对测试数据的预测值与实验实测值非常接近。

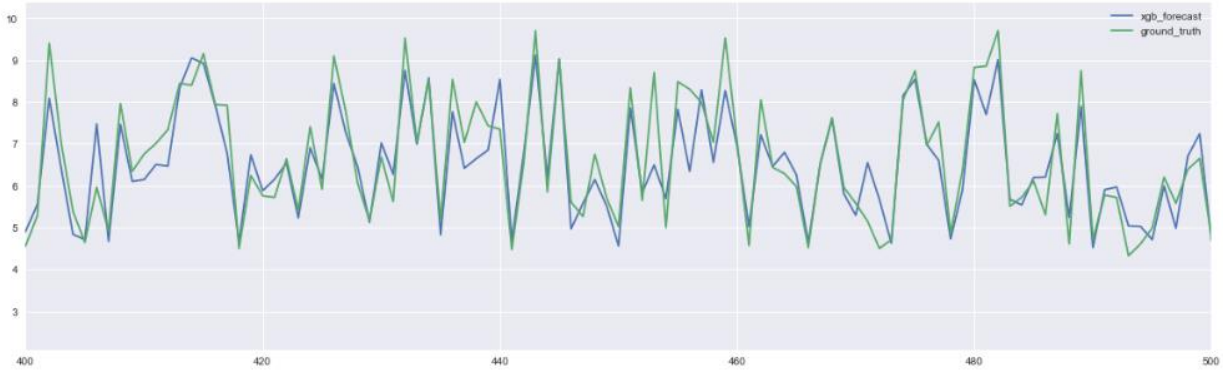


图 8、Xgboost 预测值与实验实测值

基于预测 test 表格中的 50 组数据，结果如表 4 所示。

表 4、50 个化合物的 IC50 和 pIC50 预测值

序号	IC50 预测值	pIC50 预测值	序号	IC50 预测值	pIC50 预测值	序号	IC50 预测值		pIC50 预测值
1	325.3752	6.487616	18	824.4456	6.083838	35	7007.113		5.154461
2	170.5634	6.768114	19	124.5065	6.904808	36	11177.92		4.951639
3	829.8602	6.080995	20	136.4395	6.86506	37	14882.72		4.827318
4	223.7408	6.650255	21	426.331	6.370253	38	2227.499		5.652183
5	223.4904	6.650741	22	550.7958	6.259009	39	384.0277		6.415637
6	169.3721	6.771158	23	188.2196	6.725335	40	339.1566		6.4696
7	86.57057	7.06263	24	135.3623	6.868502	41	339.1566		6.4696
8	159.035	6.798507	25	1372.248	5.862567	42	339.1566		6.4696
9	145.1661	6.838135	26	454.9886	6.342	43	339.1566		6.4696
10	77.99889	7.107912	27	389.1504	6.409883	44	332.0952		6.478737
11	72.6445	7.138797	28	836.2738	6.077652	45	339.1566		6.4696
12	67.98591	7.167581	29	845.3996	6.072938	46	69.84983		7.155835
13	25.67194	7.590541	30	1151.666	5.938673	47	116.1805		6.934867
14	68.04154	7.167226	31	3010.497	5.521362	48	208.0075		6.681921
15	140.3345	6.852836	32	3625.53	5.440629	49	1477.408		5.8305
16	136.5596	6.864678	33	7075.019	5.150272	50	34.213		7.465809
17	144.4784	6.840197	34	2882.098	5.540291				

问题 3

通过 Xgboost 和逻辑回归分别构建化合物的 Caco-2、CYP3A4、hERG、HOB、MN 的分类预测模型。针对二分类问题，可以将所有数据分为四类：样本为正类被预测为正类(TP)，样本为负类被预测为正类(FP)，样本为正类被预测为负类(FN)，样本为负类被预测为负类(TN)，如图 9 所示。采用树的结构进行打分，分数越小效果就越好。由于不可能列举出所有可能的树结构进行打分，这里可以利用分割后的左L和右R节点的实例集的得分对分割候

选节点进行评估，分类结果如表 5 所示。对两种方法的准确率(如图 10 所示)计算表明，Xgboost 相较于逻辑回归模型准确率更高。

表 5、ADMET 分类预测结果

序号/指标	Caco-2	CYP3A4	hERG	HOB	MN
1	0	1	1	0	1
1	0	1	1	0	1
2	1	0	0	0	1
3	1	1	0	0	1
4	1	0	0	0	1
5	1	0	0	0	1
6	1	0	0	0	1
7	0	0	0	0	1
8	0	0	0	1	1
9	1	0	0	0	1
10	1	1	1	0	1
11	1	1	0	0	1
12	1	1	1	0	1
13	1	1	1	1	1
14	1	1	1	0	1
15	0	1	1	0	1
16	0	1	1	0	1
17	0	1	0	0	1
18	1	0	0	1	1
19	1	0	0	1	1
20	1	0	0	0	1
21	1	1	0	0	1
22	1	1	0	0	1
23	1	0	0	0	0
24	1	0	0	0	0
25	1	1	1	1	0
26	1	1	1	1	1
27	1	1	1	0	0
28	1	1	1	0	1
29	1	1	1	1	1
30	1	1	1	0	1
31	1	1	1	1	1
32	1	1	1	1	1
33	1	1	1	1	1
34	1	1	1	1	1
35	1	1	1	1	1
36	0	1	1	0	1
37	0	1	1	1	1

38	0	1	1	0	0
39	0	1	0	0	1
40	0	1	0	0	1
41	0	1	0	0	1
42	0	1	0	0	1
43	0	1	0	0	1
44	0	1	0	0	1
45	0	1	0	0	1
46	1	1	1	0	1
47	1	1	1	0	1
48	1	1	1	0	1
49	1	1	1	0	1
50	1	0	1	0	0

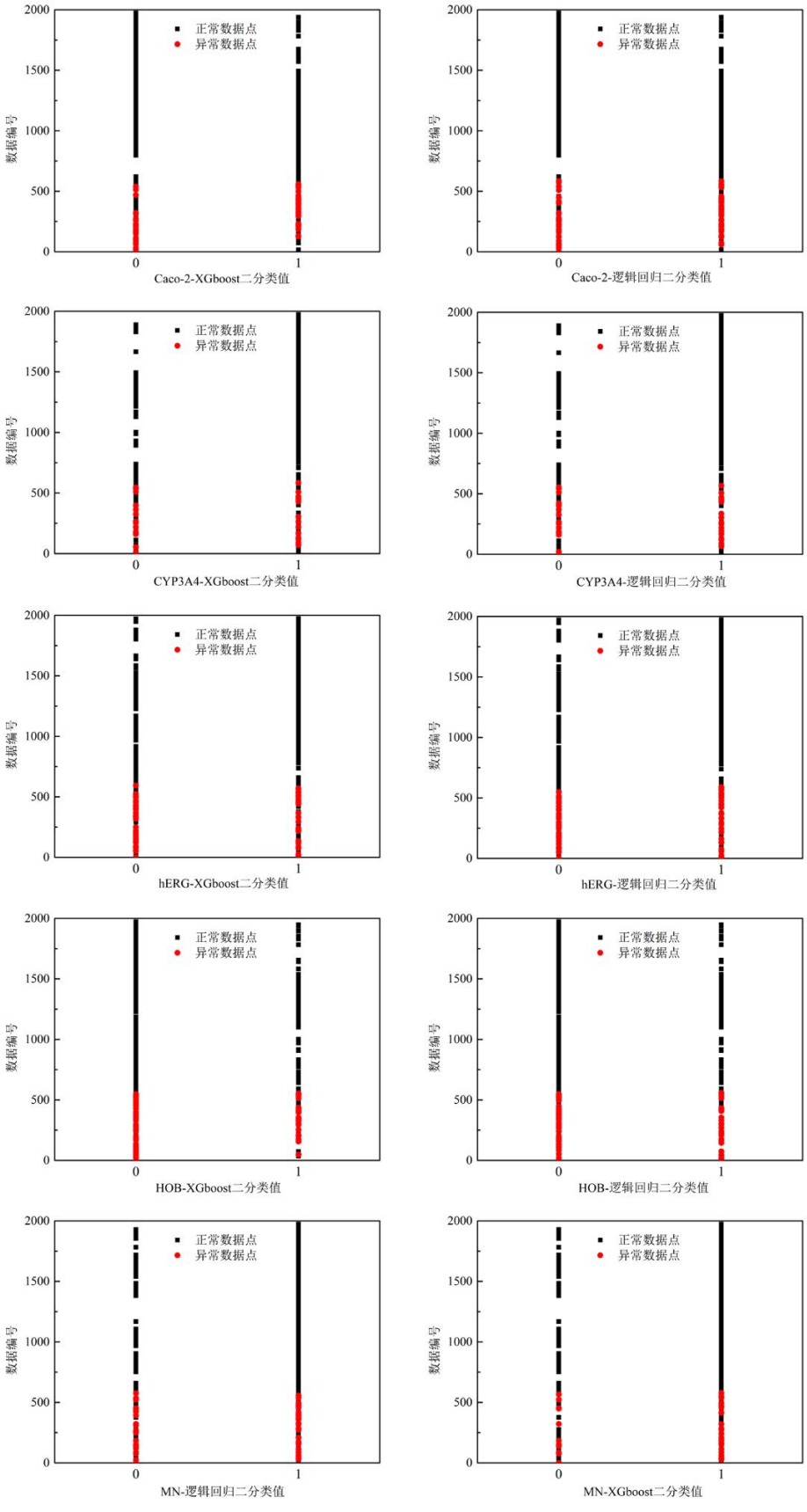


图 9、XGboost 与逻辑回归方法二分类对比

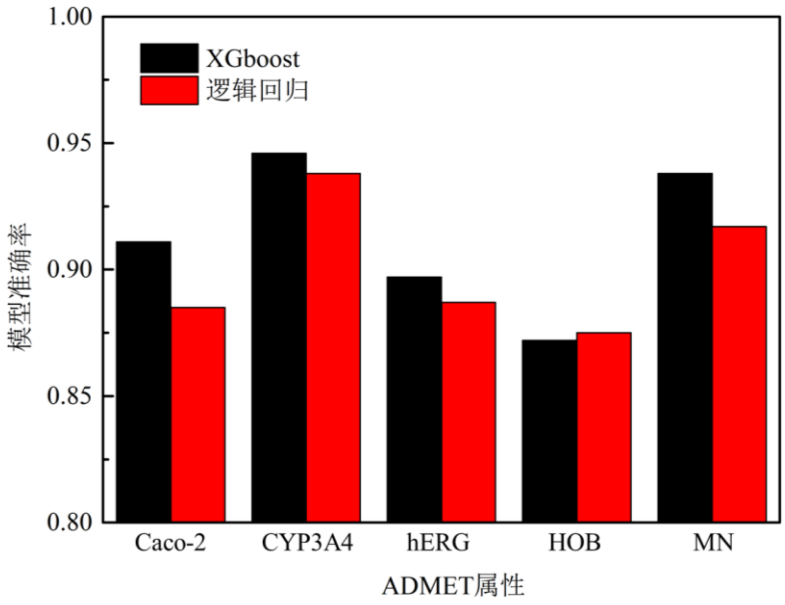
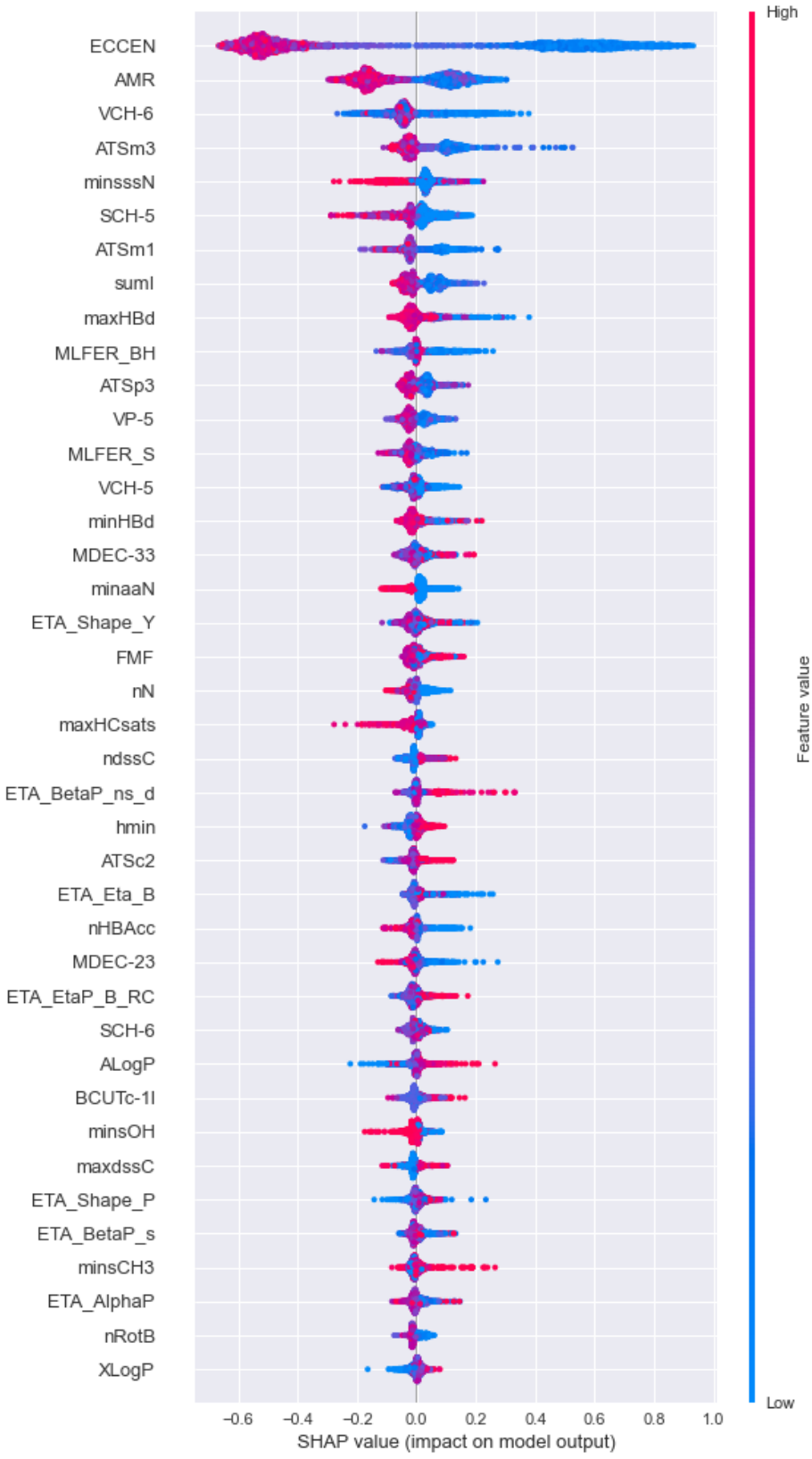


图 10、XGboost 与逻辑回归方法准确率对比

问题 4

针对化合物的综合 ADMET 属性，将其综合属性归纳为 1,2,3,4,5 类，对数据筛选后，分别采用 Xgboost 方法对 ERα 的 pIC50 值与综合 ADMET 属性进行分类决策训练。通过比较两个训练模型的前 40 个最大 Shap 绝对值，针对 ADMET 为目标函数的 Xgbboost 模型 Shap 值如图 11 与图 12 所示。从两个分子描述符集合中挑选出 15 个有交集的分子描述符。对 15 个分子描述符进行线性回归拟合，分别得到 pIC50 值与综合 ADMET 属性的数学模型；采用线性规划求出满足 ADMET 属性>=3 与 pIC50 值最大时的分子描述符取值范围或最值，即为分子描述符的最优取值范围或最值。筛选分子描述符的最优取值范围或最值结果如表 6 所示。此时 pIC50 值的变化区间为 [11.74, 12.90]，ADMET 综合属性区间为[3，



5]。

图 11、以 ADMET 多分类为目标函数的 SHAP 值排序

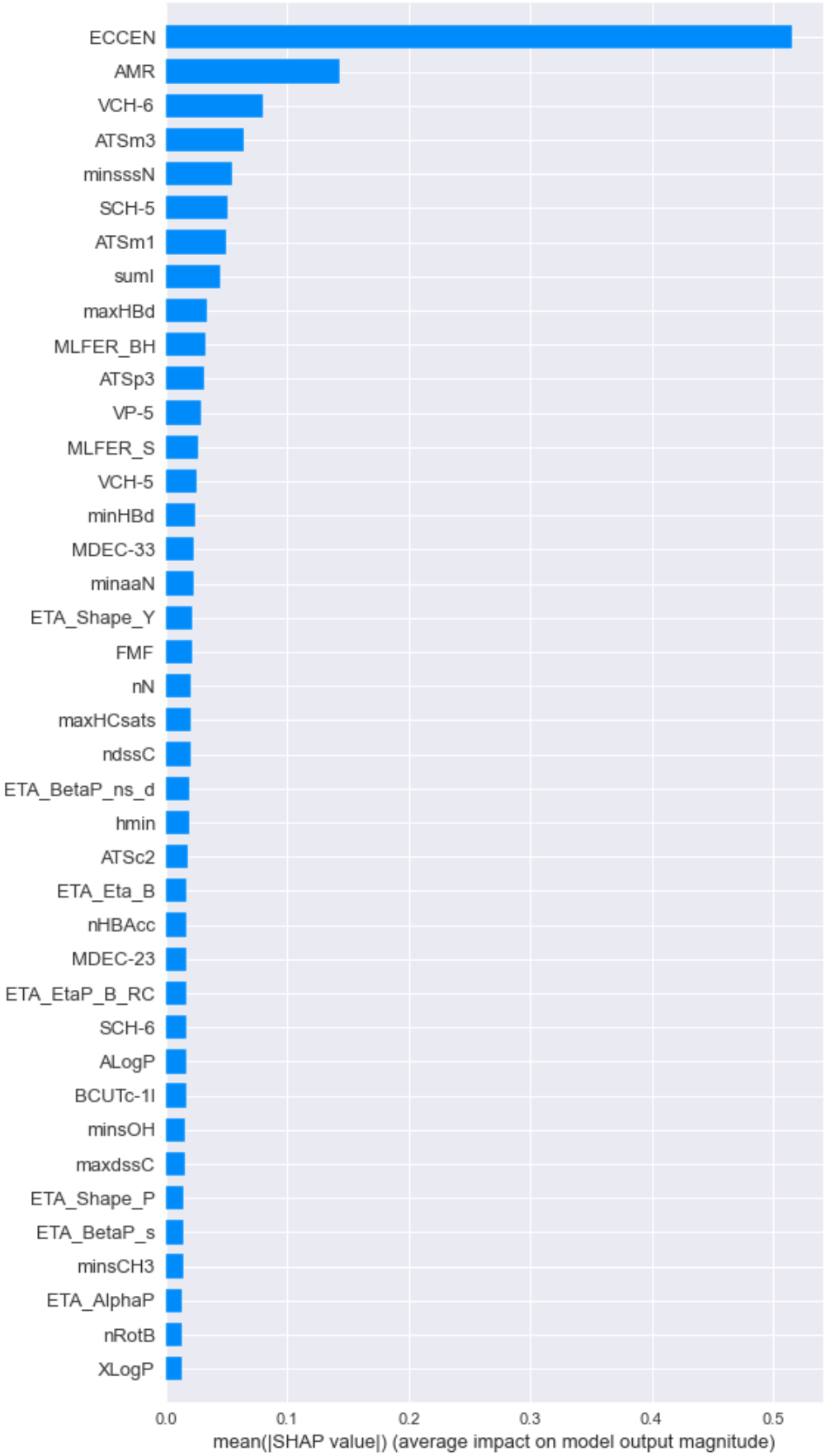


图 11、以 ADMET 多分类为目标函数的 SHAP 值绝对值排序

表 6、筛选分子描述符的最优取值范围与最值

分子描述符	取值范围
ECCEN	2090
AMR	60.829 ~ 151.125
minsssN	2.735
VCH-5	0.174
minHBd	0.763
MDEC-33	-0.002
ETA_Shape_Y	0.512
nHBAcc	0
MDEC-23	50.500
ALogP	-2.701
BCUTc-1l	-0.273
minsOH	11.731
ETA_BetaP_s	0.548
ETA_AlphaP	0.523
XLogP	-1.038 ~ 7.149

4、参考文献

[1] Benson C S, Babu S D, Radhakrishna S, et al. Expression of matrix metalloproteinases in human breast cancer tissues[J]. Disease markers, 2013, 34(6): 395-405.

[2] DeSantis C E, Ma J, Goding Sauer A, et al. Breast cancer statistics, 2017, racial disparity in mortality by state[J]. CA: a cancer journal for clinicians, 2017, 67(6): 439-448.

[3] 陈文瑄. QSPR 方法在有机污染物 PDMS/环境介质分配系数中的应用研究[D].扬州大学, 2021.

[4]李仁利.定量构-效关系(QSAR)的应用及限度[J].国外医学.药学分册, 1983(02): 65-72.

[5] 成水平, 周永欣. 有机化合物结构与活性定量关系在水生毒理学上的应用[J].生态科学, 1992(02): 36-40.

[6] Pratim Roy P, Paul S, Mitra I, et al. On two novel parameters for validation of predictive QSAR models[J]. Molecules, 2009, 14(5): 1660-1701.

[7] Chen B, Zhang T, Bond T, et al. Development of quantitative structure activity relationship (QSAR) model for disinfection byproduct (DBP) research: A review of methods and resources[J]. Journal of hazardous materials, 2015, 299: 260-279.

[8] Frid A A, Matthews E J. Prediction of drug-related cardiac adverse effects in humans-B: Use of QSAR programs for early detection of drug-induced cardiac toxicities[J]. Regulatory toxicology and pharmacology, 2010, 56(3): 276-289.

[9] Kireeva N, Kuznetsov S L, Bykov A A, et al. Towards in silico identification of the human ether-a-go-go-related gene channel blockers: discriminative vs. generative classification models[J]. SAR and QSAR in Environmental Research, 2013, 24(2): 103-117.

[10] Wang S, Sun H, Liu H, et al. ADMET evaluation in drug discovery. 16. Predicting hERG blockers by combining multiple pharmacophores and machine learning approaches[J]. Molecular pharmaceutics, 2016, 13(8): 2855-2866.

[11]Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]. Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

[12]Zhang D, Qian L, Mao B, et al. A data-driven design for fault detection of wind turbines using random forests and XGboost[J]. Ieee Access, 2018, 6: 21020-21031.

[13]Ren X, Guo H, Li S, et al. A novel image classification method with CNN-XGBoost model[C]. International Workshop on Digital Watermarking. Springer, Cham, 2017: 378-390.

[14]Torlay L, Perrone-Bertolotti M, Thomas E, et al. Machine learning–XGBoost analysis of language networks to classify patients with epilepsy[J]. Brain informatics, 2017, 4(3): 159-169.

[15]Ogunleye A, Wang Q G. XGBoost model for chronic kidney disease diagnosis[J]. IEEE/ACM transactions on computational biology and bioinformatics, 2019, 17(6): 2131-2140.

[16]王文博, 曾小梅, 赵引川, 张云云, 刘达. 基于 SMOTE-XGBoost 的变压器缺陷预测[J]. 华北电力大学学报(自然科学版), 2021, 48(05):54-60+71.

[17]Thongsuwan S, Jaiyen S, Padcharoen A, et al. ConvXGB: A new deep learning model for classification problems based on CNN and XGBoost[J]. Nuclear Engineering and Technology, 2021, 53(2): 522-531.

[18]Lundberg S M, Lee S I. A unified approach to interpreting model predictions[C]. Proceedings of the 31st international conference on neural information processing systems. 2017: 4768-4777.

Q1

In [2221]:
```
import pandas as pd

path1 = r'D:\documents\my work\py\D\Molecular_Descriptor.xlsx'
path2 = r'D:\documents\my work\py\D\ERα_activity.xlsx'
Molecular = pd.read_excel(path1, sheet_name = 'training')
Activity = pd.read_excel(path2, sheet_name = 'training')
Molecular_SMILES = Molecular['SMILES']
```

In [2222]:
```
from scipy import stats


def cor_clean(df, threshold):

    df1 = df.copy()
    des3 = len(df.columns)
    df1 = df1.iloc[:,1:]

    cor_i = set()
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = df1.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if j not in cor_i and corr_matrix.iloc[i, j] >= threshold and stats.pearsonr(df1.iloc[:,i].values,df1.iloc[:,
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
                cor_i.add(i)
    for colname in col_corr:
        del df1[colname] # deleting the column from the dataset

    df1.insert(0, 'SMILES', df['SMILES'])

    des4 = len(df1.columns)

    print ('from Remove correlation')
    print ("The initial set of " + str(des3) + ' descriptors'+
           " has been reduced to " + str(des4) + " descriptors.")

    return df1
```

In [2223]:
```
from sklearn import preprocessing
def std_clean(df, threshold):

    df1 = df.copy()
    des3 = len(df.columns)


    df1 = df1.iloc[:,1:]
    min_max_scaler = preprocessing.MinMaxScaler()
    np_scaled = min_max_scaler.fit_transform(df1)
    df_normalized = pd.DataFrame(np_scaled, columns=df1.columns)
    for i in df_normalized.columns:
            if df_normalized[i].std() <= threshold:
                del df1[i] # deleting the column from the dataset
    df1.insert(0, 'SMILES', df['SMILES'])
    des4 = len(df1.columns)

    print ('from Remove std')
    print ("The initial set of " + str(des3) + ' descriptors'+
           " has been reduced to " + str(des4) + " descriptors.")

    return df1
```

In [2224]:
```
def normalized(Fp):
    Fp_normalized = Fp.iloc[:,1:]
    Fp_normalized = (Fp_normalized - Fp_normalized.mean()) / Fp_normalized.std()
    Fp_normalized.insert(0, 'SMILES', Fp['SMILES'])
    return Fp_normalized
```

In [2225]:
```
from scipy import stats

def pvalue_clean(df,y,threshold):

    df_p = df.copy()
    des3 = len(df_p.columns)
    df_p = df_p.iloc[:,1:]
    colname = []
    cor_value = []
    df_cor = pd.DataFrame(columns = ['colname', 'cor_value'])


    for i in df_p.columns:
        corr = stats.pearsonr(df_p[i].values,y)
        if corr[1] >= threshold:
            del df_p[i]
        else:
            colname.append(i)
            cor_value.append(corr[0])

    df_cor['colname'] = colname
    df_cor['cor_value'] = cor_value

    df_p.insert(0, 'SMILES', df['SMILES'])
    des4 = len(df_p.columns)

    print ('from Remove correlation')
    print ("The initial set of " + str(des3) + ' descriptors'+
           " has been reduced to " + str(des4) + " descriptors.")

    return df_p
```

In [2226]:
```
def accuracy(y_test, y_pred):

    y= []
    for i in range(len(y1_test)):
        err = abs(y_test[i]-y_pred[i])/y_test[i]
        y.append(err)
    err_mean = statistics.mean(y)
    acc = 1 - err_mean
    return acc
```

In [2227]:
```
Molecular_s = std_clean(Molecular,0)
Molecular_n = normalized (Molecular_s)
# Molecular_p = pvalue_clean(Molecular_n,Activity['pIC50'],0.05)
Molecular_c = cor_clean(Molecular_n,0.9)
colnames_clean = Molecular_c.iloc[:,1:].columns.values
```

```
from Remove std
The initial set of 730 descriptors has been reduced to 505 descriptors.
from Remove correlation
The initial set of 505 descriptors has been reduced to 254 descriptors.
```

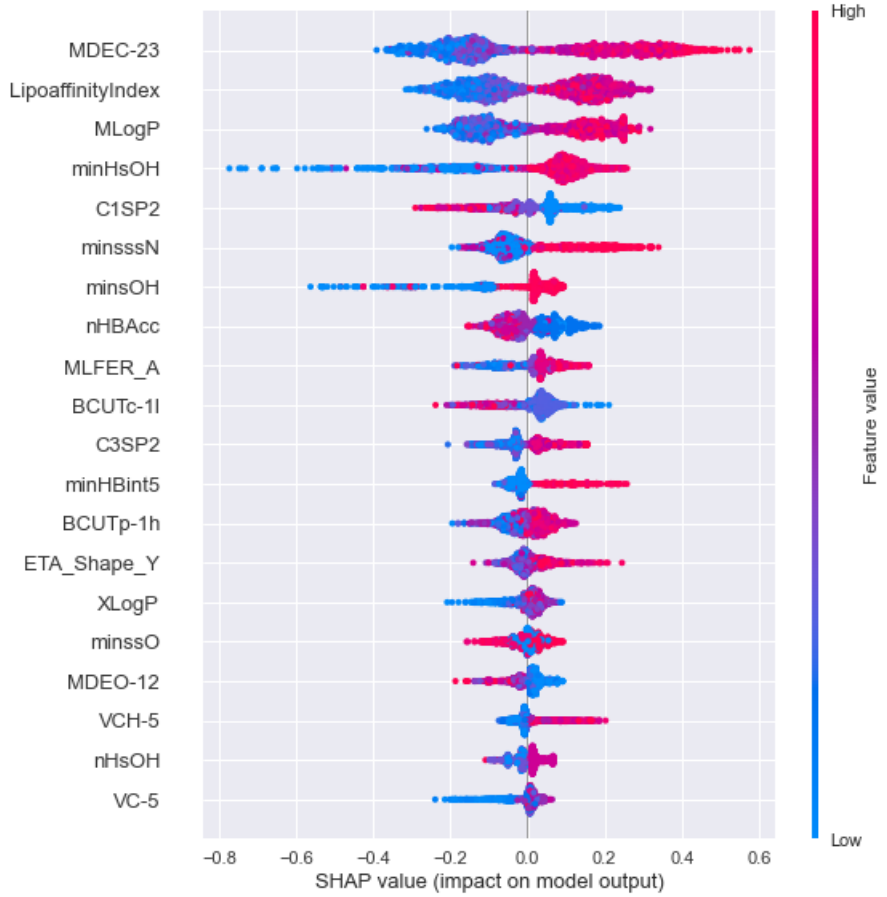
In [2228]:
```
import xgboost as xgb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt; plt.style.use('seaborn')
```

In [2229]:
```
data = Molecular_c.iloc[:,1:]
other_params = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 8, 'min_child_weight': 5, 'seed': 0,
                'subsample': 0.6, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0, 'reg_lambda': 0.5}
model = xgb.XGBRegressor(**other_params)
model.fit(data, Activity['pIC50'].values)
cols = data.columns.values
```

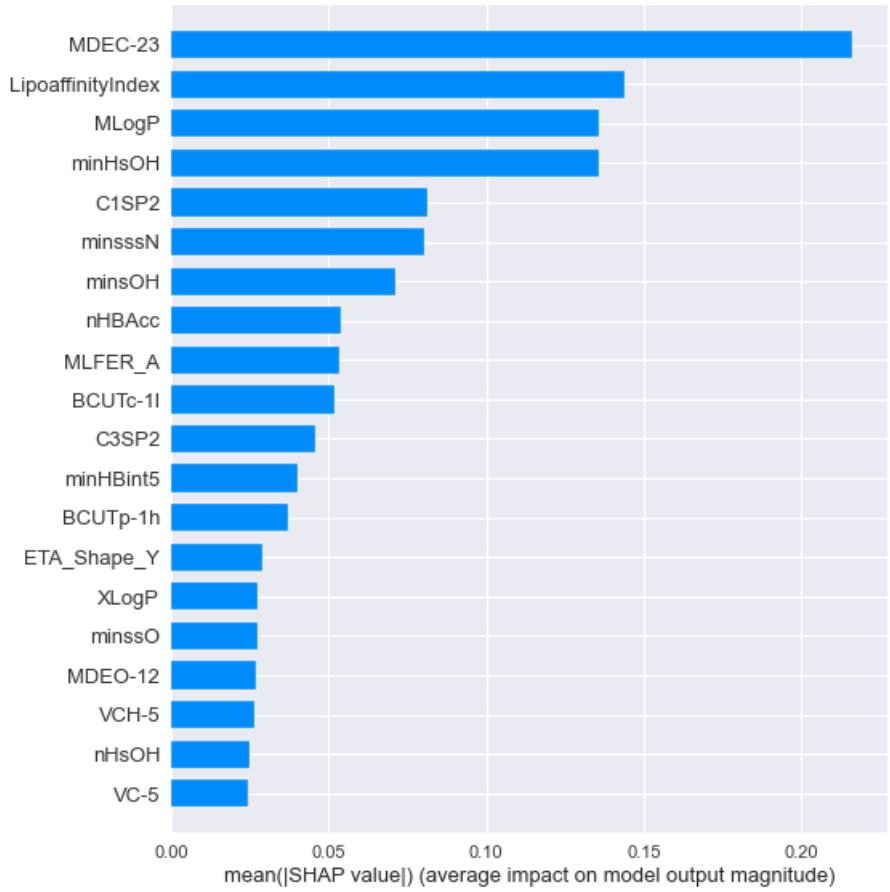
In [2230]:
```
import shap

# model是在第1节中训练的模型
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(data[cols])
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

In [2231]:
```
shap.summary_plot(shap_values, data[cols],max_display =20)
```



In [2232]:
```
shap.summary_plot(shap_values, data[cols], plot_type="bar",max_display =20)
```

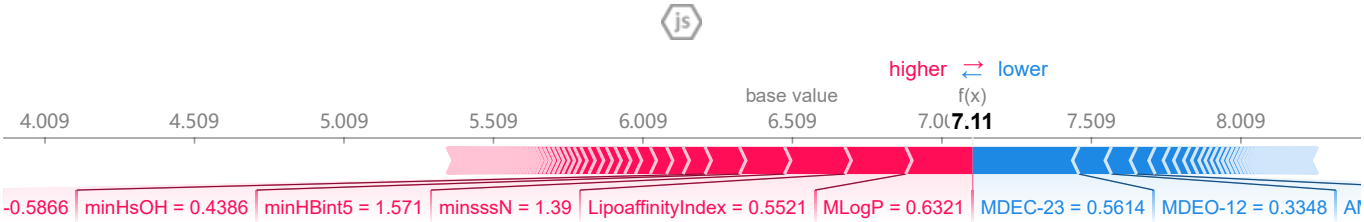


In [2233]:
```
mean = np.mean(np.absolute(shap_values), axis=0)

colname1 = Molecular_c.iloc[:,1:].columns.values

shap_mean = pd.DataFrame(columns= ['name','shap'])
shap_mean['name']=colname1
shap_mean['shap']=mean
shap_mean.sort_values( by = 'shap',ascending = False)
shap_mean.to_excel((r'D:\documents\my work\py\D\shap.xlsx'))
sh = shap_mean.sort_values( by = 'shap',ascending = False)
sh.head(20)
colname_select = sh.head(40)['name'].values
print(colname_select)
# sharp_list = mean.tolist()
# sharp_list.sort(reverse=False)
# print(sharp_list.sort(reverse=False))
```

```
['MDEC-23' 'LipoaffinityIndex' 'MLogP' 'minHsOH' 'C1SP2' 'minsssN'
 'minsOH' 'nHBAcc' 'MLFER_A' 'BCUTc-1l' 'C3SP2' 'minHBint5' 'BCUTp-1h'
 'ETA_Shape_Y' 'XLogP' 'minssO' 'MDEO-12' 'VCH-5' 'nHsOH' 'VC-5' 'AMR'
 'MDEC-33' 'minHBint10' 'maxaasC' 'ECCEN' 'ALogP' 'minHssNH' 'maxHBint10'
 'minHBd' 'minsssCH' 'nBase' 'VC-3' 'ETA_AlphaP' 'maxHBa' 'SHBint10'
 'ATSc4' 'mindssC' 'BCUTp-1l' 'nH' 'ETA_BetaP_s']
```

In [2234]:
```
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[7], data[cols].iloc[j])
```

Out[2234]:



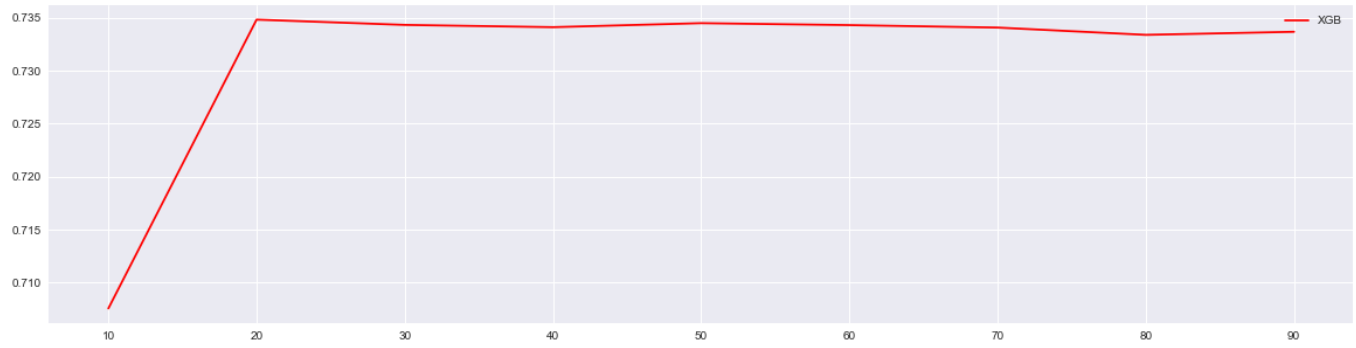
In [2235]:
```
# df_sh = pd.DataFrame(data =shap_values, columns = cols).mean()
# top20 = df_sh.sort_values(ascending = False).head(20)
colname_sh = ['MDEC-23','LipoaffinityIndex','MLogP','minHsOH','C1SP2','minsssN','minsOH','nHBAcc','MLFER_A','BCUTc-1l','C3SP
,'BCUTp-1h','ETA_Shape_Y','XLogP','minssO','MDEO-12','VCH-5','nHsOH','VC-5']

Molecular_20 = Molecular[colname_sh]
Molecular_20.insert(0, 'SMILES', Molecular_SMILES)

Molecular_20.to_excel(r'D:\documents\my work\py\D\Molecular_20.xlsx')
```

In [2236]:
```
from xgboost import XGBRegressor as XGBR
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.linear_model import LinearRegression as LinearR
from sklearn.datasets import load_boston
from sklearn.model_selection import KFold, cross_val_score as CVS, train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from time import time
import datetime

x = data
y = Activity['pIC50']


if __name__ == '__main__':
    x_train,x_test,y_train,y_test = TTS(x,y,test_size=0.1,random_state=420)

    cv_params = {'learning_rate': [0.15, 0.17, 0.2, 0.23, 0.25]}
    other_params = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 8, 'min_child_weight': 5, 'seed': 0,
                    'subsample': 0.6, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0, 'reg_lambda': 0.05}

    model = xgb.XGBRegressor(**other_params)
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='r2', cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(x_train, y_train)
    evalute_result = optimized_GBM.cv_results_
    print('每轮迭代运行结果:{0}'.format(evalute_result))
    print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
    print('最佳模型得分:{0}'.format(optimized_GBM.best_score_))
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  25 out of  25 | elapsed:    5.0s finished

每轮迭代运行结果:{'mean_fit_time': array([0.34308085, 0.33969169, 0.3400908 , 0.3213407 , 0.29022341]), 'std_fit_time': a
rray([0.00591715, 0.00607013, 0.01418908, 0.00998115, 0.06898212]), 'mean_score_time': array([0.00618358, 0.00698152, 0.0
0658216, 0.00738053, 0.00618377]), 'std_score_time': array([0.00039902, 0.00141057, 0.00048844, 0.00101692, 0.00097711]),
'param_learning_rate': masked_array(data=[0.15, 0.17, 0.2, 0.23, 0.25],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'learning_rate': 0.15}, {'learning_rate': 0.17}, {'learning_rate': 0.2}, {'learnin
g_rate': 0.23}, {'learning_rate': 0.25}], 'split0_test_score': array([0.76449697, 0.77120739, 0.78684446, 0.78212075, 0.7
7580524]), 'split1_test_score': array([0.70052761, 0.71316788, 0.72677226, 0.70818953, 0.72468603]), 'split2_test_score':
array([0.71632   , 0.73982938, 0.74017219, 0.72962051, 0.74061209]), 'split3_test_score': array([0.73341669, 0.73028999,
0.72835011, 0.73592018, 0.74052085]), 'split4_test_score': array([0.69049752, 0.71943546, 0.72027697, 0.72348537, 0.70355
279]), 'mean_test_score': array([0.72105176, 0.73478602, 0.7404832 , 0.73586727, 0.7370354 ]), 'std_test_score': array
([0.0261441 , 0.02037228, 0.02405393, 0.02489088, 0.02367122]), 'rank_test_score': array([5, 4, 1, 3, 2])}
参数的最佳取值：{'learning_rate': 0.2}
最佳模型得分:0.7404831986192637
```

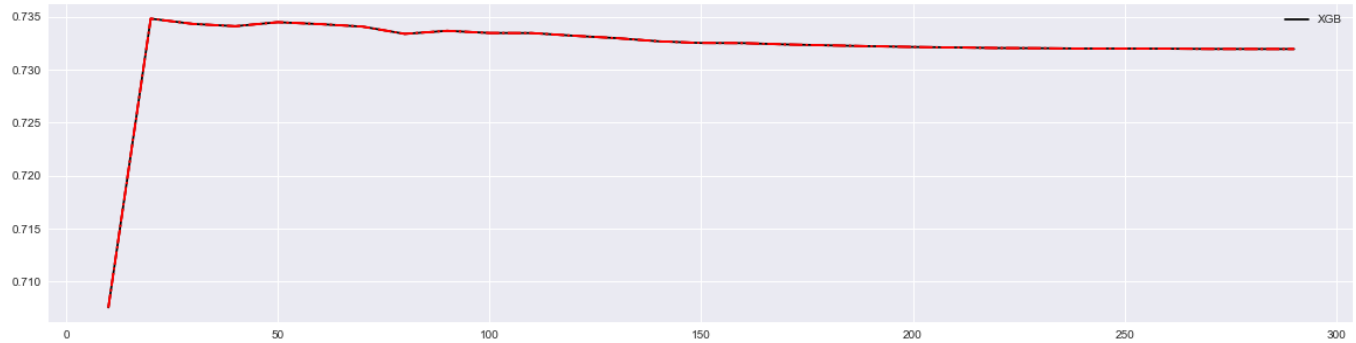
In [2237]:
```
axisx = range(10,100,10)
rs = []
for i in axisx:
    reg = XGBR(n_estimators=i,random_state=420)
    rs.append(CVS(reg,x_train,y_train,cv=5).mean())
print(axisx[rs.index(max(rs))],max(rs))
plt.figure(figsize=(20,5))
plt.plot(axisx,rs,c="red",label="XGB")
plt.legend()
plt.show()
```

```
20 0.7348130680530099
```



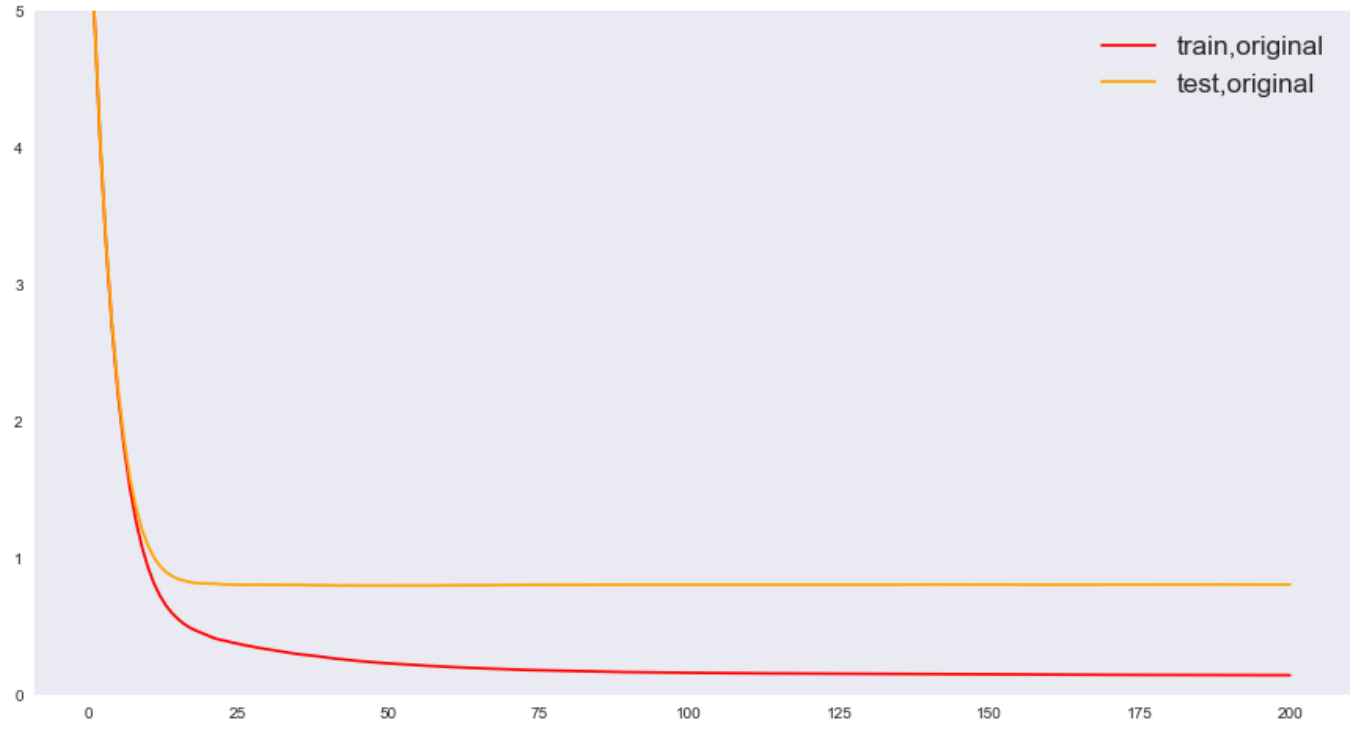
In [2238]:
```
axisx = range(10,300,10)
rs = []
var = []
ge = []
for i in axisx:
    reg = XGBR(n_estimators=i,random_state=420)
    cvresult = CVS(reg,x_train,y_train,cv=5)
    rs.append(cvresult.mean())
    var.append(cvresult.var())
    ge.append((1 - cvresult.mean())**2+cvresult.var())
print(axisx[rs.index(max(rs))],max(rs),var[rs.index(max(rs))])
print(axisx[var.index(min(var))],rs[var.index(min(var))],min(var))
print(axisx[ge.index(min(ge))],rs[ge.index(min(ge))],var[ge.index(min(ge))],min(ge))
rs = np.array(rs)
var = np.array(var)*0.01
plt.figure(figsize=(20,5))
plt.plot(axisx,rs,c="black",label="XGB")
#添加方差线
plt.plot(axisx,rs+var,c="red",linestyle='-.')
plt.plot(axisx,rs-var,c="red",linestyle='-.')
plt.legend()
plt.show()
```

```
20 0.7348130680530099 0.00016429141128273698
10 0.7075474746795959 4.68020083291317e-05
20 0.7348130680530099 0.00016429141128273698 0.07048840028674028
```



In [2239]:
```
dfull1 = xgb.DMatrix(x_test, y_test)

param1 = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 8, 'min_child_weight': 5, 'seed': 0,
                  'subsample': 0.6, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0, 'reg_lambda': 0.5}
num_round = 200

cvresult1 = xgb.cv(param1, dfull, num_round)

fig,ax = plt.subplots(1,figsize=(15,8))
ax.set_ylim(top=5)
ax.grid()
ax.plot(range(1,201),cvresult1.iloc[:,0],c="red",label="train,original")
ax.plot(range(1,201),cvresult1.iloc[:,2],c="orange",label="test,original")
ax.legend(fontsize="xx-large")
plt.show()
```

```
[07:10:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "n_estimators" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[07:10:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "n_estimators" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[07:10:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "n_estimators" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```



In [2240]:
```
from math import sqrt
from sklearn.metrics import mean_squared_error, r2_score
x = data
y = Activity['pIC50']
x_train,x_test,y_train,y_test = TTS(x,y,test_size=0.3,random_state=420)
other_params0 = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 8, 'min_child_weight': 5, 'seed': 0,
                 'subsample': 0.6, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0, 'reg_lambda': 0.05}
model0 = xgb.XGBRegressor(learning_rate= 0.15, n_estimators=40, max_depth= 9, min_child_weight= 6, seed= 0,
                 subsample= 0.8, colsample_bytree= 0.8, gamma= 0.1, reg_alpha= 0, reg_lambda= 0.03)
model0.fit(x_train,y_train)
y_pred = model0.predict(x_test)

f, ax = plt.subplots(1)
f.set_figheight(6)
f.set_figwidth(20)
ax.grid(True)
line1, = ax.plot(y_pred,label='xgb_forecast')
line2, =ax.plot(y_test.values,label='ground_truth')
plt.legend()
# plt.xlim(0, 100)
rmse_xgb = sqrt(mean_squared_error(y_test, y_pred))
print('RMSE of pIC50 XGBoost forecast: {}'
      .format(round(rmse_xgb, 3)))

print(accuracy(y_test.values, y_pred))

# plt.scatter(y_test.values,y_pred)
# dtest = xgb.DMatrix(X_test)
# model.fit(X_train, y_train)
# y_pred = model.predict(X_test)
# y_pred = list(map(lambda X: X if X >= 0 else 0,y_pred))
# MSE = np.sqrt(sum((np.array(y_test) - np.array(y_pred)) ** 2 ) / len(y_pred) ) #均方根误差作为结果
# R2  = r2_score(y_test,y_pred)
# plt.scatter(y_test,y_pred,s=20)
```

```
RMSE of pIC50 XGBoost forecast: 0.706
0.9177242936117267
```

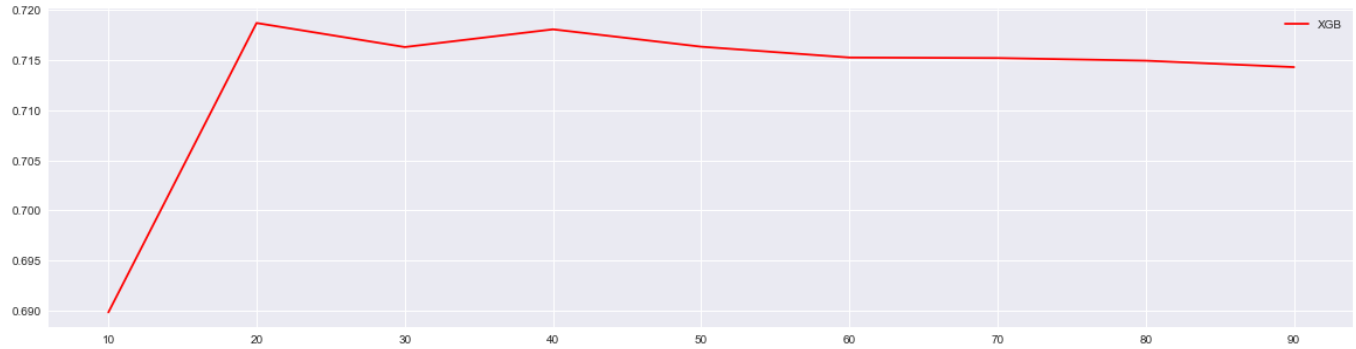
In [2241]:
```
df_sh = pd.DataFrame(data =shap_values, columns = cols).mean()
top20 = df_sh.sort_values(ascending = False).head(20)

Molecular_20 = Molecular[colname_sh]
Molecular_20.insert(0, 'SMILES', Molecular_SMILES)
Molecular_20n = normalized (Molecular_20)

Molecular_20n.to_excel(r'D:\documents\my work\py\D\Molecular_20n.xlsx')
```

Q2

In [2242]:
```
x1 = Molecular_20n.iloc[:,1:]
y1  = Activity['pIC50']

x1_train,x1_test,y1_train,y1_test = TTS(x1,y1,test_size=0.1,random_state=420)
```

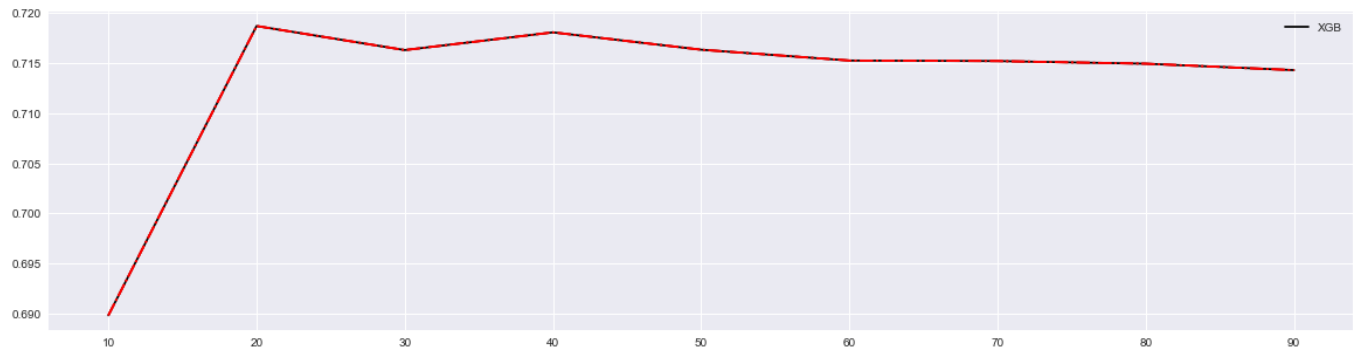
In [2243]:
```
axisx = range(10,100,10)
rs = []
for i in axisx:
    reg = XGBR(n_estimators=i,random_state=420)
    rs.append(CVS(reg,x1_train,y1_train,cv=5).mean())
print(axisx[rs.index(max(rs))],max(rs))
plt.figure(figsize=(20,5))
plt.plot(axisx,rs,c="red",label="XGB")
plt.legend()
plt.show()
```

```
20 0.7186952009312132
```



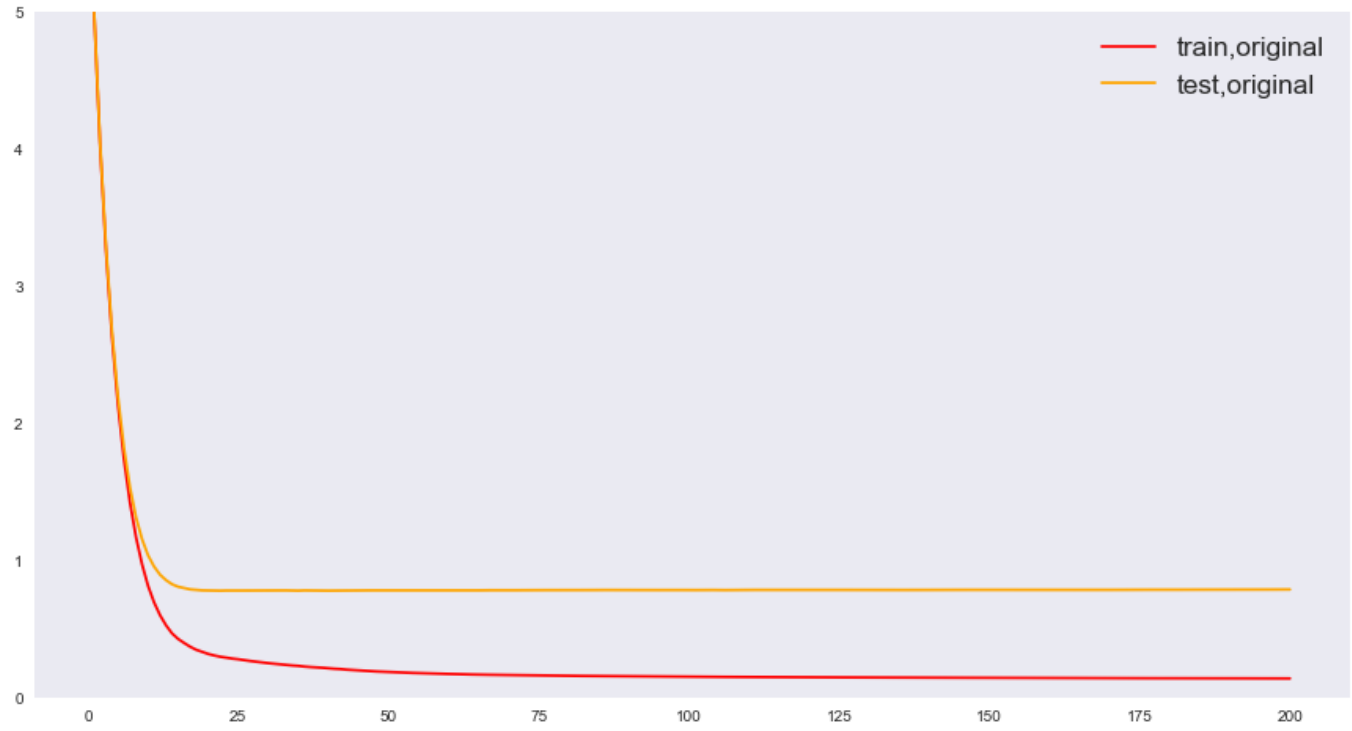
In [2244]:
```
axisx = range(10,100,10)
rs = []
var = []
ge = []
for i in axisx:
    reg = XGBR(n_estimators=i,random_state=420)
    cvresult = CVS(reg,x1_train,y1_train,cv=5)
    rs.append(cvresult.mean())
    var.append(cvresult.var())
    ge.append((1 - cvresult.mean())**2+cvresult.var())
print(axisx[rs.index(max(rs))],max(rs),var[rs.index(max(rs))])
print(axisx[var.index(min(var))],rs[var.index(min(var))],min(var))
print(axisx[ge.index(min(ge))],rs[ge.index(min(ge))],var[ge.index(min(ge))],min(ge))
rs = np.array(rs)
var = np.array(var)*0.01
plt.figure(figsize=(20,5))
plt.plot(axisx,rs,c="black",label="XGB")
#添加方差线
plt.plot(axisx,rs+var,c="red",linestyle='-.')
plt.plot(axisx,rs-var,c="red",linestyle='-.')
plt.legend()
plt.show()
```

```
20 0.7186952009312132 8.603895623357994e-05
20 0.7186952009312132 8.603895623357994e-05
20 0.7186952009312132 8.603895623357994e-05 0.07921842893536409
```



In [2245]:
```
dfull2 = xgb.DMatrix(x1_train,y1_train)

param2 = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 10, 'min_child_weight': 5, 'seed': 0,
                    'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0.1, 'reg_lambda': 0.05}
num_round = 200

cvresult2 = xgb.cv(param2, dfull2, num_round)

fig,ax = plt.subplots(1,figsize=(15,8))
ax.set_ylim(top=5)
ax.grid()
ax.plot(range(1,201),cvresult2.iloc[:,0],c="red",label="train,original")
ax.plot(range(1,201),cvresult2.iloc[:,2],c="orange",label="test,original")
ax.legend(fontsize="xx-large")
plt.show()
```

```
[07:11:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "n_estimators" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[07:11:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "n_estimators" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[07:11:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "n_estimators" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

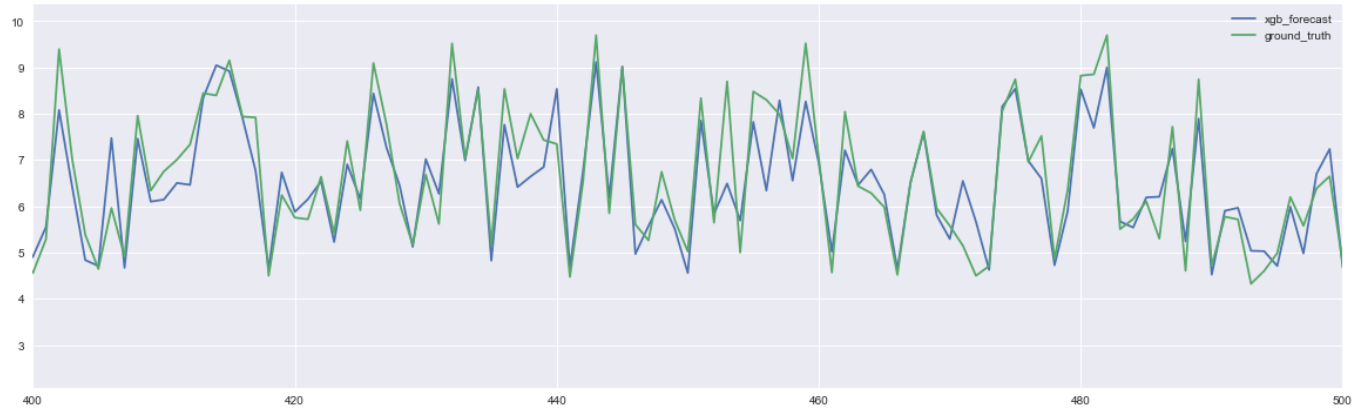


In [2246]:
```
from xgboost import XGBRegressor as XGBR
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.linear_model import LinearRegression as LinearR
from sklearn.datasets import load_boston
from sklearn.model_selection import KFold, cross_val_score as CVS, train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from time import time
import datetime


x1 = Molecular_20n.iloc[:,1:]
y1  = Activity['pIC50']
if __name__ == '__main__':
    x1_train,x1_test,y1_train,y1_test = TTS(x1,y1,test_size=0.2,random_state=420)
    cv_params = {'max_depth': [3, 4, 5, 6, 7, 8, 9, 10], 'min_child_weight': [1, 2, 3, 4, 5, 6]}
    other_params1 = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 10, 'min_child_weight': 5, 'seed': 0,
                     'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0.1, 'reg_lambda': 0.05}

    model1 = xgb.XGBRegressor(**other_params1)
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='r2', cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(x1_train, y1_train)
    evalute_result = optimized_GBM.cv_results_
    print('每轮迭代运行结果:{0}'.format(evalute_result))
    print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
    print('最佳模型得分:{0}'.format(optimized_GBM.best_score_))
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 128 tasks      | elapsed:    2.2s
[Parallel(n_jobs=4)]: Done 233 out of 240 | elapsed:    5.2s remaining:    0.1s
[Parallel(n_jobs=4)]: Done 240 out of 240 | elapsed:    5.3s finished
```

In [2247]:
```
from sklearn.metrics import accuracy_score,roc_auc_score
from sklearn.linear_model import LinearRegression
import statistics
x1_train,x1_test,y1_train,y1_test = TTS(x1,y1,test_size=0.3,random_state=50)
# print(x1_train)
other_params1 = {'learning_rate': 0.2, 'n_estimators': 20, 'max_depth': 10, 'min_child_weight': 5, 'seed': 0,
                 'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0.1, 'reg_alpha': 0.1, 'reg_lambda': 0.05}
model1 = xgb.XGBRegressor(**other_params1)
model1.fit(x1_train, y1_train)
y1_pred = model1.predict(x1_test)
f, ax = plt.subplots(1)
f.set_figheight(6)
f.set_figwidth(20)
ax.grid(True)
line1, = ax.plot(y1_pred,label='xgb_forecast')
line2, =ax.plot(y1_test.values,label='ground_truth')
plt.legend()
plt.xlim(400, 500)
rmse_xgb1 = sqrt(mean_squared_error(y1_test, y1_pred))

print('RMSE of pIC50 XGBoost forecast: {}'
      .format(round(rmse_xgb1, 3)))
print(accuracy(y1_test.values,y1_pred))

xgb_2df = pd.DataFrame(columns = ['true','xgb_pre'])
xgb_2df['true'] = y1_test.values
xgb_2df['xgb_pre'] = y1_pred
xgb_2df.to_excel(r'D:\documents\my work\py\D\xgb_2df.xlsx')
```

```
RMSE of pIC50 XGBoost forecast: 0.767
0.9124365648053261
```



In [2248]:
```
Molecular_t = pd.read_excel(path1, sheet_name = 'test')
Molecular_20t = Molecular_t[colname_sh]
Molecular_20t.insert(0, 'SMILES', Molecular_SMILES)
Molecular_20t = normalized (Molecular_20t)

Molecular_20t.to_excel(r'D:\documents\my work\py\D\Molecular_20t.xlsx')
model2 = xgb.XGBRegressor(**other_params1)
x2 = Molecular_20t.iloc[:,1:]
model2.fit(x1, y1)
y2_pred = model2.predict(x2)
```

In [2249]:
```
pred2 = y2_pred.tolist()
df_pred = pd.DataFrame(columns = ['pIC50_pred'])
df_pred['pIC50_pred'] = pred2
df_pred.to_excel(r'D:\documents\my work\py\D\pIC50_pred.xlsx')
```

Q3

In [2250]:
```
path3 = r'D:\documents\my work\py\D\ADMET.xlsx'
ADMET = pd.read_excel(path3, sheet_name = 'training')
```

In [2251]:
```
x3 = Molecular_c.iloc[:,1:]
y3  = ADMET.iloc[:,1:]
x3_train,x3_test,y3_train,y3_test = TTS(x3,y3,test_size=0.3,random_state=420)
```

In [2252]:
```
from xgboost import XGBRegressor as XGBR
from xgboost.sklearn import XGBClassifier as XGBC
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.linear_model import LinearRegression as LinearR
from sklearn.datasets import load_boston
from sklearn.model_selection import KFold, cross_val_score as CVS, train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from time import time
import datetime

if __name__ == '__main__':

    cv_params = {'learning_rate': [0.01, 0.05, 0.07, 0.1, 0.2]}
    other_params3 = {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':50,'max_depth':7,'min_child_weight':
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0, 'reg_lambda':3}

    model_AMDET = xgb.XGBClassifier(**other_params3)
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='r2', cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(x3_train, y3_train.iloc[:,4])
    evalute_result = optimized_GBM.cv_results_
    print('每轮迭代运行结果:{0}'.format(evalute_result))
    print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
    print('最佳模型得分:{0}'.format(optimized_GBM.best_score_))
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  25 out of  25 | elapsed:    1.7s finished

每轮迭代运行结果:{'mean_fit_time': array([0.24634042, 0.26050282, 0.26429338, 0.26150055, 0.23836265]), 'std_fit_time': a
rray([0.00403955, 0.0023937 , 0.00846283, 0.00510116, 0.05386691]), 'mean_score_time': array([0.00638289, 0.00638313, 0.0
0538588, 0.00618367, 0.00598383]), 'std_score_time': array([0.00048905, 0.00048846, 0.00048842, 0.00097737, 0.00063037]),
'param_learning_rate': masked_array(data=[0.01, 0.05, 0.07, 0.1, 0.2],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'learning_rate': 0.01}, {'learning_rate': 0.05}, {'learning_rate': 0.07}, {'learni
ng_rate': 0.1}, {'learning_rate': 0.2}], 'split0_test_score': array([0.04519306, 0.62551767, 0.73556605, 0.77034144, 0.81
761293]), 'split1_test_score': array([-0.11802007,  0.56397483,  0.66328792,  0.72896956,  0.75688875]), 'split2_test_sco
re': array([-0.12783854,  0.49866171,  0.61258143,  0.68665547,  0.69534709]), 'split3_test_score': array([-0.01606842,
0.5579007 ,  0.6448105 ,  0.6710546 ,  0.69919294]), 'split4_test_score': array([-0.08191212,  0.53053161,  0.623641  ,
0.68701703,  0.75807597]), 'mean_test_score': array([-0.05972922,  0.5553173 ,  0.65597738,  0.70880762,  0.74542354]),
'std_test_score': array([0.06548852, 0.0420331 , 0.04345423, 0.03628706, 0.04505055]), 'rank_test_score': array([5, 4, 3,
2, 1])}
参数的最佳取值：{'learning_rate': 0.2}
最佳模型得分:0.7454235365774624
```

In [2253]:
```
from sklearn.metrics import accuracy_score,roc_auc_score
from xgboost.sklearn import XGBClassifier
import statistics
# print(x1_train)
other_params3 = {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':150,'max_depth':7,'min_child_weight':5,
                 'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0.05, 'reg_lambda':3}



model_AMDET1 = xgb.XGBClassifier(**other_params3)
model_AMDET1.fit(x3_train, y3_train.iloc[:,0])
y3_pred = model_AMDET1.predict(x3_test)
f, ax = plt.subplots(1)
f.set_figheight(6)
f.set_figwidth(20)
ax.grid(True)

ax.scatter(x3_test.index,y3_test.iloc[:,0].values,label='ground_truth',  c='blue', alpha=0.7)  # 改变颜色
ax.scatter(x3_test.index,y3_pred,label='xgb_forecast', c='red', alpha=0.7)



plt.legend()
# plt.xlim(400, 500)
# print(accuracy(y3_test.values,y3_pred))
```

```
[07:11:29] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

Out[2253]:
```
<matplotlib.legend.Legend at 0x25031c1ba60>
```

In [2254]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
def print_precison_recall_f1(y_true, y_pre):
    """打印精准率、召回率和F1值"""
    print("打印精准率、召回率和F1值")
    print(classification_report(y_true, y_pre))
    f1 = round(f1_score(y_true, y_pre, average='macro'), 2)
    p = round(precision_score(y_true, y_pre, average='macro'), 2)
    r = round(recall_score(y_true, y_pre, average='macro'), 2)
    print("Precision: {}, Recall: {}, F1: {} ".format(p, r, f1))
```

In [2255]:
```
print_precison_recall_f1(y3_test.iloc[:,0].values, y3_pred)
```

```
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       388
           1       0.85      0.90      0.87       205

    accuracy                           0.91       593
   macro avg       0.90      0.91      0.90       593
weighted avg       0.91      0.91      0.91       593

Precision: 0.9, Recall: 0.91, F1: 0.9
```

In [2256]:
```
def xgb_AMDET(i):
    params_AMDET = []
    params_AMDET = [{'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':150,'max_depth':7,'min_child_weight'
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0.05, 'reg_lambda':3},

                {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':220,'max_depth':9,'min_child_weight':6,
                              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0.05, 'reg_lambda':3},

                   {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':70,'max_depth':5,'min_child_weight':3
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.3,  'reg_alpha':2, 'reg_lambda':0.05},

                   {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':200,'max_depth':6,'min_child_weight':
              'subsample':0.8, 'colsample_bytree':0.7, 'gamma':0.3,  'reg_alpha':1, 'reg_lambda':2},

                   {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':50,'max_depth':7,'min_child_weight':6
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0, 'reg_lambda':3}]
    model_AMDET = xgb.XGBClassifier(**params_AMDET[i])
    model_AMDET.fit(x3_train, y3_train.iloc[:,i])
    y3_pred = model_AMDET.predict(x3_test)
    print('第'+str(i)+'个XGB模型结果输出')
    print("准确率为 %2.3f" % accuracy_score(y3_test.iloc[:,i], y3_pred))
    print_precison_recall_f1(y3_test.iloc[:,i].values, y3_pred)

    err = []
    index = []
    for j in range(len(y3_pred)):
        if y3_pred[j] == y3_test.iloc[:,i].values[j]:
            pass
        else:
            err.append(y3_pred[j])
            index.append(j)

    f, ax = plt.subplots(1)
    f.set_figheight(6)
    f.set_figwidth(20)
    ax.grid(True)

    ax.scatter(x3_test.index,y3_test.iloc[:,i].values,label='ground_truth',  c='blue', alpha=0.7)  # 改变颜色
    ax.scatter(index,err,label='err', c='red', alpha=0.7)
    plt.legend()
    xgb_AMDET = pd.DataFrame(columns= ['index_truth','Truth','index_err','err'])
    xgb_AMDET['index_truth'] = x3_test.index
    xgb_AMDET['Truth'] = y3_test.iloc[:,i].values
    l=len(x3_test.index)
    l_err = len(index)
    lis1=[0]*l
    lis2=[0]*l
    lis1[:l_err] = index
    lis2[:l_err] = err
    xgb_AMDET['index_err'] = lis1
    xgb_AMDET['err'] = lis2
    xgb_AMDET.to_excel(r'D:\documents\my work\py\D\xgb'+str(i)+'AMDET.xlsx')
    return y3_pred
```

In [2257]:
```
def xgb_pre(i):
    Molecular_tn = normalized (Molecular_t)
    SMILES_tn = Molecular_t['SMILES']
    x3_train_total = Molecular_c.iloc[:,1:]
    y3_train_total = ADMET.iloc[:,1:]
    x3_pre_total = Molecular_tn[colnames_clean]

    params_AMDET = []
    params_AMDET = [{'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':150,'max_depth':7,'min_child_weight'
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0.05, 'reg_lambda':3},

                {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':220,'max_depth':9,'min_child_weight':6,
                              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0.05, 'reg_lambda':3},

                   {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':70,'max_depth':5,'min_child_weight':3
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.3,  'reg_alpha':2, 'reg_lambda':0.05},

                   {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':200,'max_depth':6,'min_child_weight':
              'subsample':0.8, 'colsample_bytree':0.7, 'gamma':0.3,  'reg_alpha':1, 'reg_lambda':2},

                   {'objective': 'binary:logistic','learning_rate':0.2, 'n_estimators':50,'max_depth':7,'min_child_weight':6
              'subsample':0.8, 'colsample_bytree':0.8, 'gamma':0.1,  'reg_alpha':0, 'reg_lambda':3}]

    model_AMDET_f = xgb.XGBClassifier(**params_AMDET[i])
    model_AMDET_f.fit(x3_train_total, y3_train_total.iloc[:,i])

    y3_pred_total = model_AMDET_f.predict(x3_pre_total)

    df3_pred = pd.DataFrame(columns = [str(i)])
    df3_pred[str(i)] = y3_pred_total

    return df3_pred[str(i)].values
```

In [2258]:
```
xgb_pre3 = pd.DataFrame(columns = ['Caco-2','CYP3A4','hERG','HOB','MN'])
xgb_pre3['Caco-2'] = xgb_pre(0)
# xgb_pre3['CYP3A4']= xgb_pre(1)
# xgb_pre3['hERG']= xgb_pre(2)
# xgb_pre3['HOB']= xgb_pre(3)
# xgb_pre3['MN']= xgb_pre(4)

# xgb_pre3.to_excel(r'D:\documents\my work\py\D\xgb_pre3.xlsx')
```

```
[07:11:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost
1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Exp
licitly set eval_metric if you'd like to restore the old behavior.
```

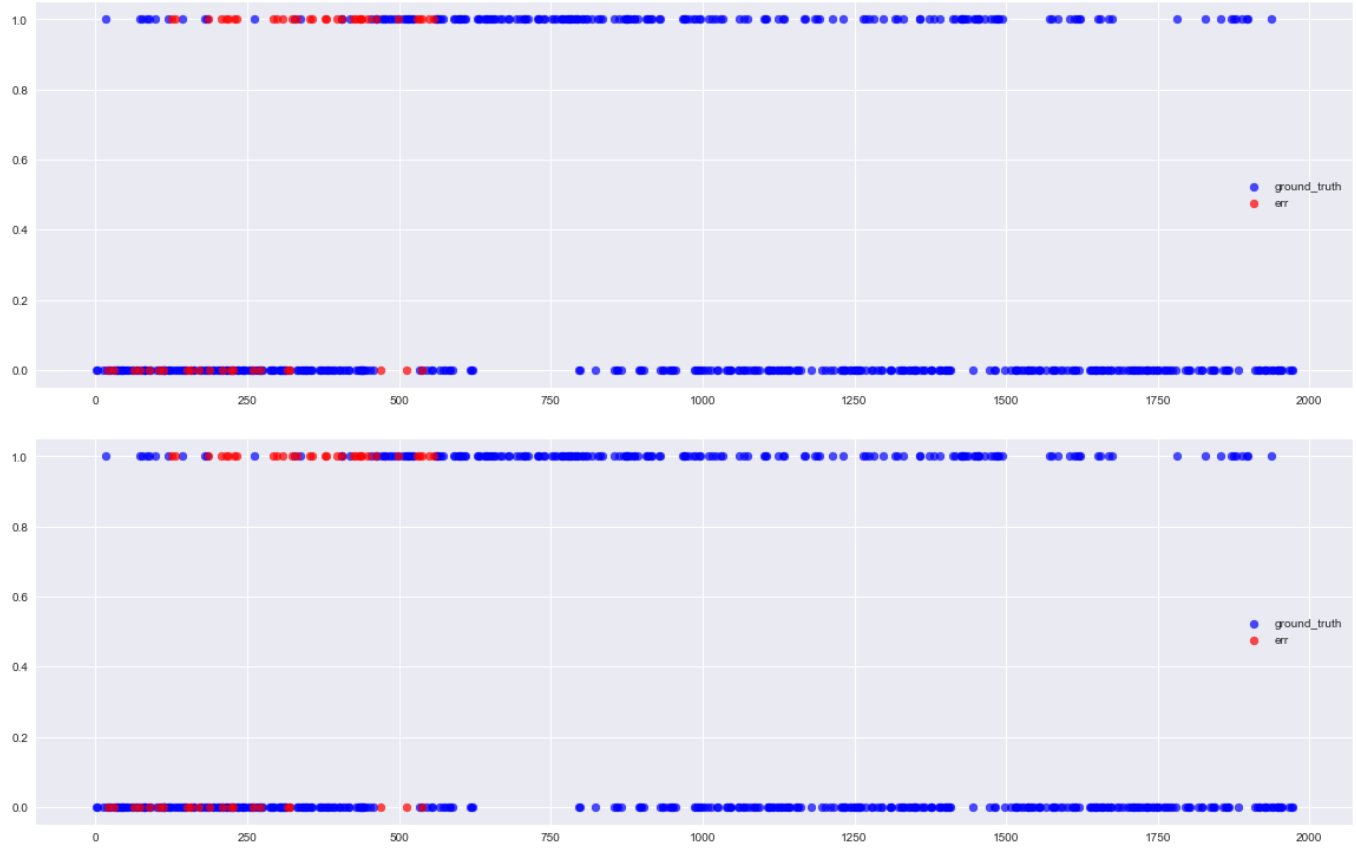
In [2259]:

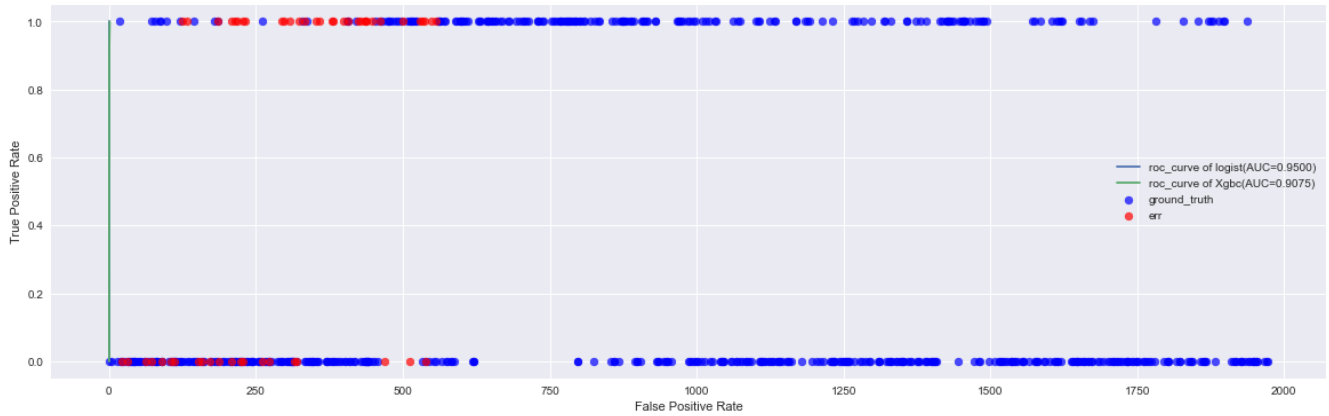
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
def LR_AMDET(i):
    logreg = LogisticRegression()
    logreg.fit(x3_train, y3_train.iloc[:,i])
    y3_pred_log = logreg.predict(x3_test)

    print('Train/Test split results:')
    print('第'+str(i)+'个模型的LR算法准确率为 %2.3f' % accuracy_score(y3_test.iloc[:,i], y3_pred_log))
    print_precison_recall_f1(y3_test.iloc[:,i].values, y3_pred_log)
    err_log = []
    index_log = []
    for j in range(len(y3_pred_log)):
        if y3_pred_log[j] == y3_test.iloc[:,i].values[j]:
            pass
        else:
            err_log.append(y3_pred_log[j])
            index_log.append(j)


    LR_AMDET = pd.DataFrame(columns= ['index_truth','Truth','index_err','err'])
    LR_AMDET['index_truth'] = x3_test.index
    LR_AMDET['Truth'] = y3_test.iloc[:,i].values

    l=len(x3_test.index)
    l_err = len(index_log)
    lis1=[0]*l
    lis2=[0]*l
    lis1[:l_err] = index_log
    lis2[:l_err] = err_log

    LR_AMDET['index_err'] = lis1
    LR_AMDET['err'] = lis2
    LR_AMDET.to_excel(r'D:\documents\my work\py\D\LR'+str(i)+'AMDET.xlsx')

    return y3_pred_log
```

In [2260]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import f1_score,precision_score,recall_score,roc_auc_score,accuracy_score,roc_curve
import matplotlib.pyplot as plt
from sklearn import metrics
for i in range(5):
    xgb_AMDET(i)


    Xgbc_auc = roc_auc_score(y3_test.iloc[:,i],xgb_AMDET(i))
    Xgbc_fpr,Xgbc_tpr,Xgbc_threasholds=metrics.roc_curve(y3_test.iloc[:,i],xgb_AMDET(i)) # 计算ROC的值,svm_threasholds为阈值

    LR_AMDET(i)
    lr = LogisticRegression()
    lr.fit(x3_train, y3_train.iloc[:,i])
    lr_y_proba=lr.predict_proba(x3_test)
    lr_auc=roc_auc_score(y3_test.iloc[:,i],lr_y_proba[:,1])
    lr_fpr,lr_tpr,lr_threasholds=roc_curve(y3_test.iloc[:,i],lr_y_proba[:,1]) # 计算ROC的值,lr_threasholds为阈值

#   plt.title("roc_curve of %s(AUC=%.4f)" %('logist',lr_auc))
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.plot(lr_fpr,lr_tpr,label = "roc_curve of %s(AUC=%.4f)" %('logist',lr_auc) )
    plt.plot(Xgbc_fpr,Xgbc_tpr,label = "roc_curve of %s(AUC=%.4f)" %('Xgbc',Xgbc_auc))
    plt.legend()
    plt.show()
```

```
[07:11:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第0个XGB模型结果输出
准确率为 0.911
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       388
           1       0.85      0.90      0.87       205

    accuracy                           0.91       593
   macro avg       0.90      0.91      0.90       593
weighted avg       0.91      0.91      0.91       593

Precision: 0.9, Recall: 0.91, F1: 0.9
[07:11:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第0个XGB模型结果输出
准确率为 0.911
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       388
           1       0.85      0.90      0.87       205

    accuracy                           0.91       593
   macro avg       0.90      0.91      0.90       593
weighted avg       0.91      0.91      0.91       593

Precision: 0.9, Recall: 0.91, F1: 0.9

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[07:11:33] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第0个XGB模型结果输出
准确率为 0.911
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       388
           1       0.85      0.90      0.87       205
```

```
    accuracy                           0.91       593
   macro avg       0.90      0.91      0.90       593
weighted avg       0.91      0.91      0.91       593

Precision: 0.9, Recall: 0.91, F1: 0.9

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

Train/Test split results:
第0个模型的LR算法准确率为 0.885
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.93      0.89      0.91       388
           1       0.81      0.87      0.84       205

    accuracy                           0.89       593
   macro avg       0.87      0.88      0.88       593
weighted avg       0.89      0.89      0.89       593

Precision: 0.87, Recall: 0.88, F1: 0.88
```





The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[07:11:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第1个XGB模型结果输出
准确率为 0.946
打印精准率、召回率和F1值

	precision	recall	f1-score	support
0	0.89	0.89	0.89	144
1	0.96	0.96	0.96	449
accuracy			0.95	593
macro avg	0.93	0.93	0.93	593
weighted avg	0.95	0.95	0.95	593

Precision: 0.93, Recall: 0.93, F1: 0.93
[07:11:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第1个XGB模型结果输出
准确率为 0.946
打印精准率、召回率和F1值

	precision	recall	f1-score	support
0	0.89	0.89	0.89	144
1	0.96	0.96	0.96	449
accuracy			0.95	593
macro avg	0.93	0.93	0.93	593
weighted avg	0.95	0.95	0.95	593

Precision: 0.93, Recall: 0.93, F1: 0.93

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

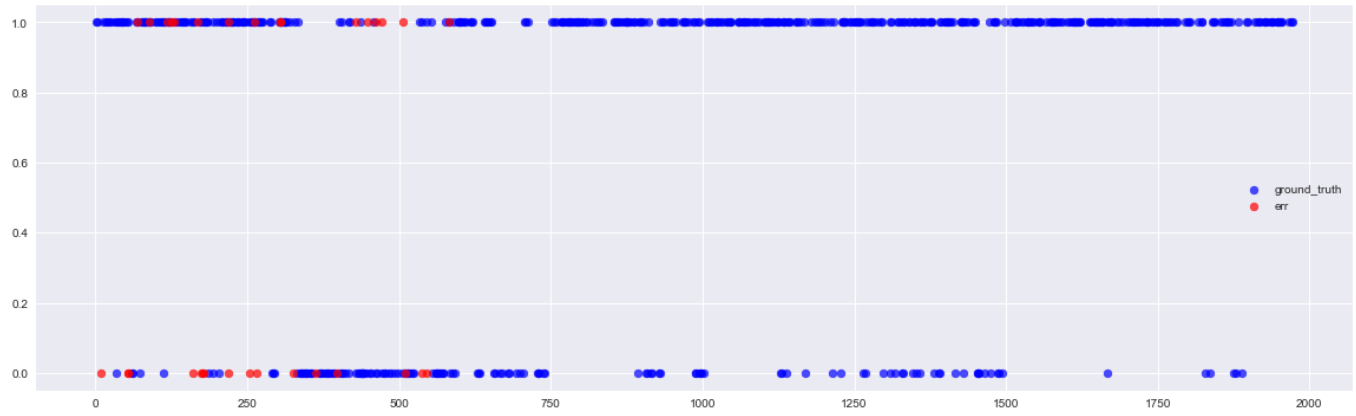
[07:11:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第1个XGB模型结果输出
准确率为 0.946
打印精准率、召回率和F1值

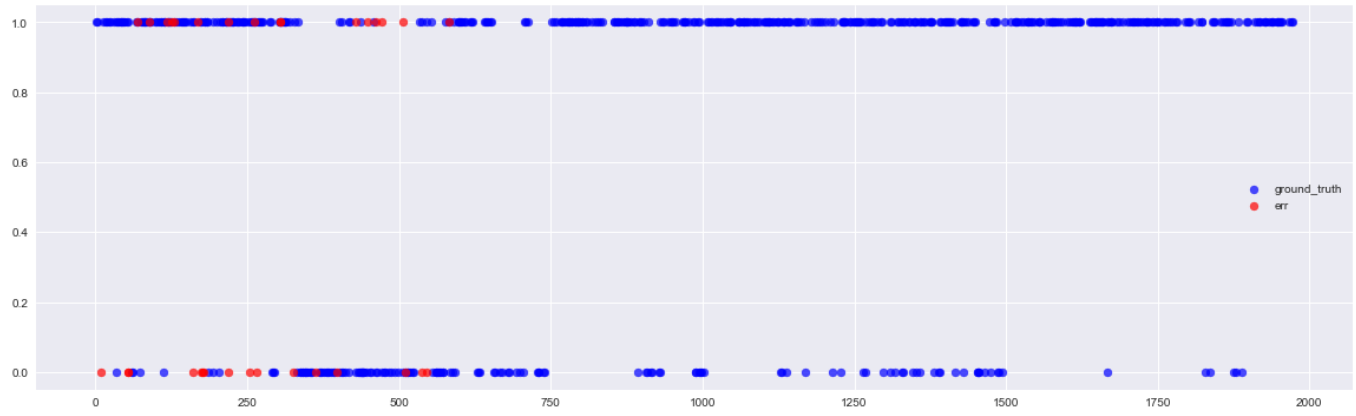
	precision	recall	f1-score	support
0	0.89	0.89	0.89	144
1	0.96	0.96	0.96	449
accuracy			0.95	593
macro avg	0.93	0.93	0.93	593
weighted avg	0.95	0.95	0.95	593

Precision: 0.93, Recall: 0.93, F1: 0.93

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

Train/Test split results:
第1个模型的LR算法准确率为 0.938
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       144
           1       0.96      0.96      0.96       449

    accuracy                           0.94       593
   macro avg       0.92      0.91      0.91       593
weighted avg       0.94      0.94      0.94       593

Precision: 0.92, Recall: 0.91, F1: 0.91
```





The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[07:11:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第2个XGB模型结果输出
准确率为 0.897
打印精准率、召回率和F1值
              precision   recall f1-score   support

           0   0.87   0.89   0.88   247
           1   0.92   0.90   0.91   346

    accuracy   0.90   593
   macro avg   0.89   0.90   0.89   593
weighted avg   0.90   0.90   0.90   593

Precision: 0.89, Recall: 0.9, F1: 0.89
[07:11:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第2个XGB模型结果输出
准确率为 0.897
打印精准率、召回率和F1值
              precision   recall f1-score   support

           0   0.87   0.89   0.88   247
           1   0.92   0.90   0.91   346

    accuracy   0.90   593
   macro avg   0.89   0.90   0.89   593
weighted avg   0.90   0.90   0.90   593

Precision: 0.89, Recall: 0.9, F1: 0.89
[07:11:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

第2个XGB模型结果输出
准确率为 0.897
打印精准率、召回率和F1值
              precision   recall f1-score   support

           0   0.87   0.89   0.88   247
           1   0.92   0.90   0.91   346

    accuracy   0.90   593
   macro avg   0.89   0.90   0.89   593
weighted avg   0.90   0.90   0.90   593

Precision: 0.89, Recall: 0.9, F1: 0.89

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

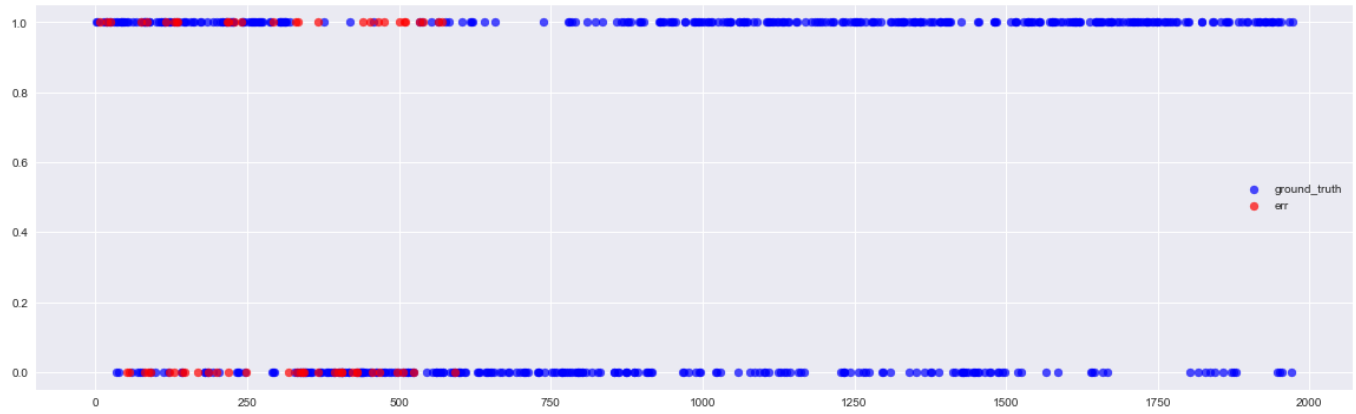
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)

Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

Train/Test split results:
第2个模型的LR算法准确率为 0.887
打印精准率、召回率和F1值

	precision	recall	f1-score	support
0	0.87	0.86	0.86	247
1	0.90	0.90	0.90	346
accuracy			0.89	593
macro avg	0.88	0.88	0.88	593
weighted avg	0.89	0.89	0.89	593

Precision: 0.88, Recall: 0.88, F1: 0.88

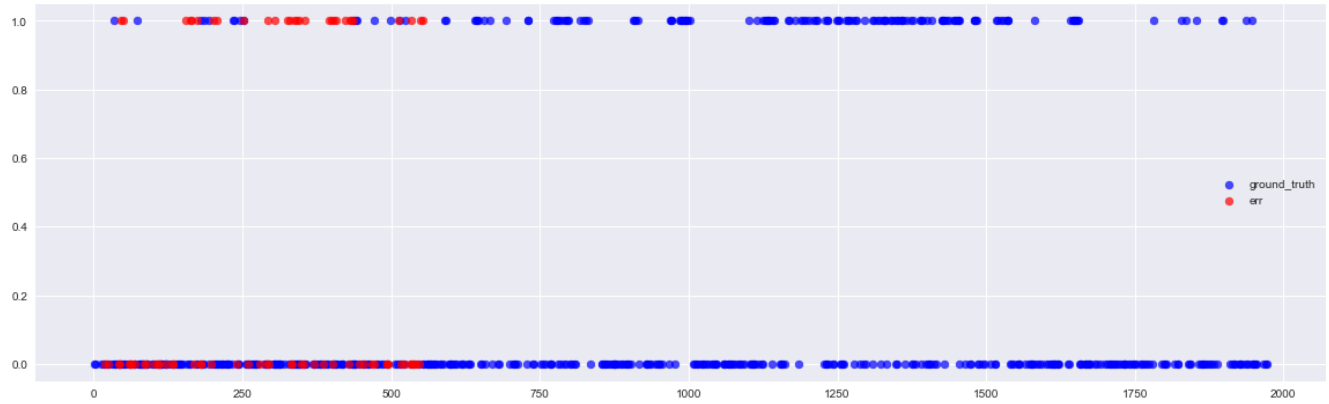


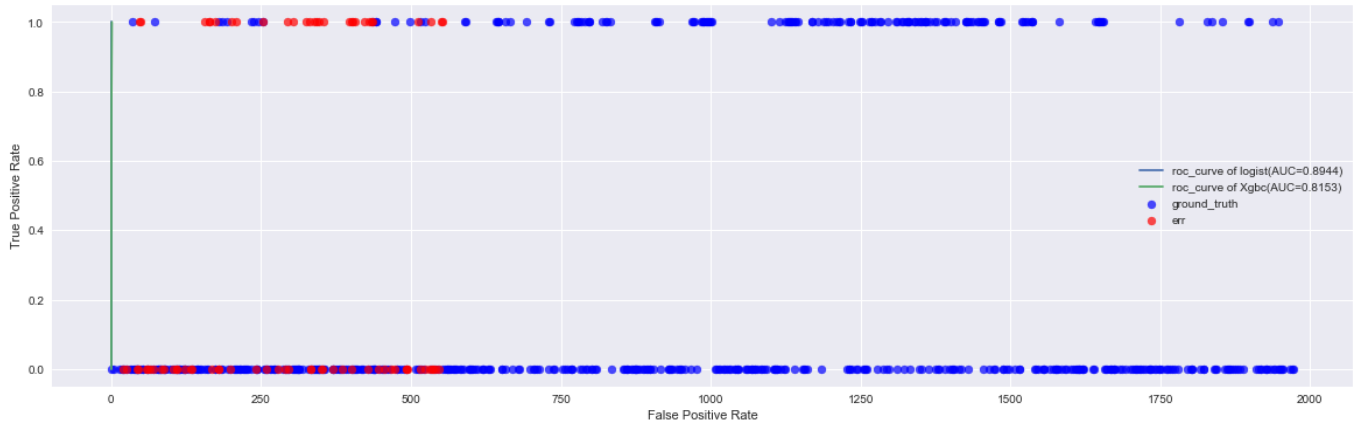
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[07:11:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第3个XGB模型结果输出
准确率为 0.872
打印精准率、召回率和F1值

```
              precision    recall  f1-score   support

           0       0.90      0.93      0.91       438
           1       0.79      0.70      0.74       155

    accuracy                           0.87       593
   macro avg       0.84      0.82      0.83       593
weighted avg       0.87      0.87      0.87       593

Precision: 0.84, Recall: 0.82, F1: 0.83
[07:11:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
第3个XGB模型结果输出
准确率为 0.872
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.90      0.93      0.91       438
           1       0.79      0.70      0.74       155

    accuracy                           0.87       593
   macro avg       0.84      0.82      0.83       593
weighted avg       0.87      0.87      0.87       593

Precision: 0.84, Recall: 0.82, F1: 0.83
[07:11:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your label
s (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

第3个XGB模型结果输出
准确率为 0.872
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.90      0.93      0.91       438
           1       0.79      0.70      0.74       155

    accuracy                           0.87       593
   macro avg       0.84      0.82      0.83       593
weighted avg       0.87      0.87      0.87       593

Precision: 0.84, Recall: 0.82, F1: 0.83

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.htm
l)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/module
s/linear_model.html#logistic-regression)
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.htm
l)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/module
s/linear_model.html#logistic-regression)

Train/Test split results:
第3个模型的LR算法准确率为 0.875
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       438
           1       0.80      0.70      0.75       155

    accuracy                           0.88       593
   macro avg       0.85      0.82      0.83       593
```

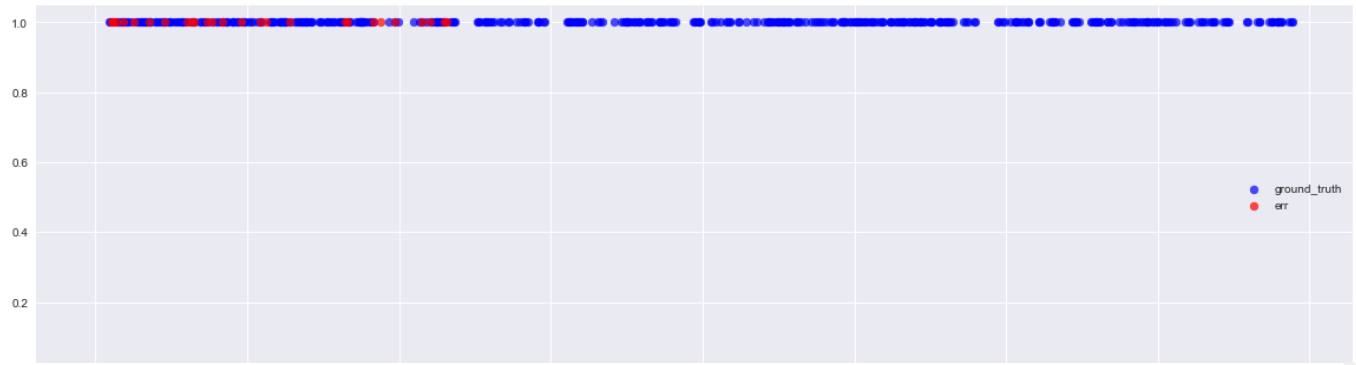
```
weighted avg       0.87      0.88      0.87       593

Precision: 0.85, Recall: 0.82, F1: 0.83
```

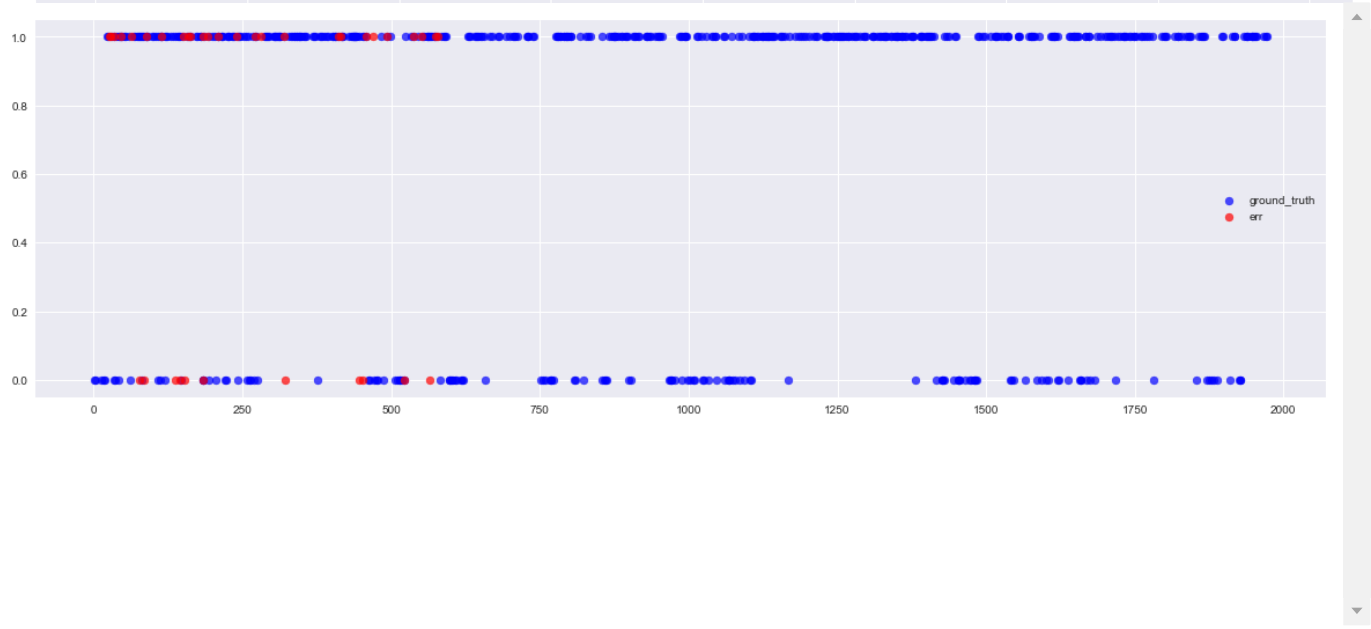


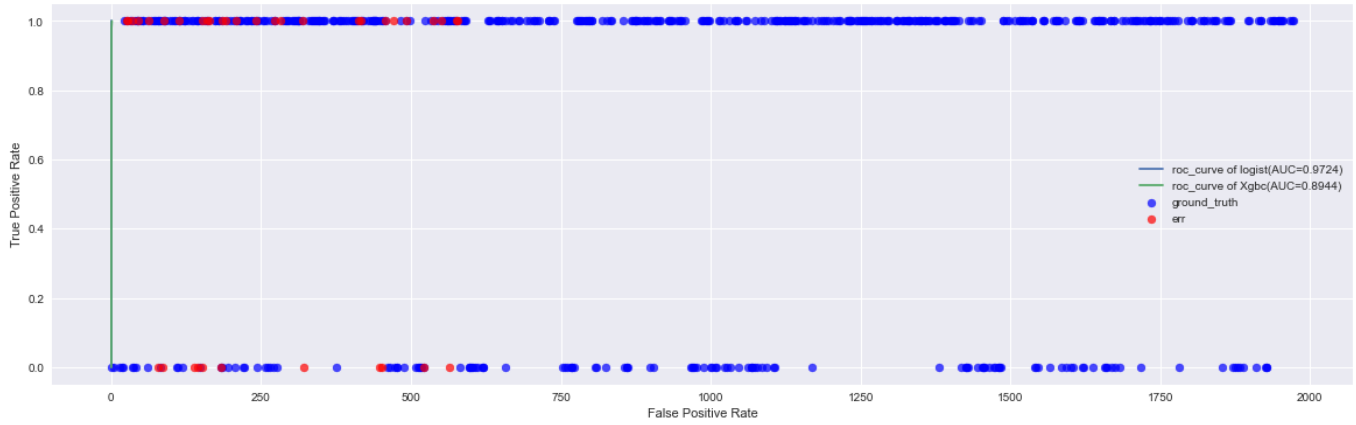



```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[07:11:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
第4个XGB模型结果输出
准确率为 0.938
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.91      0.81      0.86       139
           1       0.94      0.98      0.96       454

    accuracy                           0.94       593
   macro avg       0.93      0.89      0.91       593
```

```
weighted avg       0.94      0.94      0.94       593

Precision: 0.93, Recall: 0.89, F1: 0.91
[07:11:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
第4个XGB模型结果输出
准确率为 0.938
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.91      0.81      0.86       139
           1       0.94      0.98      0.96       454

    accuracy                           0.94       593
   macro avg       0.93      0.89      0.91       593
weighted avg       0.94      0.94      0.94       593

Precision: 0.93, Recall: 0.89, F1: 0.91
[07:11:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your label
s (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].


第4个XGB模型结果输出
准确率为 0.938
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.91      0.81      0.86       139
           1       0.94      0.98      0.96       454

    accuracy                           0.94       593
   macro avg       0.93      0.89      0.91       593
weighted avg       0.94      0.94      0.94       593

Precision: 0.93, Recall: 0.89, F1: 0.91

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.htm
l)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/module
s/linear_model.html#logistic-regression)
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.htm
l)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/module
s/linear_model.html#logistic-regression)

Train/Test split results:
第4个模型的LR算法准确率为 0.917
打印精准率、召回率和F1值
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       139
           1       0.94      0.95      0.95       454

    accuracy                           0.92       593
   macro avg       0.89      0.88      0.88       593
weighted avg       0.92      0.92      0.92       593

Precision: 0.89, Recall: 0.88, F1: 0.88
```







Q4

In [2261]:
```
import pandas as pd

path4 = r'D:\documents\my work\py\D\Molecular_Descriptor_1.xlsx'
data4 = pd.read_excel(path4, sheet_name = 'training')
```

In [2262]:
```
Molecular4 = data4.iloc[:,:-2]
Activity4 = data4['pIC50']
SUM4 = data4['SUM']
SMILES4 = data4['SMILES']
```

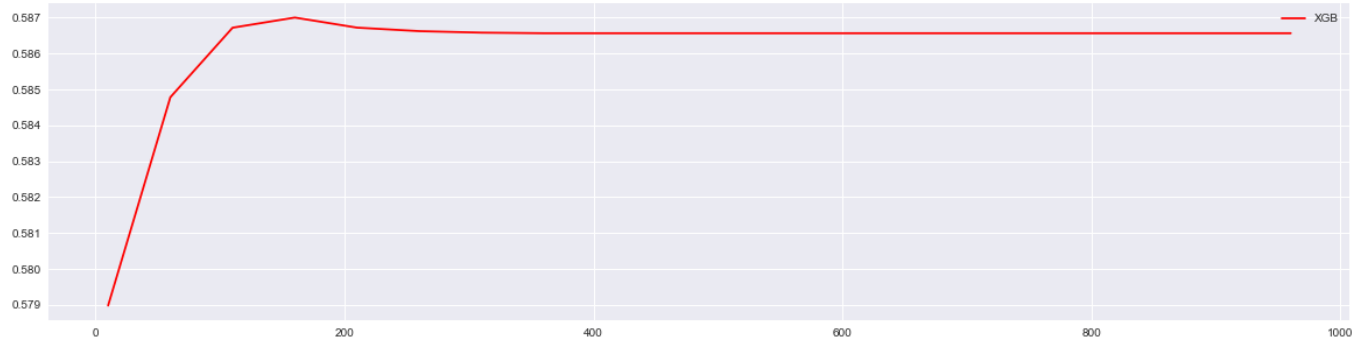
In [2263]:
```
Molecular4_s = std_clean(Molecular4,0)
Molecular4_n = normalized (Molecular4_s)
# Molecular_p = pvalue_clean(Molecular_n,Activity['pIC50'],0.05)
Molecular4_c = cor_clean(Molecular4_n,0.9)
colnames_clean4 = Molecular4_c.iloc[:,1:].columns.values
```

```
from Remove std
The initial set of 730 descriptors has been reduced to 505 descriptors.
from Remove correlation
The initial set of 505 descriptors has been reduced to 254 descriptors.
```

In [2264]:
```
x4 = Molecular4_c.iloc[:,1:]
y4 = SUM4
```

In [2265]:
```
axisx = range(10,1000,50)
rs = []
for i in axisx:
    reg = XGBR(n_estimators=i,random_state=420)
    rs.append(CVS(reg,x4,y4,cv=5).mean())
print(axisx[rs.index(max(rs))],max(rs))
plt.figure(figsize=(20,5))
plt.plot(axisx,rs,c="red",label="XGB")
plt.legend()
plt.show()
```

```
160 0.5870020055885622
```



In [ ]:

In [2266]:
```
from xgboost import XGBRegressor as XGBR
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.linear_model import LinearRegression as LinearR
from sklearn.datasets import load_boston
from sklearn.model_selection import KFold, cross_val_score as CVS, train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from time import time
import datetime


if __name__ == '__main__':
    x4_train,x4_test,y4_train,y4_test = TTS(x4,y4,test_size=0.1,random_state=420)

    cv_params = {'learning_rate': [0.01, 0.05, 0.07, 0.1, 0.2]}
    other_params4 = {'learning_rate': 0.2, 'n_estimators': 70, 'max_depth': 10, 'min_child_weight': 3, 'seed': 0,
                     'subsample': 0.6, 'colsample_bytree': 0.6, 'gamma': 0.5, 'reg_alpha': 0.1, 'reg_lambda': 0.1}

    model4 = xgb.XGBRegressor(**other_params4)
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='r2', cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(x4_train, y4_train)
    evalute_result = optimized_GBM.cv_results_
    print('每轮迭代运行结果:{0}'.format(evalute_result))
    print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
    print('最佳模型得分:{0}'.format(optimized_GBM.best_score_))
```

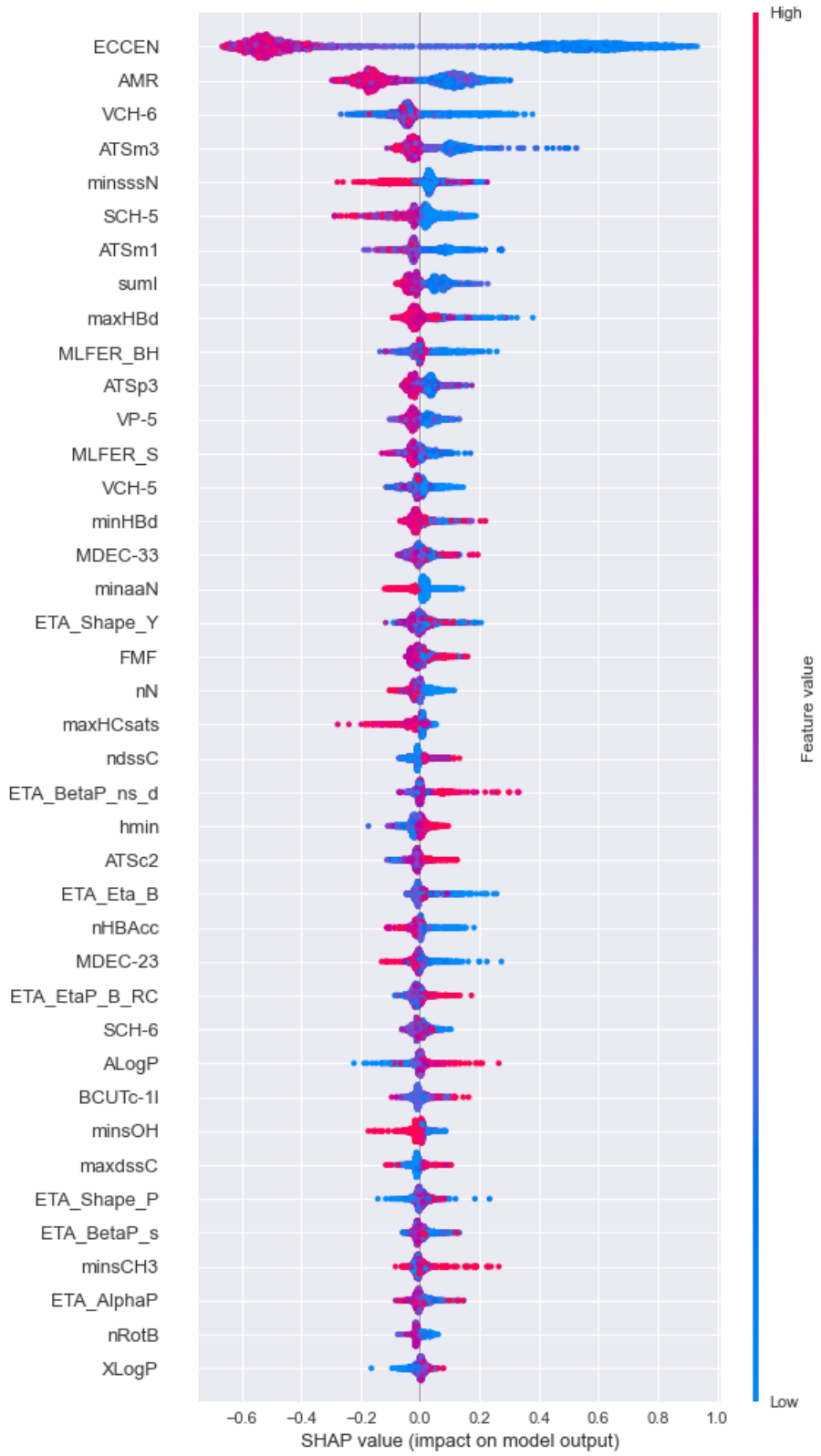
```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  25 out of  25 | elapsed:    2.6s finished

每轮迭代运行结果:{'mean_fit_time': array([0.41887965, 0.43024921, 0.40411935, 0.40431814, 0.338096  ]), 'std_fit_time': a
rray([0.01899663, 0.01611343, 0.00628885, 0.00811734, 0.0794491 ]), 'mean_score_time': array([0.00937529, 0.00837793, 0.0
0738053, 0.00718093, 0.00678167]), 'std_score_time': array([0.00079807, 0.00173922, 0.00149263, 0.00097699, 0.00039964]),
'param_learning_rate': masked_array(data=[0.01, 0.05, 0.07, 0.1, 0.2],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'learning_rate': 0.01}, {'learning_rate': 0.05}, {'learning_rate': 0.07}, {'learni
ng_rate': 0.1}, {'learning_rate': 0.2}], 'split0_test_score': array([-0.15608228,  0.60133471,  0.71202752,  0.74783612,
0.77510416]), 'split1_test_score': array([-0.19657076,  0.62048787,  0.74081813,  0.80125045,  0.81714288]), 'split2_test
_score': array([-0.20867783,  0.58804236,  0.69803641,  0.76121055,  0.8003966 ]), 'split3_test_score': array([-0.1849550
3,  0.58983628,  0.70531381,  0.77719992,  0.80816888]), 'split4_test_score': array([-0.19005493,  0.60468826,  0.7159129
8,  0.77194855,  0.79234452]), 'mean_test_score': array([-0.18726817,  0.6008779 ,  0.71442177,  0.77188912,  0.7986314
1]), 'std_test_score': array([0.01749622, 0.01171279, 0.01453381, 0.01779255, 0.01435138]), 'rank_test_score': array([5,
4, 3, 2, 1])}
参数的最佳取值：{'learning_rate': 0.2}
最佳模型得分:0.7986314086517993
```

In [2267]:
```
model4 = xgb.XGBRegressor(**other_params4)
model4.fit(x4, y4)
cols4 = x4.columns.values
```

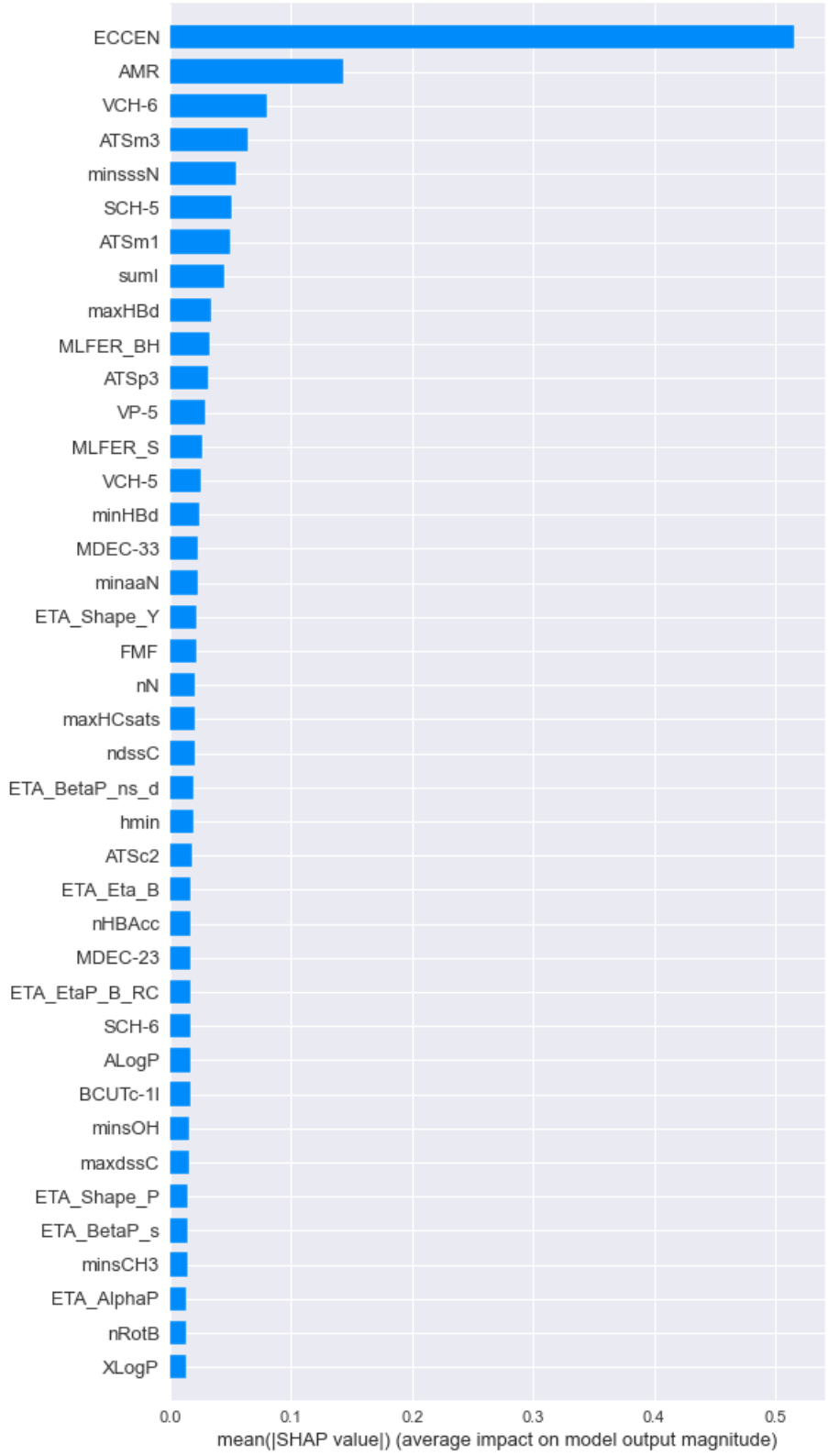
In [2268]:
```
explainer = shap.TreeExplainer(model4)
shap_values4 = explainer.shap_values(x4[cols4])
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

In [2273]:
```
shap.summary_plot(shap_values4, x4[cols4],max_display =40)
```



In [2274]:
```
shap.summary_plot(shap_values4, x4[cols4], plot_type="bar",max_display =40)
```



In [2271]:
```
mean4 = np.mean(np.absolute(shap_values4), axis=0)

colnames_4 = Molecular4_c.iloc[:,1:].columns.values

shap_mean4 = pd.DataFrame(columns= ['name','shap'])
shap_mean4['name']=colnames_clean4
shap_mean4['shap']=mean4
shap_mean4_sort = shap_mean4.sort_values( by = 'shap',ascending = False)
sh4 = shap_mean4.sort_values( by = 'shap',ascending = False)
colname_select4 = sh4.head(40)['name'].values
colnames4_fal=[]
for i in colname_select4:
    if i in colname_select:
        colnames4_fal.append(i)

# shap_mean4_sort.to_excel((r'D:\documents\my work\py\D\shap4.xlsx'))
fiter = Molecular4_c[colnames4_fal]

print(colnames4_fal)
fiter.to_excel((r'D:\documents\my work\py\D\fiter.xlsx'))
```

```
['ECCEN', 'AMR', 'minsssN', 'VCH-5', 'minHBd', 'MDEC-33', 'ETA_Shape_Y', 'nHBAcc', 'MDEC-23', 'ALogP', 'BCUTc-1l', 'minsO
H', 'ETA_BetaP_s', 'ETA_AlphaP', 'XLogP']
```

In [2272]:
```
mean = np.mean(np.absolute(shap_values), axis=0)

colname1 = Molecular_c.iloc[:,1:].columns.values

shap_mean = pd.DataFrame(columns= ['name','shap'])
shap_mean['name']=colname1
shap_mean['shap']=mean
shap_mean.sort_values( by = 'shap',ascending = False)
# shap_mean.to_excel((r'D:\documents\my work\py\D\shap.xlsx'))
sh = shap_mean.sort_values( by = 'shap',ascending = False)
sh.head(20)
colname_select = sh.head(50)['name'].values
print(colname_select)
```

```
['MDEC-23' 'LipoaffinityIndex' 'MLogP' 'minHsOH' 'C1SP2' 'minsssN'
 'minsOH' 'nHBAcc' 'MLFER_A' 'BCUTc-1l' 'C3SP2' 'minHBint5' 'BCUTp-1h'
 'ETA_Shape_Y' 'XLogP' 'minssO' 'MDEO-12' 'VCH-5' 'nHsOH' 'VC-5' 'AMR'
 'MDEC-33' 'minHBint10' 'maxaasC' 'ECCEN' 'ALogP' 'minHssNH' 'maxHBint10'
 'minHBd' 'minsssCH' 'nBase' 'VC-3' 'ETA_AlphaP' 'maxHBa' 'SHBint10'
 'ATSc4' 'mindssC' 'BCUTp-1l' 'nH' 'ETA_BetaP_s' 'BCUTc-1h' 'SC-4'
 'maxHBd' 'VC-4' 'ndssC' 'ETA_dEpsilon_B' 'mindO' 'MLFER_S' 'maxaaCH'
 'ATSc2']
```