古代玻璃文物特征的预测与分析

摘要

本文的研究背景为分析玻璃文物的各个特征之间关系的研究，本文意在通过分析玻璃类型、纹饰、颜色、是否风化以及化学成分之间的关系，使可以在获取部分数据的前提下，对文物进行：风化前化学成分预测、未标记文物分类、对文物进行亚分类等操作。分别建立决策树模型、多元线性回归模型、Logistics 回归模型、聚类模型等多种算法，并对这些策略进行了评价分析。

在第一问中，首先需要分析玻璃类型、纹饰和颜色与是否风化之间的关系，由于特征无法量化，本文则通过计算信息增益，绘制饼状统计图，建立决策树等方法对相关性进行了定性分析。其次，在求解化学成分与玻璃类型、有无风化之间的规律时，本文将类型和风化等特征设置为 0-1 变量，从而构建多元线性回归模型，再利用该模型，通过分析变化水平和归一化处理，最终预测出各个文物风化点在风化前的化学成分含量。

在第二问中，首先建立多种分类器集成的投票集成模型，构建通过化学成分判断类型分类器，采用了 Logistics，adaboost，SVM 等分类模型，最终得到较好的分类效果，第二问主要运用聚类算法，需要选择合适的化学成分作为特征，拉普拉斯分数可以作为计算特征重要性的一个指标，其核心做法在与通过轮廓系数来决策特征数量以及聚类中心个数。

问题三题干可知，我们需要根据附件中给出的八个检测样本的化学成分以及是否风化的属性来预测样本所属的玻璃类型。在前面的分类中我们已经发现了由于样本空间少极易出现过拟合，会对我们模型的预测性能造成影响。所以我们会对所选的预测算法进行对比选择。并且由于数据采集的不确定性，我们也更加希望模型对于数据存在的微小震动仍然有一个较好的稳定性。所以这一问我们主要从这几个角度来进行建模，以及参数调整优化预测性能。

第四问我们先按风化程度与玻璃类型对样本集划分为了 4 类，使用了 pearson 系数相关分析，通过热力图分析出两两成分之间的关系，构建关联组，并通过对关联组之间不断合并得到最终关联关系，同时通过不同类之间的关联组的不同分析类间的化学成分关联的差异性

关键字： 多元线性回归　分类模型　聚类算法　拉普拉斯评价分数

目录

一、 问题重述

在研究古代中西方文化交流的过程中，古代玻璃是一种非常适合考究的工艺制品。在研究时，首先需要了解其化学成分，玻璃中最主要的化学成分便是其原料中的 SiO_2，且为了降低原料的熔点，加入助熔剂的不同也会导致化学成分的差异。例如，我国古代助熔剂常用铅矿石，便导致所制玻璃中含有较多 PbO 和 BaO, 被分类为铅钡玻璃；而草木灰做助熔剂高钾玻璃中则含有较多钾元素。且在制作过程中，加入的稳定剂会氧化为化学成分 CaO。其次，由于储藏条件的不同，玻璃会或多或少的被风化，风化程度越大，其本身的化学成分改变也越多，难以判断其类别。其中，无风化的的玻璃其纹饰及颜色都较为明显，而风化玻璃中严重风化区域常呈线灰黄色，同时风化玻璃中也可能存在一般风化区和未风化区，而无风化文物也可能有浅风化区域。

现有一批已知文物类别、颜色、纹饰等信息古代玻璃制品，以及这些玻璃在不同采样点所收集到的主要化学成分及其占比，其中成分比之和在 85% 到 105% 之间的数据才作为处理数据。同时有一批已知化学成分及占比的未分类文物数据。根据以上信息，解决如下问题：

1. 试分析玻璃风化与其类型、纹饰和颜色之间的关系；根据玻璃类型，分析有无风化的玻璃其化学成分占比的统计规律，并预测风化点在风化前的化学成分数据。
2. 根据附件分析高钾及铅钡两种玻璃的分类规则，选择恰当的化学成分，制定分类方法，对每个类别划分亚类，并展现划分结果及其合理性和敏感性分析
3. 分析表单 3 中未分类文物的化学成分数据，判断其类别，并分析判断结果的敏感性
4. 分析不同类别文物化学成分之间的关联关系，并比较其差异性。

二、 问题分析

通过分析题目可知，需要对玻璃文物的风化程度、类别、颜色、纹饰、化学成分及占比建立数学模型，其中，玻璃文物类别主要为高钾玻璃及铅钡玻璃两类。而风化程度分为未风化和风化两类，其中风化文物存在严重风化、一般风化和无风化三种区域情况，未风化的颜色纹饰清晰，而严重风化玻璃呈线灰黄色。分析化学成分时，可知高钾玻璃可能含有较多的钾元素，而铅钡玻璃可能含有较多 PbO 和 BaO，且风化严重的采样点可能导致化学成分数据失真，难以建立与类别之间的联系，根据题目信息及所给数据，进行以下问题分析：

2.1 问题一的分析

问题一的第一问要求析不同风化程度与类别、颜色和纹饰等特征之间的关系，对于这种难以定量处理的特征数据，我们决定使用概率计算不同特征对风化的信息增益，得

到另外三个特征对决定是否风化的贡献大小。再通过图像可视化分析各个特征的每个属性与是否风化的关系，进一步细化风化与每个单独特征之间的具体联系。最后做出通过三种特征判断是否风化的决策树，得到他们之间总体化的关系。

第二问要求结合玻璃类型，对有无风化与其化学成分之间的关系建立模型，于是我们取不同玻璃类型及不同采样点的风化程度作为自变量，以每种化学成分单独作为一种因变量，构建出 14 种化学成分的多元线性回归模型，并通过该模型，将不同类型玻璃代入各个模型求出风化前的各个化学成分理论值，再通过比例，预测特定文物风化前的化学成分数据。

2.2 问题二的分析

问题二的核心是分类，对于已有的分类需要寻找分类规律，本文对所给分类数据训练出合适的分类模型，通过具体的分类流程来分析其规律。由题目可知，不同的玻璃类别主要是因为助熔剂不同导致其化学成分不同，可知分类的核心特征是化学成分，本文则使用 14 种化学模型作为分类的特征。

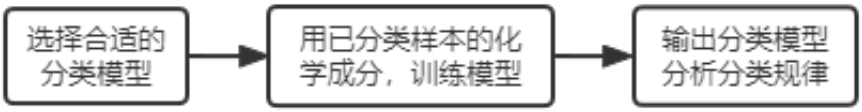

图 **1**　问题二的求解过程

接着要求我们根据化学成分对不同类别进行亚分类，在无监督模型中的聚类算法则是一个合适的选择。在求解模型前，我们还需要根据题目要求，选择合适的化学成分作为特征，拉普拉斯分数可以作为计算特征重要性的一个指标，再可以通过观察轮廓系数，最终选出所需要的化学成分指标和模型参数。在完成亚分类后，我们可以通过轮廓系数描述聚类效果从而分析模型合理性，同时可以通过改变化学成分选择，对聚类效果进行敏感度分析。

2.3 问题三的分析

本文问是在第二问建立的分类模型基础上，对未知类别的数据进行分类预测，为了使总体的分类效果更好，在第二问的模型基础上，对是否风化和所用化学特征进行了细化，使得模型对数据的利用率更高。在分析敏感性时，则通过改变迭代次数和输入数据，来分析对模型的影响大小。

2.4 问题四的分析

这一题要求对不同类别的玻璃文物之间的化学成分进行相关性和差异性分析。经过回带附录文物相关位点的验证，符合我们归纳出的对应类别的化学成分规律，类间的差异性也相对明显。

三、 基本假设

1. 假设表单 1 中颜色空缺值是因风化严重而无法识别，并自定义为“杂色”。
2. 假设未风化文物采样点均为未风化
3. 除第一问以外，将严重风化与风化视为同种类型

四、 符号说明

符号	符号说明	单位
W_0	总样本中未被风化文物数量	个
W_1	总样本中风化文物数量	个
S	总样本文物数量	个
$H(x)$	表示事件 x 的信息熵	bit
$shape$	表示样本文物的纹饰	\
$type$	表示样本文物的类型	\
$color$	表示样本文物的颜色	\
$\Delta H(x)$	表示事件 x 的信息增益	bit
$f_i(x)$	某化学成分第 i 个样本模型预测值	%
f_true_i	某化学成分第 i 个样本真实值	%
ch_{now}	样本某种化学成分当前含量	%
ch_{pre}	样本某种化学成分风化后模型预测值	%
ch_{pro}	样本某种化学成分风化前模型预测值	%
ch_{true}	样本某种化学成分风化前真实预测值	%
m	聚类时所选化学成分特征的数量	个
K	聚类中心的个数	个

五、 数据预处理

本题附件中共有三个表单，其中表单一提供了文物编号、纹饰、类型、颜色和表面风化等特征的数据。通过观察我们发现在颜色一栏中，有四项值是缺失值，且皆是风化

文物，我们推断该类文物为因其风化程度严重，导致无法辨别其颜色，故将其定义为“杂色”, 如表1所示：

表 **1**　对表单一中颜色填补处理

文物编号	纹饰	类型	颜色	表面风化
19	A	铅钡	杂色	风化
40	C	铅钡	杂色	风化
48	A	铅钡	杂色	风化
58	C	铅钡	杂色	风化

同时根据题目信息，对于表单二中的化学成分，缺失项则代表未检测出该成分，故对表单中缺失值均赋值为 0，其次，剔除所有总成分占比之和在 85%~105% 之外的无效数据，得到有效数据。其中剔除值有两项，如表2，所示所剩有效采样点数据为 67 项：

表 **2**　表单二删去无效数据

文物采样点	SiO_2	Na_2O	...	占比之和
17	60.71	2.12	...	71.89
15	61.87	3.21	...	79.47

最后，在对玻璃类型、表面风化这类非数值化特征进行定量处理时，需要先量化标记，将每件文物是否属于某种类型，以及风化程度不同，标记为 0 或 1 的变量，在模型建立中具体介绍。在对所有数据进行预处理后，即可建立模型求解。

六、 模型建立与求解

6.1 问题一决策树及多元线性回归模型

6.1.1 计算信息增益及可视化分析

由题意，需要找出文物表面是否风化与玻璃类型、纹饰、颜色之间的关系，由于所给特征数据均是描述类数据而非数值型数据，且若硬性定值，无参考依据会造成较大误差，故希望通过算概率的方式对数据进行分析，这里采用计算三个特征与是否风化之间的信息增益，以三个特征对判断的贡献程度来展现它们之间的紧密关系顺序。

首先需要计算总体数据集的信息熵，其中设置变量 W_0 表示表单一总样本中未被风化的文物数量，W_1 表示已被风化的文物数量，S 表示样本总量，根据公式(1)所提供的信息熵计算公式, 计算得出总体样本关于是否风化的信息熵 $H(S)$。

$$H(S) = -\sum_{k=0}^{1} \frac{|W_k|}{|S|} \log_2 \frac{|W_k|}{|S|} \tag{1}$$

$$|S| = 58, \quad |W_0| = 24, \quad |W_1| = 34$$

接着需要分别计算三种特征的信息熵以及信息增益，鉴于三种特征：纹饰 shape，类型 type，颜色 color，其计算信息增益的公式及方法均相同，下列只以纹饰的计算方法为例进行展示。定义变量 $shape_1$、$shape_2$、$shape_3$ 分别表示纹饰为 A、B 和 C 的样本，其中 $shape_1^0$ 表示纹饰为 A 且未被风化的样本，$shape_1^1$ 表示纹饰为 A 且被风化的样本，以同样的含义继续定义变量 $shape_2^0$、$shape_2^1$、$shape_3^0$、$shape_3^1$，则可以根据以下公式计算纹饰对于风化的信息增益：

$$H(S|shape) = -\sum_{i=1}^{3} \frac{|shape_i|}{|S|} \sum_{k=0}^{1} \frac{|shape_i^k|}{|shape_i|} \log_2 \frac{|shape_i^k|}{|shape_i|} \tag{2}$$

$$\Delta H(shape) = H(S) - H(S|shape) \tag{3}$$

通过以上步骤可以分别算出三种特征的信息增益为 $\Delta H(shape)$、$\Delta H(type)$ 和 $\Delta H(color)$，具体数据如表3所示，通过对比数据，我们可以得到关系：其中文物的颜色对决定是否风化的影响最大，即两者的相关关系最紧密，接着是纹饰，最后与是否风化相关性最弱的是文物类型。

表 **3**　计算信息增益的结果

	纹饰	类型	颜色
信息增益	0.088550480	0.085674140	0.157160445

在计算得出三种特征的相关性排序后，我们需要进一步分析每个特征与风化之间的具体联系，由于总体样本量较少、特征属性较少、且大部分属性的偏向较为明显，故我们使用 $python$ 将表格数据绘制为饼状统计图，分别对三种属性进行可视化分析。

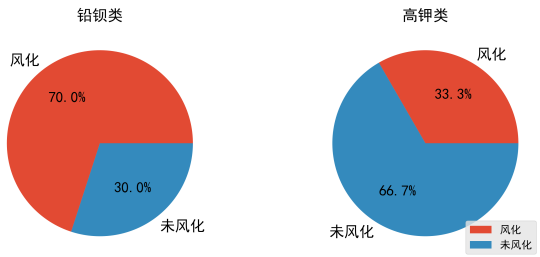

图 **2**　类型与风化之间的关系

由图2可以看出，在所得样本集中，当玻璃类型是铅钡玻璃时，该文物较多被风化，占比为 70%；而当玻璃类型时高钾玻璃时，文物较少风化，风化频率只有 33.3%。即可以得出结论：铅钡类玻璃比高钾类玻璃更容易被风化。

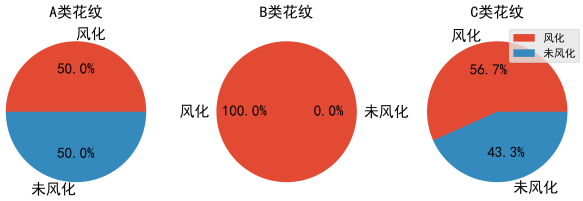

图 **3**　纹饰与风化之间的关系

由图3可以发现，当样本中的玻璃纹饰为 A 类时，文物风化与不风化占比相同，均为 50%，可知 A 类纹饰与是否风化基本无相关性；而样本中 B 类纹饰的文物全部为被风化，可由此得到结论，B 类纹饰的文物极易风化；当样本中文物为 C 类纹饰时，有 56.7% 的被风化，43.3% 的未被风化，则 C 类纹饰与风化相关性较小，且为正相关。

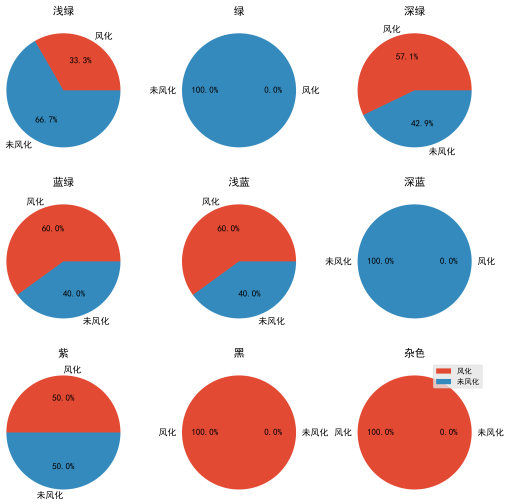

图 **4**　颜色与风化之间的关系

图4反映的是颜色与风化之间的关系，其中绿色和深蓝色的样本文物全为未风化，可知这两种颜色极不易风化；而样本中的黑色和未能辨别颜色的杂色文物均为被风化，杂色由于无法判断其风化前的颜色，故无法分析，而黑色文物则于风化呈限显著的正相关性。其余情况，深绿、蓝绿和浅蓝的风化情况相近，风化占比为 60% 左右，均为较易风化；浅绿色文物则为较不易风化；紫色文物与风化无明显相关性，其两种情况的样本占比均为 50%。

6.1.2 绘制决策树进一步观察相关性

在对三种特征和是否风化完成总体和局部的相关性分析后，我们根据第一步中的信息增益计算公式以及 $ID3$ 算法，对于三种特征判断是否风化建立了如图22所示的决策树模型，其中除了第四层中，由 A 分支出的叶子节点之外，其余的叶节点均为完全分类，只含有一种情况，而由 A 分支的叶节点，由于风化占比高于未风化占比，故以风化作为该叶节点的属性。$ID3$ 算法是通过不断迭代，计算当前信息增益最大特征作为分类标准，以实现从上到下直到叶子节点的分类方法。

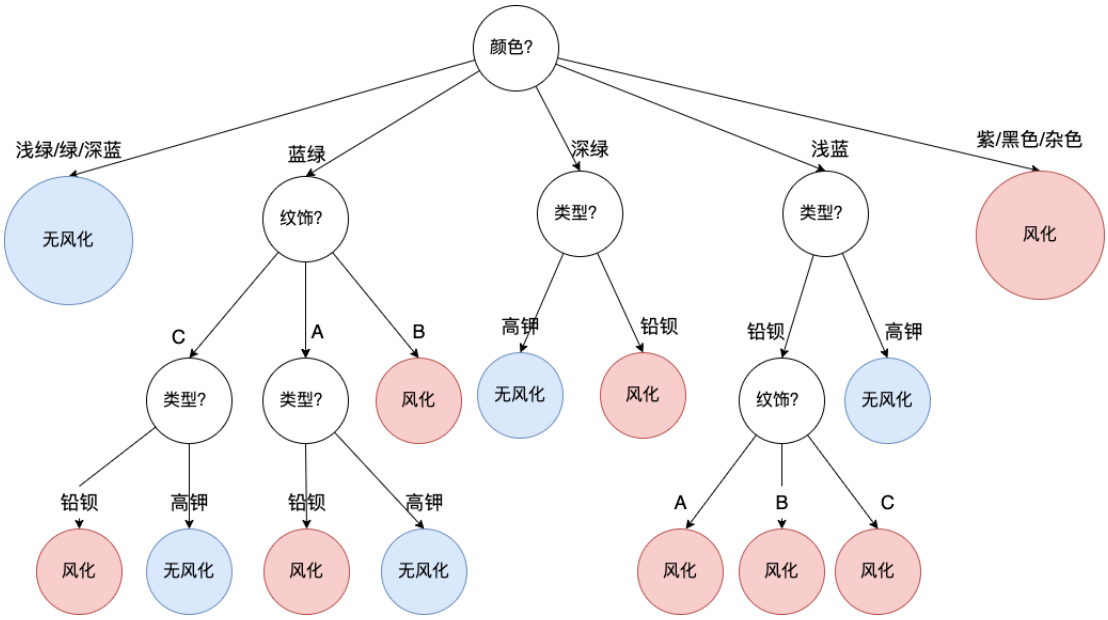

图 **5**　构建分类决策树

6.1.3 多元线性回归模型进行分析和预测

多元线性回归模型是一种以多个自变量来估计和预测目标值的模型，在本题中，要求通过玻璃类型和风化程度对化学成分进行拟合和预测，对于这一类有多个影响因素的问题求解，多元线性回归模型是一个非常合理的解决方案。在求解多元线性回归模型时，我们采用了遗传算法，对每一个特征属性定义为一个 0-1 自变量，以模型的合理性作为约束条件，以求得使标准误差最小的最优解模型。

由上述分析，我们采用多元线性回归模型对每一种化学成分含量进行拟合，得到两个特征与化学成分之间统计规律计算公式，并可以运用该公式，预测风化采样点风化前的化学成分含量。我们设置形如 $X^{\mathrm{T}}=\{X_1,X_2,X_3,X_4,X_5\}$ 的自变量，其中 X_1 表示采样点是否未风化；X_2 表示采样点是否一般风化；X_3 表示玻采样点是否严重风化 X_4 表示玻璃类型是否是高钾；X_5 表示玻璃类型是否是铅钡。并依照遗传算法，建立如下所示的多元线性回归模型，其中 $f_i(x)$ 为某化学成分第 i 个样本模型预测值，f_true_i 为其真实值，以下是单个化学成分回归模型的遗传算法公式。其中，公式(4)是对公式合理性的约束，即第 i 个样本的某化学成分的函数预测值，应当处于与其自变量相同的所有样本化学成分含量之间。而公式(5)的约束，则解释了模型对所有可能样本集的合理性，即当输入训练集中没有的自变量情况时，通过模型计算不会出现目标值为负的异常现象。

$$\begin{aligned} min \qquad & z=\sqrt{\frac{\sum[f_i(x)-f\_true_i]^2}{67}} \\ constraint: \qquad & f(x)=(a,b,c,d,e,u)X \end{aligned}$$

$$\min_{X=X_i}(f\_true_i)\le f_i(x)\le \max_{X=X_i}(f\_true_i) \tag{4}$$

$$\min(a,b,c)+\min(d,e)+u\ge 0 \tag{5}$$

$$X_1+X_2=1 \qquad X_3+X_4+X_5=1$$

$$X_1=0,1 \quad X_2=0,1 \quad X_3=0,1 \quad X_4=0,1 \quad X_5=0,1$$

$$f^{\mathrm{T}}=\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ ... \\ f_{14} \end{bmatrix} \qquad X=\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & ... & x_{1,14} \\ x_{2,1} & x_{2,2} & x_{2,3} & ... & x_{2,14} \\ x_{3,1} & x_{3,2} & x_{3,3} & ... & x_{3,14} \\ x_{4,1} & x_{4,2} & x_{4,3} & ... & x_{4,14} \\ x_{5,1} & x_{5,2} & x_{5,3} & ... & x_{5,14} \\ 1 & 1 & 1 & ... & 1 \end{bmatrix}$$

将表单二中以及处理过的数据代入模型，最终可以求得对于每一类化学成分模型系数如表4所示 (见文件问题 1 模型参数.xlsx)，该回归模型即为玻璃类型、风化程度与各类化学成分含量之间的统计规律。

表 **4**　各个化学成分多元线性回归模型系数

元素	a(未风化)	b(一般风化)	c(严重风化)	d(高钾)	e(铅钡)	u(bias)	平均标准差
二氧化硅	72.58	58.54	33.00	19.36	-13.82	-10.63	14.64
氧化钠	-25.18	-26.26	-26.26	35.12	35.53	-8.86	1.51
氧化钾	-24.45	-25.91	-25.86	39.18	33.61	-7.70	2.52
氧化钙	-24.10	-24.18	-24.13	36.84	35.04	-8.86	2.17
氧化镁	-42.20	-42.39	-42.59	36.92	36.84	6.12	0.63
氧化铝	-22.03	-24.23	-24.94	35.03	34.11	-7.20	2.76
氧化铁	-27.55	-28.04	-28.40	37.44	36.89	-8.40	1.09
氧化铜	-25.51	-25.13	-24.58	36.84	36.55	-9.35	2.21
氧化铅	-10.44	3.97	3.30	17.66	43.29	-7.21	10.26
氧化钡	-22.49	-21.48	-9.59	28.89	37.97	-6.29	6.59
五氧化二磷	-26.45	-23.97	-18.74	34.18	35.61	-7.52	2.88
氧化锶	-25.99	-25.88	-25.49	36.67	36.77	-10.53	0.22
氧化锡	-42.57	-42.58	-42.66	36.79	36.78	5.89	0.33
二氧化硫	-27.07	-27.08	-16.89	36.75	36.91	-9.65	1.65

在得到多元线性回归模型后，需要对各个风化采样点，预测其风化前的数据，考虑到不同的采样点除了玻璃类型之外，还存在较强的特异性，单纯以模型所得结果作为统一预测值可能导致误差较大。而不同类型的文物风化前后，某种化学成分变化可能存在一定比例，我们决定通过不同类型文物风化前后化学物质变化比例，对各个独立的采样点进行预测。以预测第 i 个风化样本的风化前化学成分为例，其中，ch_{now} 为该样本某种化学成分当前含量，ch_{pre} 为该化学成分的风化后模型预测值，ch_{pro} 为该化学成分的风化前模型预测值，ch_{true} 为该化学成分的风化前真实预测值，则通过以下公式求得 ch_{true}：

$$ch_{pre} = (a, b, c, d, e, u)X$$

$$ch_{pro} = (a, b, c, d, e, u)X_{change}$$

$$ch_{true} = ch_{now}\frac{ch_{pro}}{ch_{pre}}$$

其中 X 自变量描述的是该样本化当前的属性，X_{change} 为该样本同种类型但是未风化的自变量情况，即改变 x_1 的值为 1。但通过分析初步计算结果，我们发现由于部分化学成分线性关系不显著、模型存在误差、部分化学物质在风化前后可能不存在或消失，使得量值为 0 等特殊问题，使得预测比例过大，或存在比例中分母为 0 的情况，导致实际预测值出现异常。在上述异常情况下，我们则选用模型预测值作为该化学成分最终的真实预测值。

$$ch_{true} = \begin{cases} ch_{pro} & ch_{pre} = 0, \quad ch_{now}\frac{ch_{pro}}{ch_{pre}} > \max(f\_true) \\ ch_{now}\frac{ch_{pro}}{ch_{pre}} & else \end{cases}$$

最后为了结果合理性，我们将所求得化学成分含量预测值，除以所有化学成分含量预测值之和，作为该化学成分含量最终值，使所预测的化学成分占比之和为 100%。得到部分结果如下表所示 (完整结果见：问题 1 预测风化点未风化点化学成分.xlsx)：

表 **5**　部分样本风化前化学成分预测

文物采样点	08 严重风化点	08 预测风化前	26 严重风化点	26 预测风化前
二氧化硅 (SiO2)	4.61	37.27504991	3.72	26.99941347
氧化钾 (K2O)	0	0	0.4	16.14244912
氧化钙 (CaO)	3.19	4.639697765	3.01	3.929701705
氧化铝 (Al2O3)	1.11	3.958864969	1.18	3.777668102
氧化铜 (CuO)	3.14	2.902269739	3.6	2.986790133
氧化铅 (PbO)	32.45	30.36340809	29.92	25.129942
氧化钡 (BaO)	30.62	18.31189959	35.45	19.02998882
五氧化二磷 (P2O5)	7.56	1.905762389	6.04	1.366715049
氧化锶 (SrO)	0.53	0.254255066	0.62	0.266980485
二氧化硫 (SO2)	15.03	0.388792488	15.95	0.370351113

6.2 问题二多种分类模型及聚类分析

6.2.1 建立分类模型

在本问中我们需要根据化学成分的不同，对高钾玻璃和铅钡玻璃构建分类器模型，由于不确定各个化学成分与不同类别的相关性关系，我们决定采投票式的多种分类器模型集成算法，即选用几种分类器对训练集数据进行分类训练，在对测试集数据进行分类预测时，多种分类器模型同时预测，最终的预测结果以少数服从多数的规则决定。由此在不确定数据可分类型的情况下，也可以获得较高的准确率。

本文主要选择了 Logistic 回归、Adaboost、SVM 三种基本分类器。其中 Logistics 回归模型是一种常用的广义线性回归模型，且在二分类问题上有着显著优势。该分类模型主要通过事件发生几率与概率之间的联系，构建形如 $\log(\frac{p}{1-p}) = \omega x$ 的公式，等式变换后得到模型公式，即 $p = \frac{e^{\omega x}}{1+e^{\omega x}}$，这样则把特征数据转化为概率，通过事件发生概率的大小判断分类结果。

Adaboost 算法则是一种实用的 Boosting 算法，Boosting 算法是一种将多个基本弱分类器集成学习从而得到更高分类精度的算法，Adaboost 则是通过不断迭代分类，利用每次求得的分类难度，不断调整各个弱分类器权重的优化方法。而 SVM 支持向量机模型，则是利用各种核函数，将低维空间不可分的或分类效果差的样本集，映射到高维线性可分空间的分类模型，以 $g(x)$ 表示核函数，其模型公式可表示为：

$$\begin{cases} \min \quad (\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j (g(x_i) * g(x_j)) - \sum_{i=1}^{n} a_j) \\ st. \qquad \sum_{i=1}^{n} a_i y_i = 0 \end{cases} \tag{6}$$

在确立模型之后，首先本文使用了标准差标准化（StandardScaler）对各类化学成分数据进行了归一化处理，公式表现形式为 $x^* = \frac{x-\mu}{\sigma}$, 将处理后 14 种化学成分的数据作为自变量代入模型，将总样本分为训练集：测试集 =7：3，可以得到以下分类效果：

表 **6**　不同分类模型的效果展示

	Logistics	Adaboost	SVM
测试集准确率	1	1	1
训练集准确率	1	1	1

对分类所得的结果进行分析，发现三种模型的训练集和测试集分类准确率均为 1，对三种模型进行 5 折交叉验证，分类准确率依然非常高（99%）。对于这种较为异常的

情况，我们推测可能是因为样本数据量过少，或者不同类别的文物之间的化学成分差异性显著导致的。为了进一步验证所得结果，我们决定对 14 种化学成分与文物类别之间做相关性分析。我们对将文物类别进行标记，其中，高钾记为 0，铅钡记为 1。通过计算各类化学成分与类别的 $pearson$ 相关系数，得到如表7所示的数据，表7中带 * 的数据表示相关性强。得出结论：大部分化学物质与类型都有较显著的线性相关性，可知出现较异常情况的原因为特征性质过于显著。

表 **7**　相关系数表

	二氧化硅	氧化钠	氧化钾	氧化钙	氧化镁	氧化铝	氧化铁
玻璃类型	-0.69**	0.12	-0.72**	-0.34*	-0.096	-0.2	-0.27*
	氧化铜	氧化铅	氧化钡	五氧化二磷	氧化锶	氧化锡	二氧化硫
玻璃类型	-0.055	0.76**	0.53**	0.29*	0.53**	-0.097	0.12

在进行以上分析之后，则可以放心以一种分类模型来对该分类规律进行描述。本文以 Logistics 回归模型所训练出的公式为准，解释 14 种化学物质与玻璃类别之间的分类关系。其中所得的公式和具体参数如下所示：

$$p = \frac{e^{\omega x}}{1 + e^{\omega x}}$$

表 **8**　**Logistics** 回归模型参数表

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	二氧化硅	氧化钠	氧化钾	氧化钙	氧化镁	氧化铝	氧化铁
ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
2.71	-1.060	0.356	-1.065	-0.234	0.194	0.025	-0.105
	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
	氧化铜	氧化铅	氧化钡	五氧化二磷	氧化锶	氧化锡	二氧化硫
	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}
	-0.505	1.179	0.797	-0.017	0.573	-0.051	0.119

其中，x_i 代表回归模型的第 i 个自变量，而 ω_i 则表示自变量对应的系数参数，该参数绝对值越大，说明 x_i 变量对分类结果的影响越大，由表可看出二氧化硅、氧化钾、氧

化铅、氧化钡等四种化学含量对分类结果的影响最大，说明不同类型的文物这些化学成分特征差异较为明显。

6.2.2 聚类模型构建

由题目分析可知，这道题主要运用无监督算法中的聚类模型。按照 2.1 所给出的已知两种玻璃类型将样本集划分为两类，需要根据合适的化学成分特征给出亚类的划分个数，一方面我们需要评估各个化学成分子集对最终聚类效果的影响，选取其中的较优解，另一方面在给定的特征（化学成分）的前提下通过无监督聚类的方法得到聚类簇的个数。

后者的解决方案已经相对成熟，可以根据轮廓系数等方式得出给定确定化学成分输入情况下的聚类个数的选择。而前者是一个无监督特征选择改善聚类效果的问题，根据文献 [1]，无监督特征选择分为三种模型：

1. $Filter$ 方法：只使用数据内在属性来判定特征选择的先后顺序，不使用聚类等其他辅助方法来判断，优点是快速可行。
2. $Wrapper$ 方法：分别对不同特征使用聚类算法评估子集特征，求解空间大，速度较慢，一般需要用到启发式算法。
3. $Hybrid$ 方法：综合上面两种算法，在计算速度和模型效果做折中。

考虑到本题中给出的是一个较小的数据集，且聚类效果明显，因此并不需要在 2^{14} 个空间中遍历选取最优的特征子集，只需要得到每个特征量化给出在数据分布中的贡献占对聚类效果的影响，因此本文使用基于拉普拉斯评分的特征选择算法，详见文献 [2]。即通过对样本即特征打分的方式，量化特征聚类贡献，最终选取最低的 k 个特征作为最后的特征子集。算法流程如下：

1. 首先构建连通图，对于无监督学习即构建最近邻图 G，其中 x_i 是否与 x_j 相邻, 根据 KNN 中样本 i 与样本 j 的类型值是否相同来判定:

$$G_{m\times m}:(G_{ij})=\begin{cases}1, & if\ \ x_i\ \ close\ \ to\ \ x_j\\ 0, & otherwise\end{cases}$$

2. 计算权重矩阵:

$$S_{m\times m}:(S_{ij})=\begin{cases}e^{-\frac{\|x_i-x_j\|^2}{t}}, & if\ \ G_{i,j}=1\\ 0, & otherwise\end{cases}$$

3. 计算 Laplacian Graph，其中对于第 r 个特征，定义 $f_r=[f_{r1},f_{r2},...,f_{rm}]^{\mathrm{T}}, L=D-S$，矩阵 L 被称为图拉普拉斯。

$$\tilde{f}_r = f_r - \frac{f_r^{\mathrm{T}} D1}{1^{\mathrm{T}} D1} 1$$

4. 计算第 r 个特征拉普拉斯分数，L_r 为第 r 个特征的 Laplacian Graph，$f_{ri} - f_{rj}$ 表示第 i 个样本和第 j 个样本的第 r 个特征的差值，S_{ij} 为权重矩阵中对应的值：

$$L_r = \frac{\tilde{f}_r^{\mathrm{T}} L \tilde{f}_r}{\tilde{f}_r^{\mathrm{T}} L \tilde{f}_r}$$

对于一个好的特征，$S_{i,j}$ 越大，我们倾向于 $|f_{r,i} - f_{r,j}|$ 越小的特征，因为不希望该特征在同亚类间变化太大，同时 $Var(f_r)$ 越大越好，因为这样越容易区分，都是合理的算法直觉。

6.2.3 模型实现与结果

我们先将有效文物样本根据玻璃类型分为‘高钾’和‘铅钡’两个子集，根据拉普拉斯评分法分别两个子集对化学成分的拉普拉斯评分进行计算，并从小到大进行排序，分别对应着影响聚类从大到小的各个化学成分的排名，如下图所示，由于篇幅原因只显示前四位：

表 **9**　拉普拉斯评分排序

高钾类	评分	铅钡类	评分
氧化锡	0	氧化锡	2.9E-06
氧化纳	8.79E-07	二氧化硅	4.47E-06
氧化锶	1.97E-06	氧化钠	1.65E-05
氧化铅	6.31E-06	氧化铅	2.15E-05

后我们参照评分排序，用轮廓系数分析值 m，使得前 m 项化学成分为特征时可以得出最好的聚类效果。轮廓系数表征了聚类效果的好坏，因此我们以其最大值的标准，并将前 m 个化学成分样本点以此聚类后最高轮廓系数表征当前选取这 m 个化学成分子集特征在聚类时效果的好坏，通过算法可以绘制出 m 与轮廓系数的关系如下图所示：

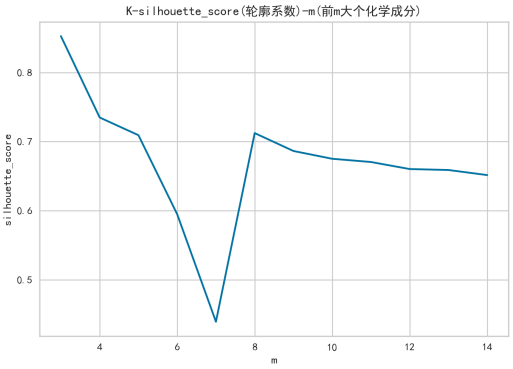

图 **6**　高钾类轮廓系数与 **m** 关系图

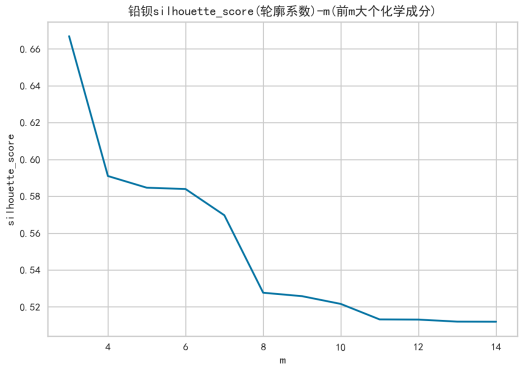

图 **7**　铅钡类轮廓系数与 **m** 关系图

从而可以确定当选取 m=3，也就是氧化锡，氧化钠，氧化锶为分类标准的时候，聚类高钾玻璃类型的文物是聚类效果最好的，其轮廓系数达到了 0.89。同时通过绘制高钾和铅钡两种类型 m=3 时，关于聚类个数 K 的轮廓系数表，如 6.2.4 中所示，当 K=2 时，两个聚类模型的效果最好。得到如下图所示的聚类效果：

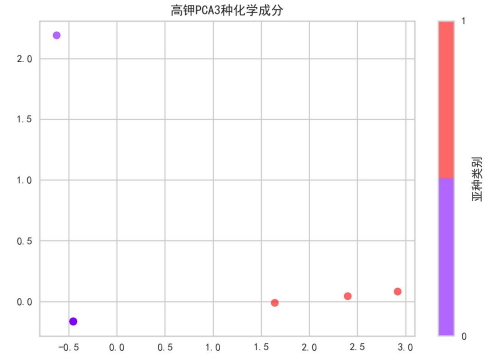

图 **8**　高钾类聚类分析图

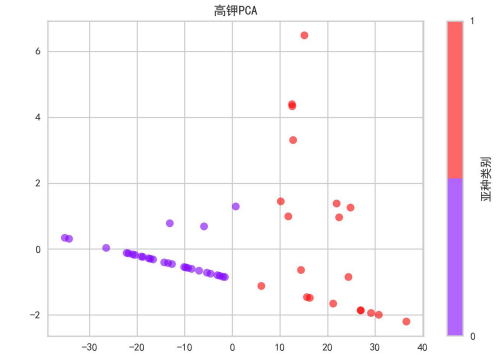

图 **9**　铅钡类聚类分析图

最终分类效果如下表所示：

表 **10**　两种类型子类情况

类型	类别一编号	类别二编号
高钾	1，3，4，5，6，7，9 10，12，16，19，20，17	13，14，15
铅钡	2，8，11，19，20，24，26，30 34，36，38，39，40，41，43，49 49，50，51，52，54，56，57，58	23，25，28，29，31，32，33 33，35，37，42，44，45，46 47，48，49，50，53，55

6.2.4 模型合理性和敏感性解释

为了分析模型合理性，我们将从所选参数 m 和 K 入手进行分析，首先需要判断，运用所选 m=3 个化学成分作为特征进行聚类是否合理，其次需要检验 K 个聚类中心的优越性，于是我们做出了所选三个化学成分和 14 个化学成分的轮廓系数表如下所示：

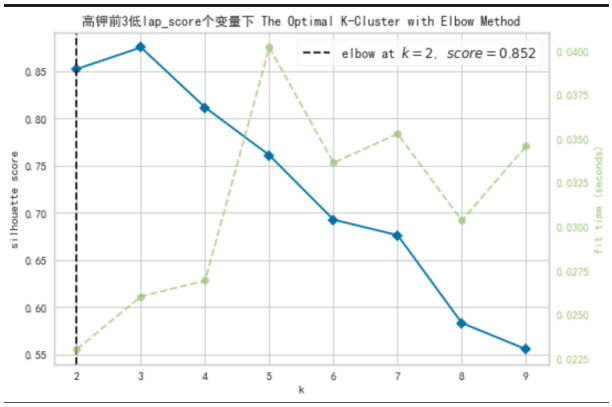

图 **10**　**3** 成分高钾类轮廓系数

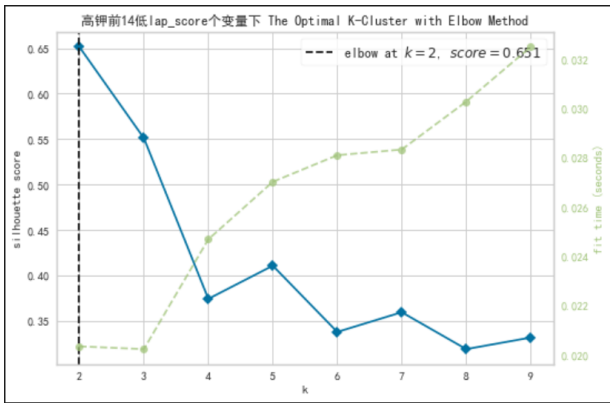

图 **11**　**14** 成分高钾类轮廓系数

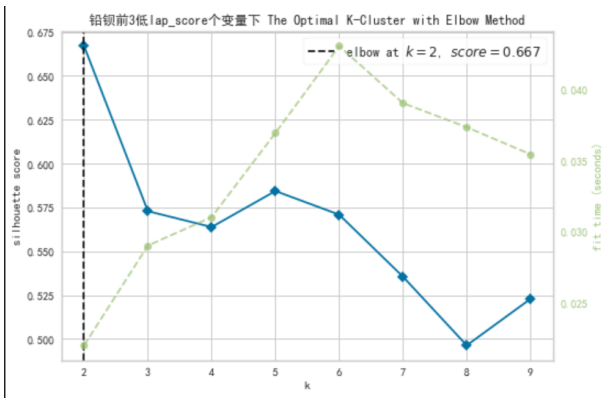

图 **12**　**3** 成分铅钡类轮廓系数

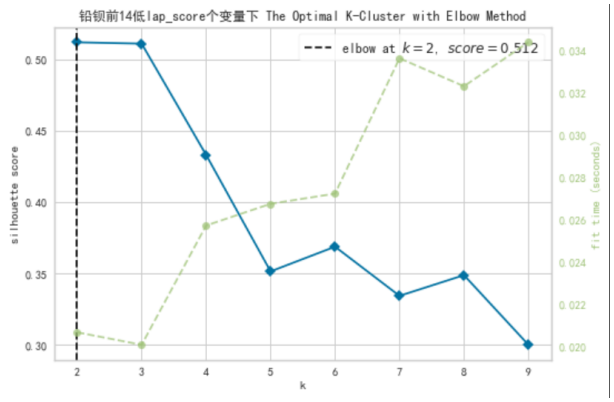

图 **13**　**14** 成分铅钡类轮廓系数

可以看出，本文模型中所选三种化学成分的聚类效果（轮廓系数）明显高于将 14 种成分均作为特征进行分类的效果，对于两种玻璃类型均有这种结论。故本文中所建立模型起到了增强目的性，简化操作的目的。同时可以得出，当 K=2 时，聚类的轮廓系数最高，故认为选择两个聚类中心是对模型合理的决定。

接下来对模型进行敏感性分析，通过调整不同化学成分子集的个数 m，我们可以通过 PCA 可视化降维观察聚类的变化情况，以下只选取调参过程中较有代表意义的过程图进行展示。

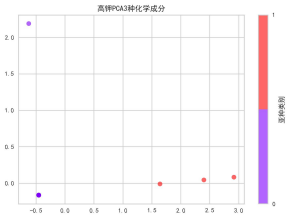

图 **14**　钾 **K=3**

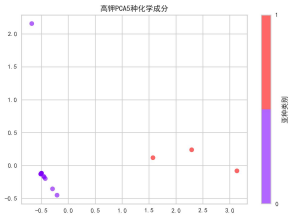

图 **15**　钾 **K=5**

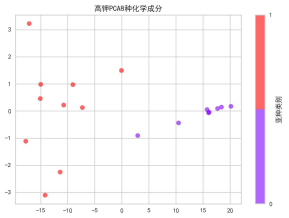

图 **16**　钾 **K=8**

对于高钾玻璃类的文物而言，当 m 从 3 变化到 5 的过程中虽然亚种 a 中 0 号亚类的聚类效果逐渐变差，但幅度是相较缓慢的，因此我们可以认为在该范围内的选取的分类方式都是合理的，算法是稳定的，但一旦 m 到达了 6-8 的区间，类别 1 变得相当离散，出现了较大的偏差，因此，该聚类算法模型在此 m>6 时相当敏感，效果变差。

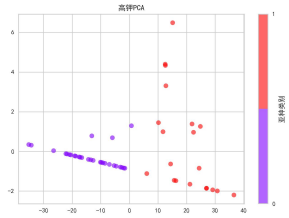

图 **17**　铅钡 **K=3**

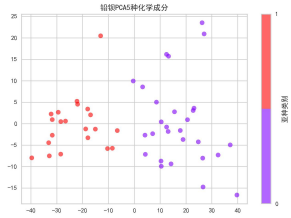

图 **18**　铅钡 **K=5**

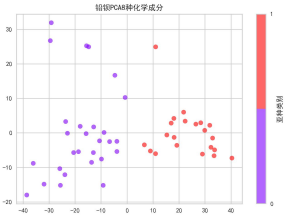

图 **19**　铅钡 **K=8**

对于铅钡类的文物，则是不同的结果，虽然直觉上 m=3 的聚类并不像 m>4 情况下的球状聚类效果那么理想，但是仍然可以注意到有很多点在该低维平面上是相互叠加，相互遮挡的，因此 m=3 的轮廓系数高于后者是合理的，（合理性，如果没写可以补充到上面），当 m 变大时，我们可以发现一直到 m=8，聚类效果依然显著，因此我们可以认为该算法在铅钡类子类聚类上是相当稳定的，敏感性不高。

6.3 问题三优化分类模型预测文物类别

6.3.1 建立更细化的分类模型

第三问需要我们根据附件中 8 个未分类的文物进行类别区分，所给特征数据为是否风化以及 14 种化学成分含量占比。在 2.1 问题中，我们根据已分类文物数据，创建并训练出了一套效果较好的分类模型，但当时并未将是否风化作为决策因素之一，考虑到个别文物会因为风化导致化学成分发生较大变化，以及 2.1 中的分类模型对未分类数据可能存在过拟合的现象，为了能够更加贴合地预测分类结果，我们决定进一步细化分类模型。延用 2.1 中的分类模型，我们选择 Logistics 回归模型和 SVM 支持向量机模型分别对已分类数据进行训练，其中我们将所有已分类数据按照是否风化继续分为两个集合，

对这两个集合再分别进行分类训练，从而得到两个更加细化的分类器。将已分类数据代入模型，按照 5 折交叉验证训练两种模型最终可以得到以下分类效果：

表 **11**　两种分类模型交叉验证效果展示

	Logistics	SVM
未风化集准确率	0.97	1
风化集准确率	1	1

通过观察，考虑到 SVM 分类模型可能对未知数据发生过拟合现象，本文选择较为合理的 Logistics 回归模型作为分类模型。同时我们考虑到部分化学成分对于结果的影响性可能不高，可以删去一些不必要的特征以简化模型，也可使分类模型有更广的适用性。本文将数据集按照 1：4 的比例随机分割出测试集和训练集，对于训练出的 Logistics 回归模型，如下表所示，筛去参数绝对值较小的七项特征，即选择二氧化硅、氧化钾、氧化钙、氧化铅、氧化钡、氧化铁、氧化锶作为模型自变量。

表 **12**　**Logistics** 回归模型未风化参数表

1	二氧化硅	氧化钠	氧化钾	氧化钙	氧化镁	氧化铝	氧化铁
ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
1.01	-0.236	0.094	-0.364	-0.334	-0.083	-0.027	-0.208
	氧化铜	氧化铅	氧化钡	五氧化二磷	氧化锶	氧化锡	二氧化硫
	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}
	-0.186	0.369	0.250	-0.002	0.191	0.016	0.028

表 **13**　**Logistics** 回归模型风化集参数表

1	二氧化硅	氧化钠	氧化钾	氧化钙	氧化镁	氧化铝	氧化铁
ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
1.76	-0.415	0.113	-0.149	0.137	0.072	0.093	0.069
	氧化铜	氧化铅	氧化钡	五氧化二磷	氧化锶	氧化锡	二氧化硫
	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}
	0.025	0.340	0.207	0.156	0.235	0.078	0.047

6.3.2 训练模型预测分类情况

在选择合适的分类模型集自变量之后，我们需要计算出最终的分类模型，且根据所给数据，按照题目要求预测未分类文物的玻璃类型。首先根据上面所选择的六个化学成分特特征，对未风化和已风化且已分类的数据集按照 1：4 的比例划分，进行模型训练，得到如表14和表15所示的模型参数。

表 **14** **Logistics** 回归模型未风化参数表 **2**

1	二氧化硅	氧化钾	氧化钙	氧化铁	氧化铅	氧化钡	氧化锶
ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
1.00	-0.209	-0.404	-0.320	-0.212	0.407	0.247	0.218

表 **15** **Logistics** 回归模型风化集参数表 **2**

1	二氧化硅	氧化钾	氧化钙	氧化铁	氧化铅	氧化钡	氧化锶
ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
1.72	-0.439	-0.158	0.163	0.094	0.399	0.217	0.253

再将未风化和已风化且未分类的数据集 $X_{pro}^{\mathrm{T}} = 1, x_{pro}^{1}, x_{pro}^{2}, ..., x_{pro}^{6}$ 分别代入模型，预测其类型。最终得到分类结果（问题 3 预测结果.elsx）：

表 **16** 对文物的分类结果

文物编号	A1	A2	A3	A4	A5	A6	A7	A8
玻璃类型	高钾	铅钡	铅钡	铅钡	高钾	高钾	高钾	铅钡

其中，由原始数据可知，A1，A5，A6，A7 显然二氧化硅的含量较高，而 A2，A3，A4，A8 的氧化铅和氧化钡的含量较高。这些显著特征和我们从题目中了解到两种玻璃的制作过程，以及附件中数值分布情况来看吻合程度较好。并且在我们训练出模型的验证准确率足够高的情况下，我们认为我们的预测结果正确率较高。

6.3.3 敏感性分析

所用数据解释：由于 A1-A8 的数据是我们通过自己的模型预测得到的，结果仍然可能会有误差性，所以我们在判断我们的模型的敏感性的时候，仍然采用的是附件里面的数据作为训练集，验证集，测试集。

1 lr 回归模型训练的最大迭代次数

我们的模型是一个 lr 回归模型，我们在训练的时候，我们可以设置最大迭代次数，作为我们结果收敛的一个限制。因此我们分别设置 1-20(步长为 1)，然后 40，60，100，200 次为最大迭代次数，通过 k=5，交叉训练验证得到的平均精度和标准差变化图如下：

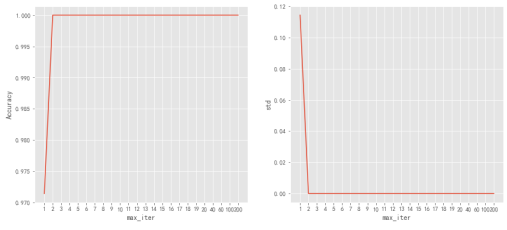

图 **20**　未风化 **LR1**

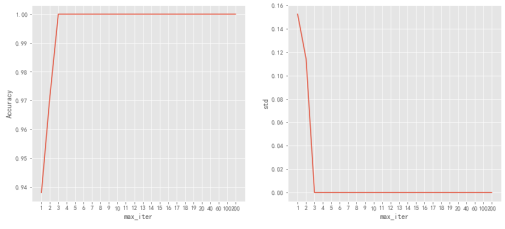

图 **21**　风化 **LR1**

在最早的几次迭代中，我们的模型的精度是在提高的，是因为此时还没有收敛。但是当迭代到 20 次之后，通过训练的反馈得知已经开始收敛，并且最后稳定在 1。我们分析这是由于我们训练所用到数据集的样本空间较小，会不可避免地出现一定的过拟合。就输出结果而言，我们可以看出模型在输出效果一开始会很敏感地提升，但是最大迭代次数提升到 10 以后，输出效果就极度不敏感（非常稳定）。

2　模型预测输入的数据发生微小抖动

我们的预测模型的目的是通过检测样本的化学成分含量，来判断样本的玻璃类型。但是在实际的检测过程中，我们的检测仪器并不是完美的，所以我们的检测结果会有一定的误差。或者我们的数据在计算机处理的过程中，发生一些微小的扰动。因此我们在评估模型的时候，我们需要考虑到这种情况。

针对我们预测 A1-A8 样本的模型 LR1 和 LR2。我们从附件中的原始数据进行随机抽取，按照步长为 1 % 依次选择 0 % ～ 10% 的扰动范围 (最大 95% ～ 105%，共 11 次) 对数据特征进行扰动，然后对扰动后的数据进行预测，得到的预测结果准确度变化如下：

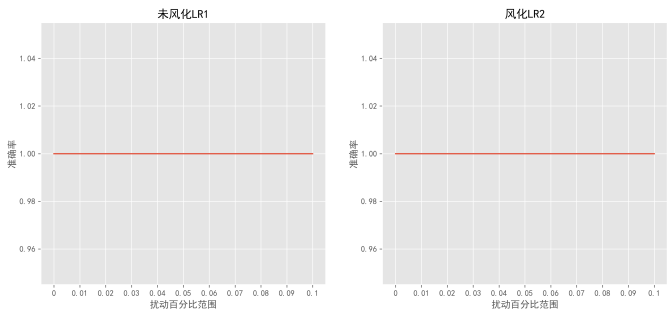

图 **22**　结果准确度变化图

可以看到我们的模型对于即使有扰动的情况下，也能保持较高的预测准确率。我们分析这是由于我们构建 LR1 和 LR2 模型的时候，我们选择了最重要的七个特征作为我们的输入，很大程度的规避了不重要特征对我们的数据的影响。因此即使我们的数据发生了一定的扰动，对我们模型预测结果影响有限。

6.4 问题四皮尔森相关系数分析

为找出不同类别的化学成分的关联关系，我们根据不同类别对样本集进行划分，分为风化高钾，未风化高钾，风化铅钡，未风化铅钡，四个样本集，对其矩阵的皮尔森系数相关性进行了分析，根据系数的大小关系，来判定不同化学物质之间两两之间的关联关系，系数绝对值越大，两者的联系越强，其中正数值越接近 1 代表正相关性越强，负数越接近-1 代表负相关性越强。分别根据两两之间的关系，构建强相关性的联系元组，并通过不断合并邻近子集，最终得到极大无关组，作为最终分析的单元。现分析不同类别的关联性和差异性结果如下：

对于未风化高钾部分：

1. 二氧化硅，氧化钙，氧化钾：二氧化硅与氧化钙，氧化钾都有很强的正负相关性，即二氧化硅高的未风化高钾文物往往后两种化学成分会相应的变低
2. 五氧化二磷，氧化铁，氧化锶，氧化铝：五氧化二磷与氧化铁, 氧化锶, 氧化铝有很强的正相关性，即五氧化二磷高的未风化高钾文物往往后三种化学成分也会相应的变高。

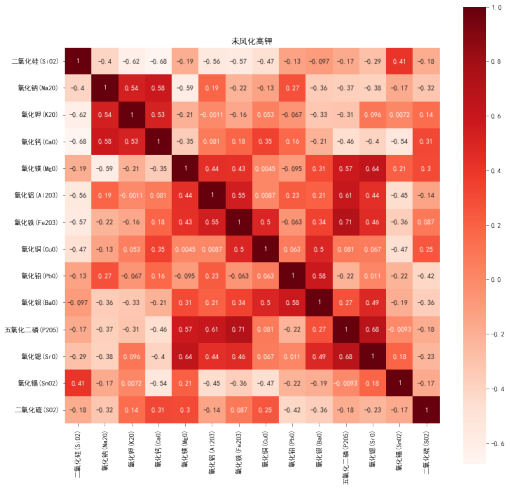

图 **23**　未风化高钾

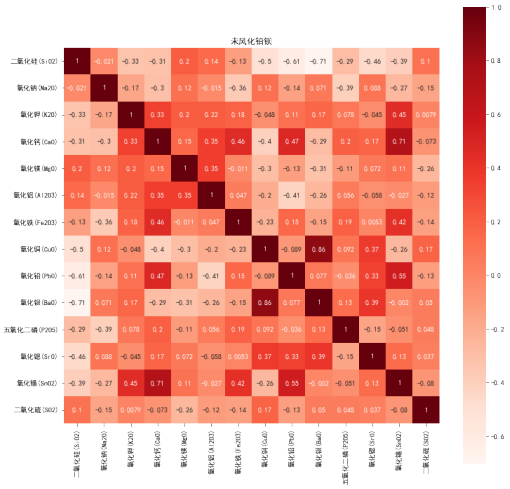

图 **24**　未风化铅钡

对于未风化铅钡部分（化简后得到三组极大关联组）：

1. 二氧化硅，氧化，氧化钡，氧化锶组内四个都有很强的负相关性，即二氧化硅高的未风化铅钡文物往往后四种化学成分会相应的变低

2. 氧化铜，氧化铅两者具有很强的正相关性
3. 氧化锡，氧化钾，氧化钙，氧化铁，氧化铅组内五个具有很强的正相关性，即氧化锡高的未风化铅钡文物往往后四种化学成分会相应的变低

对于风化高钾，由于该类别样本点个数远远少于特征个数，绝大多数的相关性系数为 0，无法分析出有效的数据，但是我们仍然可以从已有的数据分析出器大致规律：

1. 氧化铁 (Fe2O3)，氧化铜 (CuO) 之间存在较强的正相关性关系

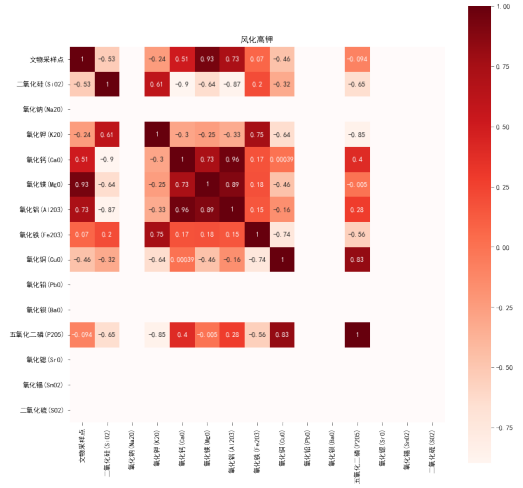

图 **25**　风化高钾

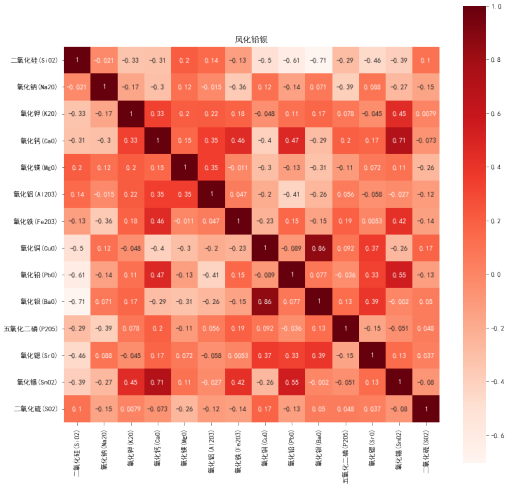

图 **26**　风化铅钡

对于风化铅钡类文物，有以下结果：

1. 二氧化硅，氧化铝，氧化锡，二氧化硫，氧化锶组内二氧化硅含量变高往往代表着氧化铝，氧化锡含量的变高，同时也代表着二氧化硫，氧化锶较低
2. 氧化铁，氧化镁，氧化铝组内是正相关的

七、 模型的评价

7.1 模型的优点

1. 采用了 Laplacian 评价方法，合理的评价了各个元素对于对于聚类最终效果的贡献程度，给出了量化的结果，同时选择部分较少化学成分聚类的结果比 14 种化学元素全部用来聚类效果更为明显。
2. 模型假设的通过风化文物的位点来判定文物本身所属亚类，最终只有 49 号文物的两个位点出现不同的亚类结果，说明模型本身很稳定，并不会大幅度受点位风化程度影响聚类过程进而最终实现只通过分析化学元素得出所属亚类。

7.2 模型的缺点

1. 在第一问的处理过程中，未明确分析类别、颜色和纹饰三种特征与是否风化的具体相关系数，只是比较三者的信息增益，若有可以具体看出相关性大或小数据会更好。
2. 模型将风化文物的未风化点与未风化文物采样点归为一类，且没考虑未风化文物可能存在轻微风化区域，可能导致误差。
3. 该方法本质上使用不断提升阈值计算选取化学成分个数对于最终结果的影响，并没有完全对原始求解空间进行求解，得出的并不一定是最优解。
4. 该方法本质上使用不断提升阈值计算选取化学成分个数对于最终结果的影响，并没有完全对原始求解空间进行求解，得出的并不一定是最优解。

参考文献

[1] 侯鲜婷，露天石灰岩文物表面防护材料及工艺探索，西北大学 [D].2015

[2] Saúl Solorio-Fernández et al.“A review of unsupervised feature selection methods”*Artificial Intelligence Review* (2020): n. pag.

[3] He, Xiaofei et al.“Laplacian Score for Feature Selection.”*NIPS* (2005).

附录 A　附录 1：支撑材料文件列表

表 17　附录 1：支撑材料文件列表

文件列表名
问题 1 模型参数.xlsx
问题 1 预测风化点未风化点化学成分.xlsx
问题 3 预测结果.xlsx
问题一相关代码
$Q1\_1.py$
$Q1\_2.py$
$Q1\_3.py$
问题二相关代码
$Q2\_1.py$
$Q2\_2.py$
问题三相关代码
$Q3\_1.py$
$Q3\_2.py$
$Q3\_3.py$
问题四相关代码
$Q4.py$

附录 B　1.1

```
# %%
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

# %%
df2=pd.read_excel("附件.xlsx",sheet_name=1)
df2_valid=df2[(df2.iloc[:,1:].sum(axis=1)<105)&(df2.iloc[:,1:].sum(axis=1)>85)]
d=df2[(df2.iloc[:,1:].sum(axis=1)<105)&(df2.iloc[:,1:].sum(axis=1)>85)]
df2.reset_index(drop=True,inplace=True)77
df2_valid.to_excel("sheet2_valid.xlsx",index=False)
```

```
df1=pd.read_excel("附件.xlsx",sheet_name=0)
df1=df1.fillna("杂色")

# %%
colors=["浅绿","绿",'深绿','蓝绿','浅蓝','深蓝','紫','黑','杂色']
shapes=['A','B','C']
types=['铅钡','高钾']
def diff_color(color):
i=1
for c in colors:
if color == c:
return i
i=i+1
def diff_shape(t):
if t=='A':
return 1
elif t=='B':
return 2
else:
return 3
def diff_type(s):
if s=="铅钡":
return 1
else:
return 2

def diff(f):
if f == '风化':
return 1
else:
return 0

# %%
objects=[]
for a in df1.iterrows():
a=a[1].tolist()
objects.append((diff_shape(a[1]),diff_type(a[2]),diff_color(a[3]),diff(a[4])))

# %%
def show_shape(objects):
w=[0,0,0]
uw=[0,0,0]
for o in objects:
if(o[3]==1):
w[o[0]-1]=w[o[0]-1]+1
else:
uw[o[0]-1]=uw[o[0]-1]+1
```

```
sum=[w[0]+uw[0],w[1]+uw[1],w[2]+uw[2]]
print(sum)

wp=np.true_divide(w,sum)
uwp=np.true_divide(uw,sum)
uwp=uwp
plt.style.use('ggplot')
fig,axes=plt.subplots(1,3,figsize=(10,4))
axes[0].pie([wp[0],uwp[0]],labels=['风化','未风化'],autopct='%1.1f%%',textprops={'fontsize':
    14})
axes[0].set_title('A类花纹')
axes[1].pie([wp[1],uwp[1]],labels=['风化','未风化'],autopct='%1.1f%%',textprops={'fontsize':
    14})
axes[1].set_title('B类花纹')
axes[2].pie([wp[2],uwp[2]],labels=['风化','未风化'],autopct='%1.1f%%',textprops={'fontsize':
    14})
axes[2].set_title('C类花纹')
plt.legend()
plt.savefig("花纹类别pie.png",dpi=400)
plt.show()



# %%
show_shape(objects)

# %%
from tkinter import font


def show_type(objects):
w=[0,0]
uw=[0,0]
for o in objects:
if(o[3]==1):
w[o[1]-1]=w[o[1]-1]+1
else:
uw[o[1]-1]=uw[o[1]-1]+1

sum=[w[0]+uw[0],w[1]+uw[1]]
# print(sum)

wp=np.true_divide(w,sum)
uwp=np.true_divide(uw,sum)
plt.style.use('ggplot')
fig,axes=plt.subplots(1,2,figsize=(10,4))
```

```
axes[0].pie([wp[0],uwp[0]],labels=['风化','未风化'],autopct='%1.1f%%',textprops={'fontsize':
    14})
axes[0].set_title('铅钡类')
axes[1].pie([wp[1],uwp[1]],labels=['风化','未风化'],autopct='%1.1f%%',textprops={'fontsize':
    14})
axes[1].set_title('高钾类')
plt.legend(loc='lower right')
plt.savefig("玻璃类别pie.png",dpi=400)
plt.show()

# %%
show_type(objects)

# %%
def show_color(objects):
w=[0]*9
uw=[0]*9
for o in objects:
if(o[3]==1):
w[o[2]-1]=w[o[2]-1]+1
else:
uw[o[2]-1]=uw[o[2]-1]+1

sum=[w[i]+uw[i] for i in range(9)]

wp=np.true_divide(w,sum)
uwp=np.true_divide(uw,sum)
plt.style.use('ggplot')
fig,axes=plt.subplots(3,3,figsize=(12,12))
for i in range(3):
for j in range(3):
axes[i,j].pie([wp[i*3+j],uwp[i*3+j]],labels=['风化','未风化'],autopct='%1.1f%%',textprops={'fontsize':
    12})
axes[i,j].set_title(colors[i*3+j])
plt.legend(loc='upper right')
plt.savefig("颜色类别pie.png",dpi=400)
plt.show()

# %%
show_color(objects)

# %%
#计算信息熵
def cal_information_entropy(data):
data_label = data.iloc[:,-1]
label_class =data_label.value_counts()
```

```
    Ent = 0
    for k in label_class.keys():
    p_k = label_class[k]/len(data_label)
    Ent += -p_k*np.log2(p_k)
    return Ent

    def cal_information_gain(data, a):
    Ent = cal_information_entropy(data)
    feature_class = data[a].value_counts()
    gain = 0
    for v in feature_class.keys():
    weight = feature_class[v]/data.shape[0]
    Ent_v = cal_information_entropy(data.loc[data[a] == v])
    gain += weight*Ent_v

    print(f'特征{a}的信息增益为{Ent - gain}')
    return Ent - gain

    def cal_information_gain_continuous(data, a):
    n = len(data)
    data_a_value = sorted(data[a].values)
    Ent = cal_information_entropy(data)
    select_points = []
    for i in range(n-1):
    val = (data_a_value[i] + data_a_value[i+1]) / 2
    data_left = data.loc[data[a]<val]
    data_right = data.loc[data[a]>val]
    ent_left = cal_information_entropy(data_left)
    ent_right = cal_information_entropy(data_right)
    result = Ent - len(data_left)/n * ent_left - len(data_right)/n * ent_right
    select_points.append([val, result])
    select_points.sort(key = lambda x : x[1], reverse= True)
    return select_points[0][0], select_points[0][1]

    def get_most_label(data):
    data_label = data.iloc[:,-1]
    label_sort = data_label.value_counts(sort=True)
    return label_sort.keys()[0]

    def get_best_feature(data):
    features = data.columns[:-1]
    res = {}
    for a in features:
    if a in continuous_features:
    temp_val, temp = cal_information_gain_continuous(data, a)
    res[a] = [temp_val, temp]
    else:
```

```
        temp = cal_information_gain(data, a)
        res[a] = [-1, temp] #离散值没有划分点，用-1代替

        res = sorted(res.items(),key=lambda x:x[1][1],reverse=True)
        print(f'最佳划分特征为{res[0][0]}')
        return res[0][0],res[0][1][0]

        def drop_exist_feature(data, best_feature):
        attr = pd.unique(data[best_feature])
        new_data = [(nd, data[data[best_feature] == nd]) for nd in attr]
        new_data = [(n[0], n[1].drop([best_feature], axis=1)) for n in new_data]
        return new_data

        def create_tree(data):
        data_label = data.iloc[:,-1]
        if len(data_label.value_counts()) == 1:
        return data_label.values[0]
        if all(len(data[i].value_counts()) == 1 for i in data.iloc[:,:-1].columns):
        return get_most_label(data)
        best_feature, best_feature_val = get_best_feature(data)
        if best_feature in continuous_features:
        node_name = best_feature + '<' + str(best_feature_val)
        Tree = {node_name:{}}
        Tree[node_name]['是'] = create_tree(data.loc[data[best_feature] < best_feature_val])
        Tree[node_name]['否'] = create_tree(data.loc[data[best_feature] > best_feature_val])
        else:
        Tree = {best_feature:{}}
        exist_vals = pd.unique(data[best_feature])
        if len(exist_vals) != len(column_count[best_feature]):
        no_exist_attr = set(column_count[best_feature]) - set(exist_vals)
        for no_feat in no_exist_attr:
        Tree[best_feature][no_feat] = get_most_label(data)
        for item in drop_exist_feature(data, best_feature):
        print(f'当前特征为{best_feature},特征值为{item[0]}')
        Tree[best_feature][item[0]] = create_tree(item[1])
        return Tree

        #根据创建的决策树进行分类
        def predict(Tree , test_data):
        first_feature = list(Tree.keys())[0]
        if (feature_name:= first_feature.split('<')[0]) in continuous_features:
        second_dict = Tree[first_feature]
        val = float(first_feature.split('<')[-1])
        input_first = test_data.get(feature_name)
        if input_first < val:
        input_value = second_dict['是']
        else:
```

```
input_value = second_dict['否']
else:
second_dict = Tree[first_feature]
input_first = test_data.get(first_feature)
input_value = second_dict[input_first]
if isinstance(input_value , dict):
class_label = predict(input_value, test_data)
else:
class_label = input_value
return class_label

data = df1
data.reset_index(drop=True, inplace=True)
data=data.iloc[:,1:]
# 统计每个特征的取值情况作为全局变量
column_count = dict([(ds, list(pd.unique(data[ds]))) for ds in data.iloc[:,
    :-1].columns])
continuous_features = [] #连续值
dicision_tree = create_tree(data)
print(dicision_tree)
test_data={'颜色':'蓝绿','纹饰':'C','类型':'高钾'}
result = predict(dicision_tree, test_data)
print(result)

# %%

def data_flatten(key,val,con_s='_',basic_types=(str,int,float,bool,complex,bytes)):
if isinstance(val, dict):
for ck,cv in val.items():
yield from data_flatten(con_s.join([key,ck]).lstrip('_'), cv)
elif isinstance(val, (list,tuple,set)):
for item in val:
yield from data_flatten(key,item)
elif isinstance(val, basic_types) or val is None:
yield str(key).lower(),val

for i in data_flatten('',dicision_tree):
print(i)
```

附录 C　1.2

```
import numpy as np
from matplotlib import pyplot as plt
plt.rcParams['axes.unicode_minus'] = False # 正常显示负号
df1 = pd.read_excel("附件.xlsx",sheet_name=0)
```

```
df2 = pd.read_excel("sheet2_valid.xlsx")

# %%
conditions=[]
ids=[]
cds_labels=['未风化','一般风化','严重风化','高钾','铅钡']
for t in df2.iterrows():
name=t[1].to_list()[0]
cd=[0]*5
ids.append(name)
if name.find("未风化")!=-1:
print("找到未风化点",name)
cd[0]=1
id=str(int(name[:2]))
dd=np.array(df1[df1.文物编号.astype(str).str.contains(id)].iloc[0])
if dd[2]=='高钾':
cd[3]=1
elif dd[2]=='铅钡':
cd[4]=1
conditions.append(cd)
elif name.find("严重风化")!=-1:
print("找到严重风化点",name)
cd[2]=1
id=str(int(name[:2]))
dd=np.array(df1[df1.文物编号.astype(str).str.contains(id)].iloc[0])
if dd[2]=='高钾':
cd[3]=1
elif dd[2]=='铅钡':
cd[4]=1
conditions.append(cd)
else:

id=str(int(name[:2]))
dd=np.array(df1[df1.文物编号.astype(str).str.contains(id)].iloc[0])
if dd[4]=='无风化':
cd[0]=1
else:
cd[1]=1
if dd[2]=='高钾':
cd[3]=1
elif dd[2]=='铅钡':
cd[4]=1
conditions.append(cd)



# %%
dc1=pd.concat([pd.DataFrame(ids,columns=['文物采样点']),pd.DataFrame(conditions,columns=cds_labels)],axi
```

```
dc1.to_excel("conditions1.xlsx",index=False)

# %%
cds_labels=['未风化','一般风化','严重风化','高钾','铅钡','bias']

# %%
import geatpy as ea
import math

# %%
vars_excel=[]
elements=[]
ObjV=[]

for i in range(1,15):
elements.append(df2.columns[i])
values=df2.iloc[:,i].to_numpy().reshape(-1)
maxv=max(values)
minv=max(min(values),0)

def evalVars(Vars):
lost=[]
const=[]
for Var in Vars:
l=0
c=[]
for i in range(len(conditions)):
kVar=Var[0:5]
bVar=Var[5]
l+=(kVar.dot(conditions[i])+bVar-values[i])**2
c.append(kVar.dot(conditions[i])+bVar-maxv)
c.append(minv-kVar.dot(conditions[i])-bVar)
c.append(-(min(kVar[0:3])+min(kVar[3:5])+bVar))
l=l/len(conditions)
l=math.sqrt(l)
const.append(c)
lost.append(l)

ObjV=np.array(lost).reshape(-1,1)
CV = np.array(const)

return ObjV,CV

problem = ea.Problem(name=df2.columns[i],
M=1, # 目标维数
maxormins=[1],
Dim=6, # 决策变量维数
```

```
varTypes=[0, 0, 0, 0, 0,0], # 决策变量的类型
lb=[-100, -100, -100, -100, -100,-100], # 决策变量下界
ub=[100, 100, 100, 100, 100,100], # 决策变量上界
evalVars=evalVars)

algorithm = ea.soea_SEGA_templet(problem,
ea.Population(Encoding='RI', NIND=50),
MAXGEN=200, # 最大进化代数。
logTras=1, # 表示每隔多少代记录一次日志信息，0表示不记录。
trappedValue=1e-6, # 单目标优化陷入停滞的判断阈值。
maxTrappedCount=20) # 进化停滞计数器最大上限值。
# 求解
res = ea.optimize(algorithm, seed=1, verbose=True, drawing=1, outputMsg=True,
    drawLog=False, saveFlag=True, dirName='result')


vars_excel.append(res['Vars'].reshape(-1).tolist())
ObjV.append(res['ObjV'].reshape(-1).tolist())
print(res)



# %%
da=pd.DataFrame(elements,columns=['元素'])
db=pd.DataFrame(vars_excel,columns=cds_labels)
dc=pd.DataFrame(ObjV,columns=['平均标准差'])
dd=pd.concat([da,db,dc],axis=1)
dd.to_excel("result1_2.xlsx",index=False)
```

附录 D　1.3

```
# %%
import math
import pandas as pd
import numpy as np

# %%
df2=pd.read_excel('sheet2_valid.xlsx')
cd_df= pd.read_excel('conditions1.xlsx').iloc[:,1:]
cd_result1=pd.read_excel('result1_2.xlsx').iloc[:,1:-1]

origins=df2.iloc[:,1:]
condition_array=np.array(cd_df)
result_array=np.array(cd_result1)
origins_array=np.array(origins)
```

```
# %%
origins_max=np.max(origins_array,axis=0)
origins_max

# %%
result2=[]
for i in range(len(origins_array)):
elements=[0.]*14
if condition_array[i][0]==1:
result2.append(elements)
continue
bad=condition_array[i]
print(bad)
good=np.array([1,0,0]+list(condition_array[i])[3:])
print(good)
for j in range(result_array.shape[0]):
b=np.dot(bad,result_array[j][0:5])+result_array[j][5]
print("b",j,b)
g=np.dot(good,result_array[j][0:5])+result_array[j][5]
print("g",j,g)
print("g/b",j,g/b)
if b==0:
elements[j]=g
else:
if (g/b)*origins_array[i][j]>origins_max[j]:
elements[j]=g
else:
elements[j]=(g/b)*origins_array[i][j]
se=sum(elements)
elements=[e*100/se for e in elements]
result2.append(elements)

# %%
da=pd.DataFrame(result2,columns=origins.columns)
dr=pd.concat([df2.iloc[:,0],da],axis=1)
dr.to_excel('result2.xlsx',index=False)
dr[dr.iloc[:,1:].sum(axis=1)>0].to_excel('result2_2_cut.xlsx',index=False)
df2[dr.iloc[:,1:].sum(axis=1)>0].to_excel('result2_origin.xlsx',index=False)
```

附录 E　2.1

```
# %%
import pandas as pd
```

```
df=pd.read_excel("table2_1.xlsx")# 修改输入表格的类型
df['玻璃类型'] = df['玻璃类型'].map({'高钾':0,'铅钡':1}) #01标签映射
x=df.iloc[:,1:-3].values#训练的特征
print('训练特征为：',[column for column in df.iloc[:,1:-3]])
y=df.iloc[:,-1].values#训练标签

# %%
#SiO2 Pbo K20 sro fe2o3
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.ensemble import AdaBoostClassifier as ada
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import label_binarize
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
import pandas as pd

df=pd.read_excel("table2_1.xlsx")# 修改输入表格的类型
# labels=["二氧化硅(SiO2)","氧化钾(K2O)","氧化铅(PbO)","氧化钡(BaO)",'玻璃类型']
# df=df[labels]
df['玻璃类型'] = df['玻璃类型'].map({'高钾':0,'铅钡':1}) #01标签映射
x=df.iloc[:,1:-3].values#训练的特征
print('训练特征为：',[column for column in df.iloc[:,1:-3]])
y=df.iloc[:,-1].values#训练标签

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.3)
transformer = StandardScaler()
x=transformer.fit_transform(x)
x_train=transformer.transform(x_train)
x_test=transformer.transform(x_test)

#五种机器学习回归算法训练模型
LR=LogisticRegression(random_state=30)
svc=SVC(kernel='linear',random_state=30)
Ada=ada(random_state=30)
GBDT=GradientBoostingClassifier(random_state=30)
rf=RF(random_state=30)
```

```
for clf,label in zip([LR,svc,Ada,GBDT,rf],['LR','SVC','Ada','GBDT','RF']):
clf.fit(x_train, y_train)
if(label=='LR'or label=='SVC'):
w = clf.coef_                          # 模型系数(对应归一化数据)
b = clf.intercept_                       # 模型阈值(对应归一化数据)
print("\n------{}模型参数-------".format(label))
print( "模型系数:",w)
print( "模型阈值:",b)
y_predict=LR.predict(x_test)
print('y_test:',y_test)
print('y_pred:',y_predict)
print('{}在测试集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_test,y_predict))
print('{}在测试集模型上的召回率为: \n'.format(label),metrics.precision_score(y_test,y_predict))
print('{}在训练集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_train,clf.predict(x_train))
print('{}在训练集模型上的准确率为: \n'.format(label),metrics.precision_score(y_train,clf.predict(x_train)
print('{}在综合准确率为: \n'.format(label),metrics.accuracy_score(y,clf.predict(x)))
scores = cross_val_score(clf, x, y, cv=5)
print('{}在交叉验证准确率为: \n'.format(label),scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))


# print('{}的AUC为: '.format(label),roc_auc_score(y,LR.predict(x)))

# %%
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from pylab import mpl
import matplotlib as plt
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker

# plt.figure(dpi=300,figsize=(24,8))
plt.rcParams['axes.unicode_minus']=False
# 设置显示中文字体
mpl.rcParams["font.sans-serif"] = ["SimHei"]

df=pd.read_excel("table2_1.xlsx")
# labels=["二氧化硅(SiO2)","氧化钾(K2O)","氧化铅(PbO)","氧化钡(BaO)",'玻璃类型']
# df=df[labels]
df['玻璃类型'] = df['玻璃类型'].map({'高钾':0,'铅钡':1})
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
d=df.corr(method="pearson")
```

```
plt.subplots(figsize = (12,12))
sns.heatmap(d,annot = True,vmax = 1,square = True,cmap = "Reds")
plt.rcParams['axes.facecolor']='snow'
plt.savefig('1-1 heatmap-pearson.png')
plt.show()
```

附录 F　2.2.2

```
import numpy as np
from scipy.sparse import *
from sklearn.metrics.pairwise import pairwise_distances


def construct_W(X, **kwargs):
"""
Construct the affinity matrix W through different ways

Notes
-----
if kwargs is null, use the default parameter settings;
if kwargs is not null, construct the affinity matrix according to parameters in kwargs

Input
-----
X: {numpy array}, shape (n_samples, n_features)
input data
kwargs: {dictionary}
parameters to construct different affinity matrix W:
y: {numpy array}, shape (n_samples, 1)
the true label information needed under the 'supervised' neighbor mode
metric: {string}
choices for different distance measures
'euclidean' - use euclidean distance
'cosine' - use cosine distance (default)
neighbor_mode: {string}
indicates how to construct the graph
'knn' - put an edge between two nodes if and only if they are among the
k nearest neighbors of each other (default)
'supervised' - put an edge between two nodes if they belong to same class
and they are among the k nearest neighbors of each other
weight_mode: {string}
indicates how to assign weights for each edge in the graph
'binary' - 0-1 weighting, every edge receives weight of 1 (default)
'heat_kernel' - if nodes i and j are connected, put weight W_ij = exp(-norm(x_i -
```

```
        x_j)/2t^2)
    this weight mode can only be used under 'euclidean' metric and you are required
    to provide the parameter t
    'cosine' - if nodes i and j are connected, put weight cosine(x_i,x_j).
    this weight mode can only be used under 'cosine' metric
    k: {int}
    choices for the number of neighbors (default k = 5)
    t: {float}
    parameter for the 'heat_kernel' weight_mode
    fisher_score: {boolean}
    indicates whether to build the affinity matrix in a fisher score way, in which W_ij =
        1/n_l if yi = yj = l;
    otherwise W_ij = 0 (default fisher_score = false)
    reliefF: {boolean}
    indicates whether to build the affinity matrix in a reliefF way, NH(x) and NM(x,y)
        denotes a set of
    k nearest points to x with the same class as x, and a different class (the class y),
        respectively.
    W_ij = 1 if i = j; W_ij = 1/k if x_j \in NH(x_i); W_ij = -1/(c-1)k if x_j \in NM(x_i,
        y) (default reliefF = false)

    Output
    ------
    W: {sparse matrix}, shape (n_samples, n_samples)
    output affinity matrix W
    """

    # default metric is 'cosine'
    if 'metric' not in kwargs.keys():
    kwargs['metric'] = 'cosine'

    # default neighbor mode is 'knn' and default neighbor size is 5
    if 'neighbor_mode' not in kwargs.keys():
    kwargs['neighbor_mode'] = 'knn'
    if kwargs['neighbor_mode'] == 'knn' and 'k' not in kwargs.keys():
    kwargs['k'] = 5
    if kwargs['neighbor_mode'] == 'supervised' and 'k' not in kwargs.keys():
    kwargs['k'] = 5
    if kwargs['neighbor_mode'] == 'supervised' and 'y' not in kwargs.keys():
    print ('Warning: label is required in the supervised neighborMode!!!')
    exit(0)

    # default weight mode is 'binary', default t in heat kernel mode is 1
    if 'weight_mode' not in kwargs.keys():
    kwargs['weight_mode'] = 'binary'
    if kwargs['weight_mode'] == 'heat_kernel':
    if kwargs['metric'] != 'euclidean':
```

```
        kwargs['metric'] = 'euclidean'
        if 't' not in kwargs.keys():
        kwargs['t'] = 1
        elif kwargs['weight_mode'] == 'cosine':
        if kwargs['metric'] != 'cosine':
        kwargs['metric'] = 'cosine'

        # default fisher_score and reliefF mode are 'false'
        if 'fisher_score' not in kwargs.keys():
        kwargs['fisher_score'] = False
        if 'reliefF' not in kwargs.keys():
        kwargs['reliefF'] = False

        n_samples, n_features = np.shape(X)

        # choose 'knn' neighbor mode
        if kwargs['neighbor_mode'] == 'knn':
        k = kwargs['k']
        if kwargs['weight_mode'] == 'binary':
        if kwargs['metric'] == 'euclidean':
        # compute pairwise euclidean distances
        D = pairwise_distances(X)
        D **= 2
        # sort the distance matrix D in ascending order
        dump = np.sort(D, axis=1)
        idx = np.argsort(D, axis=1)
        # choose the k-nearest neighbors for each instance
        idx_new = idx[:, 0:k+1]
        G = np.zeros((n_samples*(k+1), 3))
        G[:, 0] = np.tile(np.arange(n_samples), (k+1, 1)).reshape(-1)
        G[:, 1] = np.ravel(idx_new, order='F')
        G[:, 2] = 1
        # build the sparse affinity matrix W
        W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
        bigger = np.transpose(W) > W
        W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
        return W

        elif kwargs['metric'] == 'cosine':
        # normalize the data first
        X_normalized = np.power(np.sum(X*X, axis=1), 0.5)
        for i in range(n_samples):
        X[i, :] = X[i, :]/max(1e-12, X_normalized[i])
        # compute pairwise cosine distances
        D_cosine = np.dot(X, np.transpose(X))
        # sort the distance matrix D in descending order
        dump = np.sort(-D_cosine, axis=1)
```

```
        idx = np.argsort(-D_cosine, axis=1)
        idx_new = idx[:, 0:k+1]
        G = np.zeros((n_samples*(k+1), 3))
        G[:, 0] = np.tile(np.arange(n_samples), (k+1, 1)).reshape(-1)
        G[:, 1] = np.ravel(idx_new, order='F')
        G[:, 2] = 1
        # build the sparse affinity matrix W
        W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
        bigger = np.transpose(W) > W
        W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
        return W

        elif kwargs['weight_mode'] == 'heat_kernel':
        t = kwargs['t']
        # compute pairwise euclidean distances
        D = pairwise_distances(X)
        D **= 2
        # sort the distance matrix D in ascending order
        dump = np.sort(D, axis=1)
        idx = np.argsort(D, axis=1)
        idx_new = idx[:, 0:k+1]
        dump_new = dump[:, 0:k+1]
        # compute the pairwise heat kernel distances
        dump_heat_kernel = np.exp(-dump_new/(2*t*t))
        G = np.zeros((n_samples*(k+1), 3))
        G[:, 0] = np.tile(np.arange(n_samples), (k+1, 1)).reshape(-1)
        G[:, 1] = np.ravel(idx_new, order='F')
        G[:, 2] = np.ravel(dump_heat_kernel, order='F')
        # build the sparse affinity matrix W
        W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
        bigger = np.transpose(W) > W
        W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
        return W

        elif kwargs['weight_mode'] == 'cosine':
        # normalize the data first
        X_normalized = np.power(np.sum(X*X, axis=1), 0.5)
        for i in range(n_samples):
        X[i, :] = X[i, :]/max(1e-12, X_normalized[i])
        # compute pairwise cosine distances
        D_cosine = np.dot(X, np.transpose(X))
        # sort the distance matrix D in ascending order
        dump = np.sort(-D_cosine, axis=1)
        idx = np.argsort(-D_cosine, axis=1)
        idx_new = idx[:, 0:k+1]
        dump_new = -dump[:, 0:k+1]
        G = np.zeros((n_samples*(k+1), 3))
```

```
G[:, 0] = np.tile(np.arange(n_samples), (k+1, 1)).reshape(-1)
G[:, 1] = np.ravel(idx_new, order='F')
G[:, 2] = np.ravel(dump_new, order='F')
# build the sparse affinity matrix W
W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
bigger = np.transpose(W) > W
W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
return W

# choose supervised neighborMode
elif kwargs['neighbor_mode'] == 'supervised':
k = kwargs['k']
# get true labels and the number of classes
y = kwargs['y']
label = np.unique(y)
n_classes = np.unique(y).size
# construct the weight matrix W in a fisherScore way, W_ij = 1/n_l if yi = yj = l,
    otherwise W_ij = 0
if kwargs['fisher_score'] is True:
W = lil_matrix((n_samples, n_samples))
for i in range(n_classes):
class_idx = (y == label[i])
class_idx_all = (class_idx[:, np.newaxis] & class_idx[np.newaxis, :])
W[class_idx_all] = 1.0/np.sum(np.sum(class_idx))
return W

# construct the weight matrix W in a reliefF way, NH(x) and NM(x,y) denotes a set of k
    nearest
# points to x with the same class as x, a different class (the class y), respectively.
    W_ij = 1 if i = j;
# W_ij = 1/k if x_j \in NH(x_i); W_ij = -1/(c-1)k if x_j \in NM(x_i, y)
if kwargs['reliefF'] is True:
# when xj in NH(xi)
G = np.zeros((n_samples*(k+1), 3))
id_now = 0
for i in range(n_classes):
class_idx = np.column_stack(np.where(y == label[i]))[:, 0]
D = pairwise_distances(X[class_idx, :])
D **= 2
idx = np.argsort(D, axis=1)
idx_new = idx[:, 0:k+1]
n_smp_class = (class_idx[idx_new[:]]).size
if len(class_idx) <= k:
k = len(class_idx) - 1
G[id_now:n_smp_class+id_now, 0] = np.tile(class_idx, (k+1, 1)).reshape(-1)
G[id_now:n_smp_class+id_now, 1] = np.ravel(class_idx[idx_new[:]], order='F')
G[id_now:n_smp_class+id_now, 2] = 1.0/k
```

```
id_now += n_smp_class
W1 = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
# when i = j, W_ij = 1
for i in range(n_samples):
W1[i, i] = 1
# when x_j in NM(x_i, y)
G = np.zeros((n_samples*k*(n_classes - 1), 3))
id_now = 0
for i in range(n_classes):
class_idx1 = np.column_stack(np.where(y == label[i]))[:, 0]
X1 = X[class_idx1, :]
for j in range(n_classes):
if label[j] != label[i]:
class_idx2 = np.column_stack(np.where(y == label[j]))[:, 0]
X2 = X[class_idx2, :]
D = pairwise_distances(X1, X2)
idx = np.argsort(D, axis=1)
idx_new = idx[:, 0:k]
n_smp_class = len(class_idx1)*k
G[id_now:n_smp_class+id_now, 0] = np.tile(class_idx1, (k, 1)).reshape(-1)
G[id_now:n_smp_class+id_now, 1] = np.ravel(class_idx2[idx_new[:]], order='F')
G[id_now:n_smp_class+id_now, 2] = -1.0/((n_classes-1)*k)
id_now += n_smp_class
W2 = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
bigger = np.transpose(W2) > W2
W2 = W2 - W2.multiply(bigger) + np.transpose(W2).multiply(bigger)
W = W1 + W2
return W

if kwargs['weight_mode'] == 'binary':
if kwargs['metric'] == 'euclidean':
G = np.zeros((n_samples*(k+1), 3))
id_now = 0
for i in range(n_classes):
class_idx = np.column_stack(np.where(y == label[i]))[:, 0]
# compute pairwise euclidean distances for instances in class i
D = pairwise_distances(X[class_idx, :])
D **= 2
# sort the distance matrix D in ascending order for instances in class i
idx = np.argsort(D, axis=1)
idx_new = idx[:, 0:k+1]
n_smp_class = len(class_idx)*(k+1)
G[id_now:n_smp_class+id_now, 0] = np.tile(class_idx, (k+1, 1)).reshape(-1)
G[id_now:n_smp_class+id_now, 1] = np.ravel(class_idx[idx_new[:]], order='F')
G[id_now:n_smp_class+id_now, 2] = 1
id_now += n_smp_class
# build the sparse affinity matrix W
```

```
W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
bigger = np.transpose(W) > W
W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
return W

if kwargs['metric'] == 'cosine':
# normalize the data first
X_normalized = np.power(np.sum(X*X, axis=1), 0.5)
for i in range(n_samples):
X[i, :] = X[i, :]/max(1e-12, X_normalized[i])
G = np.zeros((n_samples*(k+1), 3))
id_now = 0
for i in range(n_classes):
class_idx = np.column_stack(np.where(y == label[i]))[:, 0]
# compute pairwise cosine distances for instances in class i
D_cosine = np.dot(X[class_idx, :], np.transpose(X[class_idx, :]))
# sort the distance matrix D in descending order for instances in class i
idx = np.argsort(-D_cosine, axis=1)
idx_new = idx[:, 0:k+1]
n_smp_class = len(class_idx)*(k+1)
G[id_now:n_smp_class+id_now, 0] = np.tile(class_idx, (k+1, 1)).reshape(-1)
G[id_now:n_smp_class+id_now, 1] = np.ravel(class_idx[idx_new[:]], order='F')
G[id_now:n_smp_class+id_now, 2] = 1
id_now += n_smp_class
# build the sparse affinity matrix W
W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
bigger = np.transpose(W) > W
W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
return W

elif kwargs['weight_mode'] == 'heat_kernel':
G = np.zeros((n_samples*(k+1), 3))
id_now = 0
for i in range(n_classes):
class_idx = np.column_stack(np.where(y == label[i]))[:, 0]
# compute pairwise cosine distances for instances in class i
D = pairwise_distances(X[class_idx, :])
D **= 2
# sort the distance matrix D in ascending order for instances in class i
dump = np.sort(D, axis=1)
idx = np.argsort(D, axis=1)
idx_new = idx[:, 0:k+1]
dump_new = dump[:, 0:k+1]
t = kwargs['t']
# compute pairwise heat kernel distances for instances in class i
dump_heat_kernel = np.exp(-dump_new/(2*t*t))
n_smp_class = len(class_idx)*(k+1)
```

```
        G[id_now:n_smp_class+id_now, 0] = np.tile(class_idx, (k+1, 1)).reshape(-1)
        G[id_now:n_smp_class+id_now, 1] = np.ravel(class_idx[idx_new[:]], order='F')
        G[id_now:n_smp_class+id_now, 2] = np.ravel(dump_heat_kernel, order='F')
        id_now += n_smp_class
        # build the sparse affinity matrix W
        W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
        bigger = np.transpose(W) > W
        W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
        return W

    elif kwargs['weight_mode'] == 'cosine':
        # normalize the data first
        X_normalized = np.power(np.sum(X*X, axis=1), 0.5)
        for i in range(n_samples):
            X[i, :] = X[i, :]/max(1e-12, X_normalized[i])
        G = np.zeros((n_samples*(k+1), 3))
        id_now = 0
        for i in range(n_classes):
            class_idx = np.column_stack(np.where(y == label[i]))[:, 0]
            # compute pairwise cosine distances for instances in class i
            D_cosine = np.dot(X[class_idx, :], np.transpose(X[class_idx, :]))
            # sort the distance matrix D in descending order for instances in class i
            dump = np.sort(-D_cosine, axis=1)
            idx = np.argsort(-D_cosine, axis=1)
            idx_new = idx[:, 0:k+1]
            dump_new = -dump[:, 0:k+1]
            n_smp_class = len(class_idx)*(k+1)
            G[id_now:n_smp_class+id_now, 0] = np.tile(class_idx, (k+1, 1)).reshape(-1)
            G[id_now:n_smp_class+id_now, 1] = np.ravel(class_idx[idx_new[:]], order='F')
            G[id_now:n_smp_class+id_now, 2] = np.ravel(dump_new, order='F')
            id_now += n_smp_class
        # build the sparse affinity matrix W
        W = csc_matrix((G[:, 2], (G[:, 0], G[:, 1])), shape=(n_samples, n_samples))
        bigger = np.transpose(W) > W
        W = W - W.multiply(bigger) + np.transpose(W).multiply(bigger)
        return W
```

附录 G　2.2.3

```
import numpy as np
from scipy.sparse import *
from construct_W import construct_W


def lap_score(X, **kwargs):
```

```
    """
    This function implements the laplacian score feature selection, steps are as follows:
    1. Construct the affinity matrix W if it is not specified;
    2. For the r-th feature, we define fr = X(:,r), D = diag(W * ones), ones = [1,...,1]',
        L = D - W
    3. Let fr_hat = fr - (fr' * D * ones) * ones/(ones ' * D * ones)
    4. Laplacian score for the r-th feature is score = (fr_hat'*L*fr_hat)/(fr_hat'*D*fr_hat)

    Input:
    -------
    X: (numpy array), shape (n_samples, n_features) input data
    kwargs: {dictionary} W: {sparse matrix}, shape (n_samples, n_samples)
    input affinity matrix

    Output:
    -------
    score: {numpy array}, shape (n_features,) laplacian score for each feature
    """

    if 'W' not in kwargs.keys():
    W = construct_W(X)
    # construct the affinity matrix W
    W = kwargs['W']
    # build the diagonal D matrix from affinity matrix W
    D = np.array(W.sum(axis=1))
    L = W
    tmp = np.dot(np.transpose(D), X)
    D = diags(np.transpose(D), [0])
    Xt = np.transpose(X)
    t1 = np.transpose(np.dot(Xt, D.todense()))
    t2 = np.transpose(np.dot(Xt, L.todense()))
    # compute the numerator of Lr
    D_prime = np.sum(np.multiply(t1, X), 0) - np.multiply(tmp, tmp)/D.sum()
    # compute the denominator of Lr
    L_prime = np.sum(np.multiply(t2, X), 0) - np.multiply(tmp, tmp)/D.sum()
    # avoid the denominator of Lr to be 0
    D_prime[D_prime < 1e-12] = 10000

    # compute Laplacian Score for all features
    score = 1 - np.array(np.multiply(L_prime, 1/D_prime))[0, :]
    return np.transpose(score)


    def feature_ranking(score):
    """
    Rank features in ascending order according to their laplacian scores,
    the smaller the Laplacian Score is, the more important the feature is
```

```
    """
    idx = np.argsort(score, 0)
    return idx
```

附录 H　3.1

```
    # %%
    #SiO2 Pbo K20 sro fe2o3
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import GridSearchCV
    from sklearn import metrics
    from sklearn.ensemble import AdaBoostClassifier as ada
    from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.svm import SVC
    from sklearn.ensemble import RandomForestClassifier as RF
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import roc_auc_score
    from sklearn.ensemble import VotingClassifier
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.preprocessing import label_binarize
    from sklearn.metrics import precision_score
    from sklearn.metrics import f1_score
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import KFold
    import pandas as pd
    import matplotlib as plt
    import seaborn as sns
    from sklearn.ensemble import RandomForestClassifier as RF
    df=pd.read_excel("table2_3.xlsx")
    df=df[df['未风化']==1]
    df=df.drop(['未风化','风化'],axis=1)
    x=df.iloc[:,1:-1].values
    y=df.iloc[:,-1].values
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.2)
    transformer = StandardScaler()
    x=transformer.fit_transform(x)
    x_train=transformer.transform(x_train)
    x_test=transformer.transform(x_test)

    LR1=LogisticRegression(
    C=0.1, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1,l1_ratio=None, max_iter=500, multi_class='auto', n_jobs=None,
    penalty='l2', random_state=30, solver='newton-cg', tol=0.0001,
```

```
verbose=0, warm_start=False
)

svc1=SVC(C=0.1, kernel='linear', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=True, tol=0.001, cache_size=200,
class_weight=None, verbose=False, max_iter=100, decision_function_shape='ovr',
    random_state=None)
for clf,label in zip([LR1,svc1],['LR','SVC']):
# kfold = KFold(n_splits=5)

clf.fit(x_train, y_train)
w = clf.coef_                          # 模型系数(对应归一化数据)
b = clf.intercept_                        # 模型阈值(对应归一化数据)
y_predict=clf.predict(x_test)
print('y_test:',y_test)
print('y_pred:',y_predict)
print('{}在测试集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_test,y_predict))
print('{}在测试集模型上的召回率为: \n'.format(label),metrics.precision_score(y_test,y_predict))
print('{}在训练集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_train,clf.predict(x_trai
print('{}在训练集模型上的准确率为: \n'.format(label),metrics.precision_score(y_train,clf.predict(x_tra
print('{}在综合准确率为: \n'.format(label),metrics.accuracy_score(y,clf.predict(x)))
scores = cross_val_score(clf, x, y, cv=5)
print('{}在交叉验证准确率为: \n'.format(label),scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

print("\n------模型参数-------")
print( "模型系数:",w)
print( "模型阈值:",b)
print('{}的AUC为: '.format(label),roc_auc_score(y, clf.predict(x)))

# %%
df=pd.read_excel("table2_3.xlsx")
df=df[df['风化']==1]
df=df.drop(['未风化','风化'],axis=1)
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.2)
transformer = StandardScaler()
x=transformer.fit_transform(x)
x_train=transformer.transform(x_train)
x_test=transformer.transform(x_test)

LR2=LogisticRegression(
C=0.1, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1,l1_ratio=None, max_iter=500, multi_class='auto', n_jobs=None,
penalty='l2', random_state=30, solver='newton-cg', tol=0.0001,
verbose=0, warm_start=False
```

```
    )

    svc2=SVC(C=0.1, kernel='linear', degree=3, gamma='auto', coef0=0.0,
    shrinking=True, probability=True, tol=0.001, cache_size=200,
    class_weight=None, verbose=False, max_iter=100, decision_function_shape='ovr',
        random_state=None)

    rf2=RF(n_estimators= 60, max_depth=13, min_samples_split=120,
    min_samples_leaf=20,max_features=7 ,oob_score=True, random_state=10)

    for clf,label in zip([LR2,svc2],['LR','SVC']):
    # kfold = KFold(n_splits=5)

    clf.fit(x_train, y_train)
    w = clf.coef_                              # 模型系数(对应归一化数据)
    b = clf.intercept_                           # 模型阈值(对应归一化数据)
    y_predict=clf.predict(x_test)
    print('y_test:',y_test)
    print('y_pred:',y_predict)
    print('{}在测试集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_test,y_predict))
    print('{}在测试集模型上的召回率为: \n'.format(label),metrics.precision_score(y_test,y_predict))
    print('{}在训练集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_train,clf.predict(x_trai
    print('{}在训练集模型上的准确率为: \n'.format(label),metrics.precision_score(y_train,clf.predict(x_tra
    print('{}在综合准确率为: \n'.format(label),metrics.accuracy_score(y,clf.predict(x)))
    scores = cross_val_score(clf, x, y, cv=5)
    print('{}在交叉验证准确率为: \n'.format(label),scores)
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

    print("\n------模型参数-------")
    print( "模型系数:",w)
    print( "模型阈值:",b)
    print('{}的AUC为: '.format(label),roc_auc_score(y,clf.predict(x)))
```

附录 I　3.2

```
# %%
#SiO2 Pbo K20 sro fe2o3
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.ensemble import AdaBoostClassifier as ada
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
```

```
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import label_binarize
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import random
from sklearn.ensemble import RandomForestClassifier as RF
df=pd.read_excel("table2_3.xlsx")
remove_list=['氧化钠(Na2O)','氧化镁(MgO)','氧化铝(Al2O3)','氧化铜(CuO)','五氧化二磷(P2O5)',
    '氧化锡(SnO2)','二氧化硫(SO2)']
df=df.drop(remove_list,axis=1)
df=df[df['未风化']==1]
df=df.drop(['未风化','风化'],axis=1)
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=36,test_size=0.2)
transformer = StandardScaler()
x=transformer.fit_transform(x)
x_train=transformer.transform(x_train)
x_test=transformer.transform(x_test)

LR1=LogisticRegression(
C=0.1, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1,l1_ratio=None, max_iter=40, multi_class='auto', n_jobs=None,
penalty='l2', random_state=40, solver='newton-cg', tol=0.0001,
verbose=0, warm_start=False
)

clf,label = LR1,"LR"
clf.fit(x_train, y_train)
w = clf.coef_                          # 模型系数(对应归一化数据)
b = clf.intercept_
y_predict=clf.predict(x_test)
print('y_test:',y_test)
print('y_pred:',y_predict)
print('{}在测试集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_test,y_predict))
print("\n------模型参数-------")
print( "模型系数:",w)
```

```
print( "模型阈值:",b)

# %%
print(LR1.predict(x_test))
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
accs_uw=[]
shakes=[0,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1]
for i in shakes:
x_shake=[]
_,x_test,_,y_test=train_test_split(x,y,test_size=0.5)
x_test=transformer.transform(x_test)
for xs in x_test:
x_shake.append([xt*(1+i*(random.random()-0.5)) for xt in xs])
x_shake=np.array(x_shake)
y_predict=LR1.predict(x_shake)
print('y_test:',y_test)
print('y_pred:',y_predict)
acc=metrics.accuracy_score(y_test,y_predict)
print('{}在测试集模型上的准确率为: \n'.format(label))
accs_uw.append(acc)

# %%
df=pd.read_excel("table2_3.xlsx")
df=df[df['风化']==1]
df=df.drop(['未风化','风化'],axis=1)
df=df.drop(remove_list,axis=1)
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.2)
transformer = StandardScaler()
x=transformer.fit_transform(x)
x_train=transformer.transform(x_train)
x_test=transformer.transform(x_test)

LR2=LogisticRegression(
C=0.1, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1,l1_ratio=None, max_iter=40, multi_class='auto', n_jobs=None,
penalty='l2', random_state=40, solver='newton-cg', tol=0.0001,
verbose=0, warm_start=False
)

clf,label = LR2,"LR"
clf.fit(x_train, y_train)
w = clf.coef_                          # 模型系数(对应归一化数据)
b = clf.intercept_
y_predict=clf.predict(x_test)
```

```
print('y_test:',y_test)
print('y_pred:',y_predict)
print('{}在测试集模型上的准确率为: \n'.format(label),metrics.accuracy_score(y_test,y_predict))
print("\n------模型参数-------")
print( "模型系数:",w)
print( "模型阈值:",b)

# %%
accs_w=[]
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
for i in shakes:
x_shake=[]
_,x_test,_,y_test=train_test_split(x,y,test_size=0.5)
x_test=transformer.transform(x_test)
for xs in x_test:
x_shake.append([xt*(1+i*(random.random()-0.5)) for xt in xs])
x_shake=np.array(x_shake)
y_predict=clf.predict(x_shake)
print('y_test:',y_test)
print('y_pred:',y_predict)
acc=metrics.accuracy_score(y_test,y_predict)
print('{}在测试集模型上的准确率为: \n'.format(label),acc)
accs_w.append(acc)

# %%
plt.style.use('ggplot')
fig,axes=plt.subplots(1,2,figsize=(14,6))
axes[0].plot(range(len(shakes)),accs_uw,label='未风化')
axes[1].plot(range(len(shakes)),accs_w,label='风化')
axes[0].set_xticks(range(len(shakes)),shakes)
axes[1].set_xticks(range(len(shakes)),shakes)
axes[0].set_title('未风化LR1')
axes[1].set_title('风化LR2')
axes[0].set_xlabel('扰动百分比范围')
axes[0].set_ylabel('准确率')

axes[1].set_xlabel('扰动百分比范围')
axes[1].set_ylabel('准确率')

plt.savefig('shake.png',dpi=300)
plt.show()




# %%
```

```
df3=pd.read_excel("附件.xlsx",sheet_name=2).fillna(0)
df3=df3.drop(remove_list,axis=1)
tem=df3.pop('表面风化')
ans=[]
arr=np.array(tem)
for i in range(8):
x=df3.iloc[i,1:]
if arr[i]=='无风化':
pre_y=LR1.predict(x.values.reshape(1,-1))
else:
pre_y=LR2.predict(x.values.reshape(1,-1))
ans.append(pre_y[0])

# %%
df3.insert(df3.shape[1],'玻璃类型',ans)
df3.loc[df3['玻璃类型']==0,'玻璃类型'] = '高钾'
df3.loc[df3['玻璃类型']==1,'玻璃类型'] = '铅钡'
df3.insert(1,'表面风化',tem)
df3

# %%
df4=pd.concat([df3[['文物编号']],df3[['玻璃类型']]],axis=1)
df4.to_excel('问题3预测结果.xlsx',index=False)
```

附录 J　3.3

```
# %%
#SiO2 Pbo K20 sro fe2o3
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.ensemble import AdaBoostClassifier as ada
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import label_binarize
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import KFold
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns
import numpy as np
from sklearn.ensemble import RandomForestClassifier as RF
df=pd.read_excel("table2_3.xlsx")
remove_list=['氧化钠(Na2O)','氧化镁(MgO)','氧化铝(Al2O3)','氧化铜(CuO)','五氧化二磷(P2O5)',
    '氧化锡(SnO2)','二氧化硫(SO2)']
df=df.drop(remove_list,axis=1)
df=df[df['未风化']==1]
df=df.drop(['未风化','风化'],axis=1)
# labels=["二氧化硅(SiO2)","氧化钾(K2O)","氧化铅(PbO)","氧化钡(BaO)",'玻璃类型']
# df=df[labels]
# df['玻璃类型'] = df['玻璃类型'].map({'高钾':0,'铅钡':1})
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.2)
transformer = StandardScaler()
x=transformer.fit_transform(x)
x_train=transformer.transform(x_train)
x_test=transformer.transform(x_test)

# %%
accurary_uw=[]
std_uw=[]
iter_list=list(range(1,21))
iter_list=iter_list+[40,60,100,200]
print(type(iter_list))
for i in iter_list:
LR1=LogisticRegression(
C=0.1, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1,l1_ratio=None, max_iter=i, multi_class='auto', n_jobs=None,
penalty='l2', random_state=40, solver='newton-cg', tol=0.0001,
verbose=0, warm_start=False
)

clf,label = LR1,"LR"
scores = cross_val_score(clf, x, y, cv=5)
print('{}在交叉验证准确率为: \n'.format(label),scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
accurary_uw.append(scores.mean())
std_uw.append(scores.std() * 2)


# %%
```

```
plt.style.use('ggplot')
fig,axes=plt.subplots(1,2,figsize=(14,6))
axes[0].plot(range(len(accurary_uw)),accurary_uw,label='Accuracy')
axes[0].set_xlabel('max_iter')
axes[0].set_ylabel('Accuracy')
axes[0].set_xticks(range(len(accurary_uw)),[str(x) for x in iter_list])

axes[1].plot(range(len(accurary_uw)),std_uw,label='std')
axes[1].set_xlabel('max_iter')
axes[1].set_ylabel('std')
axes[1].set_xticks(range(len(accurary_uw)),[str(x) for x in iter_list])

plt.savefig('max_iter_uw.png')

# %%
df=pd.read_excel("table2_3.xlsx")
remove_list=['氧化钠(Na2O)','氧化镁(MgO)','氧化铝(Al2O3)','氧化铜(CuO)','五氧化二磷(P2O5)',
    '氧化锡(SnO2)','二氧化硫(SO2)']
df=df.drop(remove_list,axis=1)
df=df[df['风化']==1]
df=df.drop(['未风化','风化'],axis=1)
x=df.iloc[:,1:-1].values
y=df.iloc[:,-1].values
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.2)
transformer = StandardScaler()
x=transformer.fit_transform(x)
x_train=transformer.transform(x_train)
x_test=transformer.transform(x_test)

# %%
accurary_uw=[]
std_uw=[]
iter_list=list(range(1,21))
iter_list=iter_list+[40,60,100,200]
print(type(iter_list))
for i in iter_list:
LR2=LogisticRegression(
C=0.1, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1,l1_ratio=None, max_iter=i, multi_class='auto', n_jobs=None,
penalty='l2', random_state=40, solver='newton-cg', tol=0.0001,
verbose=0, warm_start=False
)

clf,label = LR2,"LR"
scores = cross_val_score(clf, x, y, cv=5)
print('{}在交叉验证准确率为: \n'.format(label),scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
    accurary_uw.append(scores.mean())
    std_uw.append(scores.std() * 2)

    # %%
    plt.style.use('ggplot')
    fig,axes=plt.subplots(1,2,figsize=(14,6))
    axes[0].plot(range(len(accurary_uw)),accurary_uw,label='Accuracy')
    axes[0].set_xlabel('max_iter')
    axes[0].set_ylabel('Accuracy')
    axes[0].set_xticks(range(len(accurary_uw)),[str(x) for x in iter_list])

    axes[1].plot(range(len(accurary_uw)),std_uw,label='std')
    axes[1].set_xlabel('max_iter')
    axes[1].set_ylabel('std')
    axes[1].set_xticks(range(len(accurary_uw)),[str(x) for x in iter_list])

    plt.savefig('max_iter_w.png')

    # %%
    df=pd.read_excel("table2_3.xlsx")
    remove_list=['氧化钠(Na2O)','氧化镁(MgO)','氧化铝(Al2O3)','氧化铜(CuO)','五氧化二磷(P2O5)',
        '氧化锡(SnO2)','二氧化硫(SO2)']
    df=df.drop(remove_list,axis=1)
    df=df[df['未风化']==1]
    df=df.drop(['未风化','风化'],axis=1)
    x=df.iloc[:,1:-1].values
    y=df.iloc[:,-1].values
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=30,test_size=0.2)
    transformer = StandardScaler()
    x=transformer.fit_transform(x)
    x_train=transformer.transform(x_train)
    x_test=transformer.transform(x_test)
```

附录 K　4

```
    # %%
    from sklearn.decomposition import PCA
    from sklearn.manifold import TSNE
    from pylab import mpl
    import matplotlib as plt
    import seaborn as sns
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import matplotlib.ticker as ticker
```

```
# plt.figure(dpi=300,figsize=(24,8))
plt.rcParams['axes.unicode_minus']=False
# 设置显示中文字体
mpl.rcParams["font.sans-serif"] = ["SimHei"]


df_wei_K=pd.read_excel("4.xlsx",sheet_name='未风化高钾')
d=df_wei_K.corr(method="pearson")
plt.subplots(figsize = (12,12))
sns.heatmap(d,annot = True,vmax = 1,square = True,cmap = "Reds")
plt.rcParams['axes.facecolor']='snow'
plt.title('未风化高钾')
plt.savefig('4 heatmap-pearson-wei-K.png')
plt.show()

df_wei_Ba=pd.read_excel("4.xlsx",sheet_name='未风化铅钡')
d=df_wei_Ba.corr(method="pearson")
plt.subplots(figsize = (12,12))
sns.heatmap(d,annot = True,vmax = 1,square = True,cmap = "Reds")
plt.rcParams['axes.facecolor']='snow'
plt.title('未风化铅钡')
plt.savefig('4 heatmap-pearson-wei-Ba.png')
plt.show()


df_yi_K=pd.read_excel("4.xlsx",sheet_name='风化高钾')
plt.subplots(figsize = (12,12))
d=df_yi_K.corr(method="pearson")
sns.heatmap(d,annot = True,vmax = 1,square = True,cmap = "Reds")
plt.rcParams['axes.facecolor']='snow'
plt.title('风化高钾')
plt.savefig('4 heatmap-pearson-yi-K.png')
plt.show()

df_yi_Ba=pd.read_excel("4.xlsx",sheet_name='风化铅钡')
plt.subplots(figsize = (12,12))
d=df_wei_Ba.corr(method="pearson")
sns.heatmap(d,annot = True,vmax = 1,square = True,cmap = "Reds")
plt.rcParams['axes.facecolor']='snow'
plt.title('风化铅钡')
plt.savefig('4 heatmap-pearson-yi-Ba.png')
plt.show()
```