黄土高原植被覆盖面积数据分析

摘要

黄土高原地区是中国生态环境最为脆弱、水土流失最为严重的地区之一，这一地区的植被覆盖情况有着重要的生态学意义。在问题一中，我们通过时间序列聚类的方法对不同象元进行了分类，并在问题二中通过选取代表类并采用三次指数平滑法进行分析预测，确定了 NDVI 数据的变化趋势。在问题三和四中，我们对比了不同因素和 NDVI 的偏相关系数，从而确定了影响植被覆盖面积的主要原因。

在问题一中，由于地区的植被覆盖情况和季节密切相关，因此不能在聚类时忽略地区的时间特性，直接选取某一地区 NDVI 的均值来反映当地的植被覆盖情况。我们可以考虑使用 Kmeans 算法对数据进行时间序列聚类，并且用 **DTW 算法**计算类间距离。我们对比了使用不同聚类数量时的样本轮廓系数，最终把样本聚为 5 类。

问题二中可以选择问题一中部分包含元素较多，代表性较强的 4 个分类的时间序列数据进行进一步分析。考虑到数据的季节周期性，我们采用考虑季节因素的**三次指数平滑法**对数据进行了建模分析，这一方法相对传统的回归方法能在受到多种因素影响的实际情况下，更好的反映数据在未来的变化情况。之后我们通过移动平均法分析发现 NDVI 数据呈现周期性的上升趋势。

问题三要求我们分析植被状况和全球变暖的关系。我们把这一问题转化成了分析 NDVI 指数与温度的关系。我们整理了黄土高原地区 5 个县市的气象观测点从 1984 年起以月为单位的气温数据，并与植被覆盖情况的增长趋势相对比，最终通过计算气温数据与 NDVI 指数的相关系数的方法确定二者间的相关性。得到的几种相关系数的取值都大于 0.8，说明气温指数与 NDVI 有较强的正相关性。

在问题四中我们需要分析近年来黄土高原植被变化的主要原因。考虑到人类活动与气候变化因素间也有一定的相关性，我们在控制人类活动影响的情况下求解温度和降水量关于 NDVI 的**偏相关系数**，并在控制气候因素的影响后求解植树量关于 NDVI 的偏相关系数，对比两个偏相关系数来确定了环境温度是黄土高原植被覆盖面积变化的主要原因。并且就进一步扩大植被覆盖面积给出了合理建议。

关键字： DTW 算法　KMeans 聚类　三次指数平滑　偏相关系数

一、 问题重述

黄土高原地处黄河中游地区, 是中国生态环境最为脆弱、水土流失最为严重的地区之一。已有研究表明, 人类不合理的开发活动是导致该地区生态退化的最主要原因，而近年来的退耕还林 (草)、自然林保护等生态工程建设是改善该地区生态环境的主要措施。我们需要分析黄土高原的归一化植被指数 (NDVI)，从而得到黄土高原的植被状况以及以上状况和全球变暖，人类活动等因素的关系。

某个地区的归一化植被指数（NDVI）与生物量、叶面积指数有较高的相关性，能够较好的表示某一地区的地表植被情况。我们获取了黄土高原地区的 9150 个象元的 NDVI 值，时间跨度为 1982 年 01 月-2015 年 12 月，时间分辨率为一个月。对 NDVI 的时间序列分析可以反映植物的生长状态和变化规律。

在问题 1 中，我们需要依据 NDVI 数据对 9150 个象元进行聚类分析，从而实现对于数据的分类和降维，为进一步的处理做准备。

在问题 2 中，我们需要依据聚类的结果选取几个代表类，在每个代表类中把数据进行平均得到以月为单位的时间序列数据，并对以上数据的变化趋势建模分析其变化趋势。

在问题 3 中，在全球变暖的背景下黄土高原的植被状况与全球变暖的相关关系。在问题 4 中，我们需要结合问题 2 的结果，分析近年来黄土高原植被变化的主要原因并确定使黄土高原的植被覆盖面积取得最大改善的条件。

二、 问题分析

在问题一中，由于地区的植被覆盖情况和季节密切相关，因此不能在聚类时忽略地区的时间特性，直接选取某一地区 NDVI 的均值来反映当地的植被覆盖情况。我们可以考虑使用 DTW 算法计算距离，并且与 Kmeans 算法复合来对数据进行时间序列聚类，由于聚类数量这一参数的取值无法确定，我们需要对比使用不同聚类数量时的样本轮廓系数进行敏感性分析。

在获得了问题一的聚类结果后，我们在问题二中可以选择问题一中部分包含元素较多，代表性较强的分类进行进一步分析。我们首先把各个类别的数据进行平均，得到关于各个类别的、以月为单位的时间序列数据。之后我们考虑到数据的季节周期性，尝试了三次指数平滑模型对数据进行了建模分析，得到了 NDVI 在未来的预测结果，并使用移动平均法得到 NDVI 呈现增长趋势的结论。

问题三要求我们分析植被状况和全球变暖的关系。我们把这一问题转化成了分析 NDVI 指数与温度的关系。我们整理了黄土高原地区 5 个县市的气象观测点从 1984 年

起以月为单位的气温数据，并与植被覆盖情况的增长趋势相对比，最终通过计算气温数据与 NDVI 指数的相关系数的方法确定二者间的相关性。

问题四中我们需要分析近年来黄土高原植被变化的主要原因。在问题 3 中，我们分析了气候变化因素对于 NDVI 数据的影响，另外近些年来由于我国大力实施退耕还林(草) 等国家或区域性政策，客观上也对植被生长产生了积极影响。考虑到人类活动与气候变化因素间也有一定的相关性，我们可以考虑在控制人类活动影响的情况下求解温度和降水量关于 NDVI 的偏相关系数，并在控制气候因素的影响后求解植树量关于 NDVI 的偏相关系数，对比两个偏相关系数来确定对植被生长影响较大的因素。

三、 模型假设及符号说明

3.1 模型假设

- 不考虑森林火灾等可能出现的偶然因素对 NDVI 的影响
- 不考虑人类对植被在正常砍伐计划外的破坏对 NDVI
- NDVI 的测量以及气象数据的测量没有明显误差

3.2 符号说明

符号	意义
C	聚类得到的划分
P 和 Q	长度为 n 的时间序列
D	描述路径的向量
$r_{p12(3)}$	控制了变量 X_3 的线性作用，X_1 和 X_2 之间的一阶偏相关系数

四、 问题一

4.1 模型建立

首先我们可以绘制几个象元的的 NDVI 随时间的关系图来分析地区的 NDVI 在整个时间段内的大致变化情况：

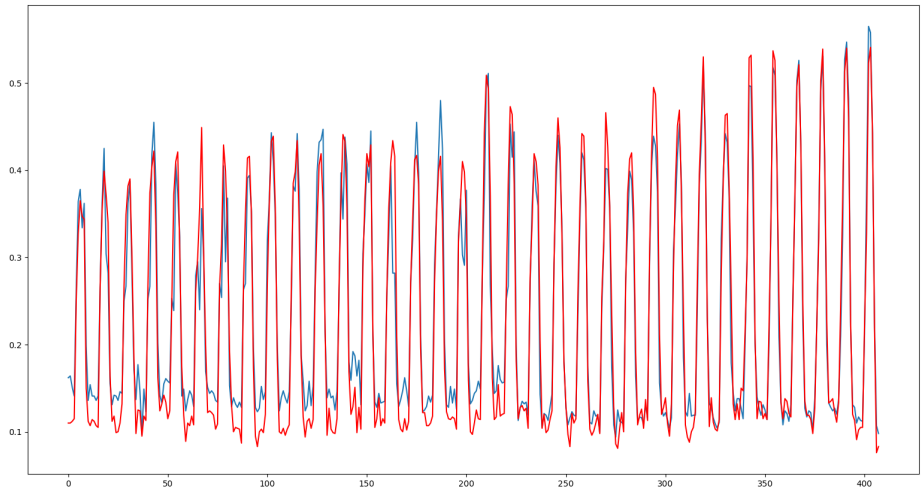


图 1　时间序列 1 和序列 6 的对比图

通过观察上面的关系图我们可以发现：地区的植被覆盖情况呈现逐年上升的趋势，且植被覆盖率有着明显的季节性，每年的春季和夏季的植被覆盖率有明显提升，而冬季由于植物落叶等原因 NDVI 会明显下降。因此在分析地区的植被覆盖情况进行聚类时不能忽略时间因素的影响。

我们可以通过 KMeans 方法进行聚类。并且在聚类过程中使用 DTW 方法计算时间序列间的相似度，这种方法能够较好的处理有相似性单变化趋势不完全相同的时间序列，从而增加聚类计算的精度。

4.1.1 DTW 方法

在实际的时间序列中，经常出现两个时间序列本身在变化趋势上具有相似性而但是具体的发生变化的时间不一致的情况（例如下图中两个时间序列在冬季的 NDVI 变化有一定的相似性，但具体取得最大值的时间有一定差异），在样本的有时间差异性的情况下，直接对比各个特定时间点取值的欧几里德算法不能够较好的解决这类问题。

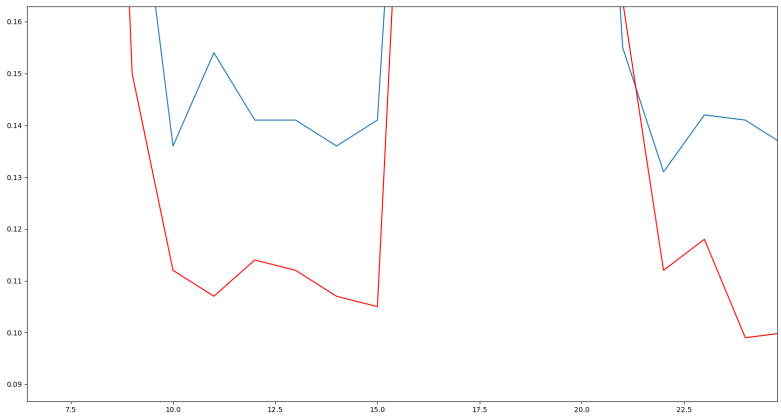


图 2　时间序列 1 和序列 6 的对比图（局部）

我们可以使用论文 [?] 提出的 DTW 距离来实现时间规整。DTW 算法是一种基于动态规划的相似度计算方法。使用 DTW 算法计算距离时，序列某个时刻的点跟另一时刻多个连续时刻的点相对应。DTW 算法能够较好的计算时间序列数据处理可能出现的，

假设有两条长度为 n 的时间序列 $P = \{p_1, p_2, \cdots, p_n\}$ 和 $Q = \{q_1, q_2, \cdots, q_n\}$。我们可以建立一个包含从 $p_i (i \in 1, 2, \cdots, n)$ 到 $q_j (j \in 1, 2, \cdots, n)$ 的闵可夫斯基距离 $dist(i, j)$ 的 $n \times n$ 矩阵。我们希望找到一条路径让矩阵的累积距离最小，从而得到两条时间序列间的最佳匹配。在这里，我们假设路径为 $D = \{d_1, d_2, \cdots, d_k\}$，其中 d_i 代表 $dist(i, j)$ 的最小值。

我们需要使得 D 对应的距离最小，这一目标可以表示为：

$$\min D = \sum_{i=1}^{k} d_i \tag{1}$$

4.1.2 KMeans 聚类

在给定的样本集 $D = x_1, x_2, \cdots, x_n$ 中，K 均值算法希望针对聚类所得的划分 $C = c_1, c_2, \cdots, c_k$ 最小化样本与簇的均值间的误差 E。用 $\mu_i = 1/|c_i| \sum_{x \in c_i}$ 表示簇 c_i 的均值向量。考虑到我们需要对时间序列进行聚类，我们可以使用上文提及的 dtw 距离来代替传统用于计算误差的欧氏距离或者曼哈顿距离，在这种情况下误差 E 可表示为：

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} dtw(x, \mu_i) \tag{2}$$

上式反映了簇内样本围绕均值向量的紧密程度，E 越小则簇内样本的相似性越强。要最大化样本的相似性，找到能够最小化 E 的最优解，我们需要遍历样本 D 的所有可能的划分空间，这种求解方法的时间复杂度较大，我们可以通过下面的迭代优化的方法来近似求解可行的聚类情况：

Algorithm 1: KMeans 聚类算法

Data: D：样本集 $x_1, x_2, \cdots, x_n$，k：聚类簇数

Result: 簇划分 $C: c_1, c_2, \cdots, c_k$

从 D 中选取 k 个样本作为初始向量;
repeat
  令 $c_i = \Phi$;
  foreach $j \in 1, 2, \cdots, m$ **do**
    计算 x_j 与均值向量 μ_i 间的 dtw 距离：$dtw(i, j)$;
    根据相距最近的均值向量来确定 x_j 所属的簇：$\lambda_j = \arg\min_{i \in 1,2,\cdots,k}$;
    $c_{\lambda_j} = c_{\lambda_j} \bigcup x_j$;
  end
  for $i = 1, 2, \cdots, k$ **do**
    计算均值向量:$\mu_i' = \frac{1}{|c_i|} \sum_{x \in c_i} x$;
    if $\mu_i' \neq \mu_i$ **then**
      $\mu_i = \mu_i'$;
    end
    else
      保持均值向量不变;
    end
  end
until 均值向量不再更新;

4.2 模型求解

在模型求解中，我们给出了基于动态规划算法的 DTW 距离的求解方法，并且计算了不同聚类簇数下对应的样本轮廓系数，来判断合理的聚类个数。并使用 KMeans 算法求解了聚类个数为 5 的情况下的聚类。

4.2.1 DTW 距离的求解

我们使用动态规划方法来求解提出的优化问题。假设存在两条长度为 n 的时间序列 $P = \{p_1, p_2, \cdots, p_n\}$ 和 $Q = \{q_1, q_2, \cdots, q_n\}$。在使用欧氏距离计算二者的距离时，总距离可以表示为序列各个位置的距离之和。

在使用 DTW 算法进行计算时，序列 P 的点与另一序列 Q 的多个连续的点相对应，整个路径满足以下性质：

1. 路径长度 = 前一步的路径长度 + 当前元素的大小

2. 对于路径的某个元素 (i, j)，它的前一个元素只可能为以下三者之一：

- 左边的相邻元素 (i, j-1)
- 上面的相邻元素 (i-1, j)
- 左上方的相邻元素 (i-1, j-1)

根据上面的性质，动态规划的状态转移方程可表示为：

$$DTW(i,j) = dist(q_i, c_j) + \min(DTW(i-1, j-1), DTW(i-1, j), DTW(i, j-1)) \quad (3)$$

通过递推得到的最终取值 $[DTW(n,n)]^{\frac{1}{p}}$ 是我们的所需的 DTW 距离，式子中的 p 是闵可夫斯基距离的参数。

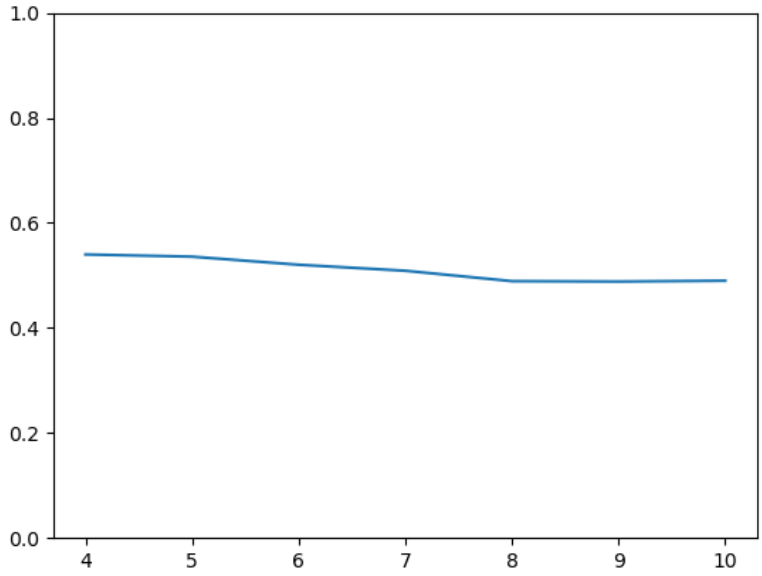
4.2.2 聚类结果

样本轮廓系数（Silhouette Coefficient）是衡量聚类效果好坏的一种评价方式。在计算轮廓系数时，我们首先需要计算簇内不相似度 $a(i)$，它代表 i 向量到同簇内其他点不相似程度的平均值，体现样本的凝聚度。之后计算簇间不相似度 $b(i)$，它代表 i 向量到其他簇的平均不相似程度的最小值，体现样本的分离度。

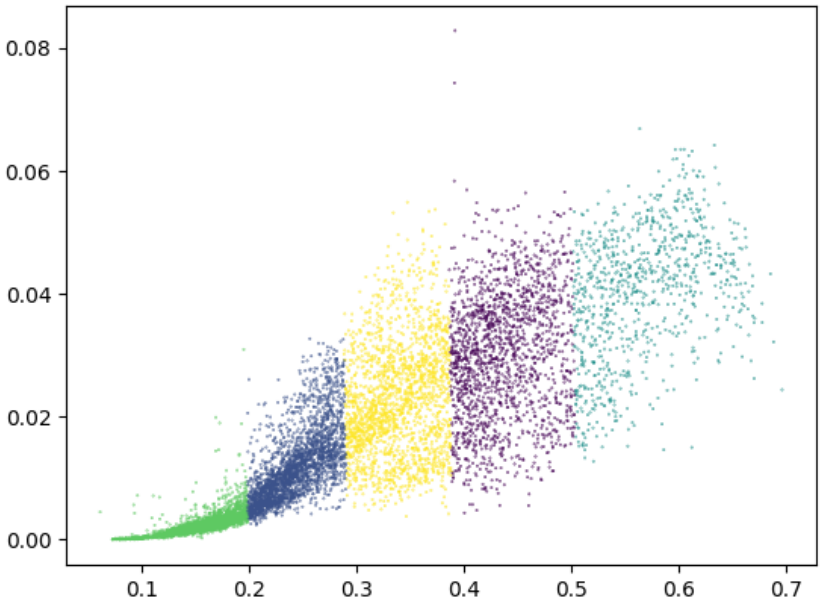
样本的轮廓系数可以表示为：

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad (4)$$

我们使用 KMeans 算法进行时间序列聚类，在聚类簇数为取 4 至 10 时的样本轮廓系数 $S(4), \cdots, S(10)$ 如下图所示：



观察上图可以发现在聚类簇数超过 4 后聚类的样本轮廓系数不再发生明显的改变，即聚类效果在聚类簇数超过 5 后的改变很不明显。因此，在以下的分析中，为了简化问题，我们选取簇数为 5 的聚类结果进行分析。在簇数为 5 时的聚类情况如下：



五、 问题二

在实际情况中，由于地区的植被指数可能受到包括环境的温度、降水量，以及人类活动等多种因素的影响，而且呈现季节性的变化，通过传统的回归方法预测地区的 NDVI 比较困难。我们可以采用指数平滑法来对地区的未来植被指数进行预测。指数平滑法是移动平均法中的一种，其特点在于给过去的观测值不一样的权重，即较近期观测值的权数比较远期观测值的权数要大。

5.1 模型建立

时间序列的变化具有一定的稳定性，所以时间序列的变化趋势可以合理地进行推延。由于近期的变化趋势与未来的变化趋势相关性更大，所以我们将较大的权数放在近期的观测值上。我们可以为近期观测值和历史观测值赋予不同的权重 α，就得到了一次指数平滑的通式：

$$\hat{y}_t = \alpha \cdot y_t + (1-\alpha) \cdot \hat{y}_{t-1} \tag{5}$$

一次指数平滑法能够通过给不同时期的观测值赋予不同的权重的方法来对未来的数据进行预测，能够较好的预测数据值趋于稳定的水平型历史数据。而对于持续增长或

持续下降的数据而言，由于一次指数平滑法没有考虑到数据变化的**趋势性**，因而对持续增长或减少趋势的数据使用这一方法，即使选取较大的参数 α 也仍然会产生较大的误差。

二次指数平滑在此基础上将不同时刻的样本的变化趋势 b_x 作为一个额外参量，保留了趋势的详细信息。即保留并更新平滑后的数据值和平滑后的趋势两个量的状态，并在预测时给予两个量的值给出预测结果。二次指数平滑法的公式如下：

$$\begin{cases} \ell_x & = \alpha y_x + (1-\alpha)(\ell_{x-1} + b_{x-1}) \\ b_x & = \beta(\ell_x - \ell_{x-1}) + (1-\beta) b_{x-1} \\ \hat{y}_{x+1} & = \ell_x + b_x \end{cases} \tag{6}$$

二次指数平滑法虽然三次指数平滑法考虑到了变化趋势的影响，但是没有考虑到数据本身的周期性变化。鉴于我们得到各个象元的的 NDVI 数据呈现明显的季节周期性（例如图1），采取二次指数平滑法进行估计会产生明显的误差。

我们可以采取与二次指数平滑法添加趋势参量 b_x 相似的方法，增加一个季节参量 s_x 来衡量数据的周期性因素对于数据的影响，并在预测时依据数据值、趋势和季节三个参量给出预测结果，从而得到了文献 [**?**] 中的三次指数平滑法。三次指数平滑法的季节表达如下：

$$\begin{cases} \ell_x & = \alpha(y_x - s_{x-L}) + (1-\alpha)(\ell_{x-1} + b_{x-1}) \\ b_x & = \beta(\ell_x - \ell_{x-1}) + (1-\beta) b_{x-1} \\ s_x & = \gamma(y_x - \ell_x) + (1-\gamma) s_{x-L} \\ \hat{y}_{x+m} & = \ell_x + m b_x + s_{x-L+1+(m-1) mod L} \end{cases} \tag{7}$$

在通过三次指数平滑法预测得到样本的 NDVI 在未来的变化情况后，我们可以通过移动平均法，使用目标点前十年内的 NDVI 值来代表目标在某一时刻的 NDVI，从而避免季节因素带来的影响。把不同时间的 NDVI 值通过移动平均法进行处理，可以得到样本在某一段时间的 NDVI 值的变化趋势。

5.2 模型求解

5.2.1 三指数平滑模型的求解

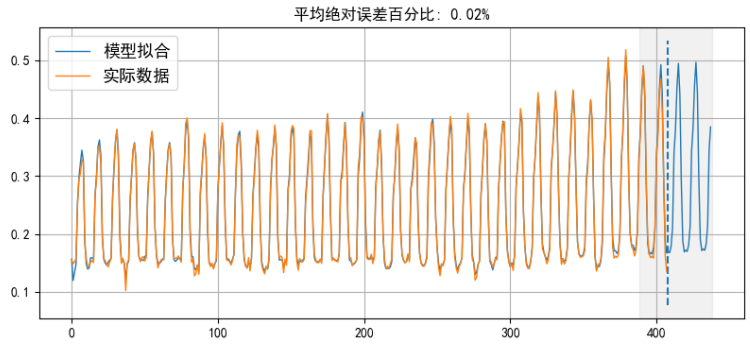
根据公式7，三指数平滑模型包含描述数据值参量的参数 α，描述趋势参量的参数 β，以及描述季节参量的参数 γ。要通过三指数平滑模型确定 NDVI 的变化情况，我们首先需要确定合适的三个参数的取值。

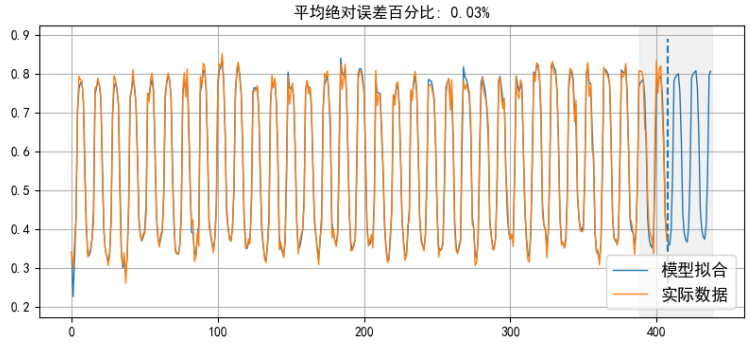
我们可以使用均方误差来表示预测值与真实值的差异情况，并且使用牛顿 CG 方法来获得使得梯度最小的参数取值。此外可以使用时间序列交叉验证的方法来增强参数的可靠性，具体方法如下：

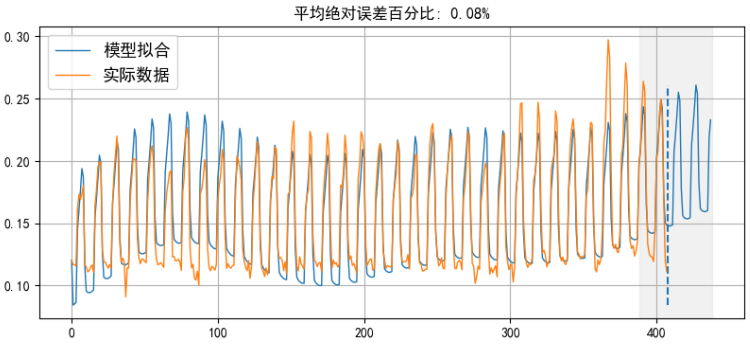
我们可以在时间序列的一小部分上训练模型，从开始的时间节点 t ，对下一个 $t+n$ 时刻进行预测，并计算误差。然后，将训练样本扩展到 $t+n$ 值，从 $t+n$ 做出预测，直到 $t+2*n$ ，并继续移动时间序列的测试段，直到达到最后一次可用的观察。因此，在初始训练样本和最后一次观察之间，可以得到与 n 一样多的数据份数。从而增加了在训练参数时使用的样本总数。

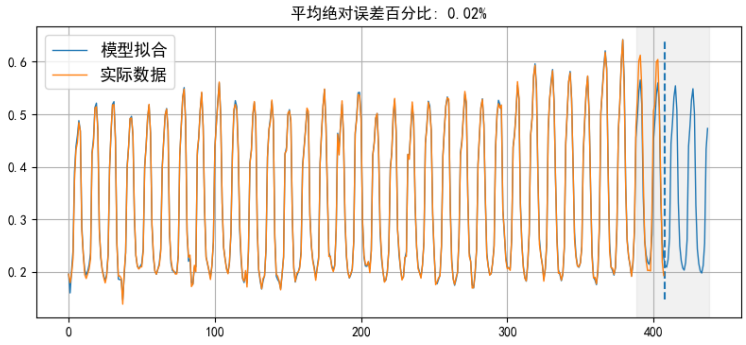
5.2.2 求解结果

我们根据数据，通过三指数平滑法得到的预测结果如下：









在上述预测结果的基础上，我们可以依据移动平滑法求解样本的变化趋势。观察上图，我们的预测算法考虑了到数据的周期性，较好的符合了原数据的上升趋势和季节高峰，能够较好的预测聚类内各个象元的平均 NDVI 值随时间的变化情况。观察图 ()，大多数象元的 NDVI 值呈现上升趋势，这说明黄土高原的大多数地区的植被覆盖面积有增加趋势。

六、 问题三

6.1 模型准备

6.1.1 数据准备

黄土高原大致北起长城，南至秦岭，西抵乌鞘岭，东到太行山，包括山西大部、陕西中北部、甘肃中东部、宁夏南部和青海东部的部分地区。我们选取了陕西省中北部以及山西省的延安、榆林、绥德、吴旗、隰县五个典型观测点从 1984 年到 2016 年的气温和降水数据 [**?**]，并对数据进行了简单的分析和处理。

观测点	月均气温均值	气温均值 (1984-2000)	气温均值 (2001-2016)
延安	10.76	10.15	11.40
榆林	9.60	8.55	10.64
绥德	10.80	10.03	11.56
吴旗	8.89	8.14	9.63
隰县	9.80	9.66	9.91
总体均值	9.97	9.31	10.63

观察上述数据可以发现，各个地区在进入 21 世纪后的平均气温相对 1984-2000 年

的平均气温有平均 1.3 摄氏度的气温上升，数据较好的反映了近年来全球变暖的趋势。我们将 5 个观测点的月平均气温的均值用于下面的相关性分析。

6.1.2 成分分解

假设时间序列 y_t，我们可以使用加法分解进行以下分解：

$$y_t = S_t + T_t + R_t$$

其中，y_t 是数据,S_t 是季节成分，T_t 是趋势成分，R_t 是剩余分量。

我们使用移动平均法估计时间序列的趋势成分 T_t，得到：

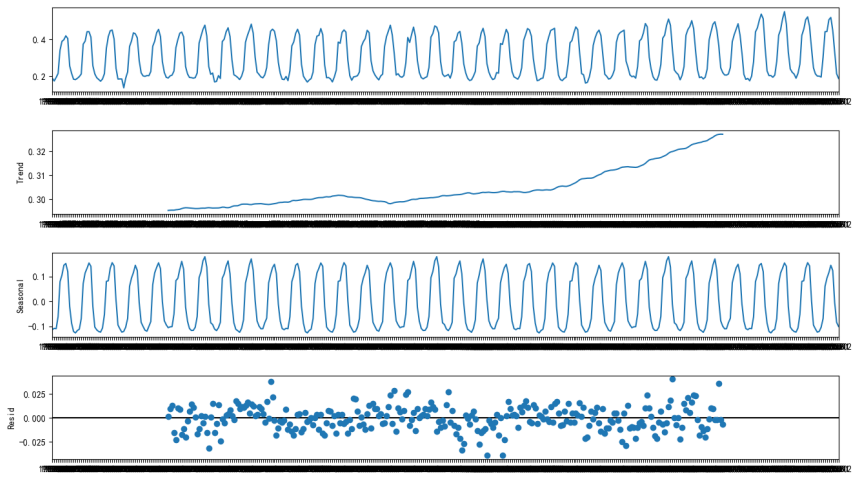
$$\hat{T}_t = \frac{1}{m}\sum_{j=-k}^{k} y_{t+j}$$

其中，$m = 2k + 1$，我们重点研究时间序列的趋势分量 T_t。

令 y_t 为时刻 t 所有象元 NDVI 的平均值，即：

$$y_t = \frac{1}{m}\sum_{i} y_{ti}$$

然后进行分解，令 $k = 60$，得到序列 $\hat{T}_t$，如下面第二张子图所示。



6.2 模型建立与求解

6.2.1 模型建立

在问题三中，我们需要整理样本整体的 NDVI 的均值以及各个聚类内部的 NDVI 的均值，并把这一数据与平均气温数据进行联系。我们可以通过求解各个观测点气温均值与各个象元的 NDVI 之间的相关系数来确定气温与 NDVI 间的相关性，从而确定黄土高原的植被状况与全球变暖的关系。

两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商：

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{8}$$

Pearson 相关系数在样本协方差的基础上除以了两个随机变量的标准差。从而消除了两个随机变量的数值对于相关性的影响。

此外，我们还使用了 Kendall 相关系数和 Spearman 相关系数对样本的相关性进行衡量。

Kendall 系数的定义为，n 个同类的统计对象按特定属性排序，其他属性通常是乱序的。同序对和异序对之差与总对数的比值定义为 Kendall 系数。其中的同序对数代表两个属性排名较大者与排名最大值的差，总对数减去同序对数为异序对数。

Spearman 系数被定义为等级变量的相关系数。我们需要把原始数据按照数值的排名转化为等级数据，并使用与求解 Pearson 系数相似的方法进行求解。

6.2.2 模型求解

我们在针对样本整体，分别求解了分类内部的气温与 NDVI 值。并计算了气温和 NDVI 值的 Pearson 相关系数与 Spearman 相关系数，具体结果如下表：

相关系数	平均	最大
Pearson 相关系数	0.806	0.928
Kendall 相关系数	0.588	0.734
Spearman 相关系数	0.803	0.915

观察上述的相关系数可以发现，在不同的样本上，我们的模型的各种相关系数都大于 0.5，这说明样本的 NDVI 值与气温情况有着明显的正相关关系。这与温度较高，降水量较大地区的植物总生产量较大的生态学规律基本相符。

七、 问题四

问题四中我们要分析黄土高原植被变化的主要原因，我们需要分析各种可能导致黄土高原的 NDVI 变化的原因与 NDVI 值的相关性。为了排除各个因素之间的相关性影响，我们可以采用计算并对比各个因素与植被的偏相关系数的方法来对比各个因素对 NDVI 值的影响情况。

7.1 模型建立

在问题三中，我们分析了气温与 NDVI 值的相关性，并且整理了黄土高原地区的气象观测站的降水数据。但如果希望进一步的分析和对比数据中的相关性信息，考虑到气温，降水，人类活动等因素之间往往有着较大的相关性（比如气温高的地区可能有着更高的降水，气候变化，因素可能促进植树造林相关政策的出台），这客观上使得我们忽略了我们关心的各个要素与 NDVI 之间的相关关系。为了更好的表现各个变量与 NDVI 的相关性，我们需要使用一种能够消除其它变量影响的方法进行相关性分析。

偏相关系数是在多要素所构成的系统中，当研究某一个要素对另一个要素的影响或相关程度时，把其他要素的影响视作常数，即暂时不考虑其他要素影响，单独研究两个要素之间的相互关系的密切程度时所得到的数值结果。

在分析变量 X_1 和 X_2 之间的净相关时，当控制了变量 X_3 的线性作用后，X_1 和 X_2 之间的一阶偏相关系数定义为：

$$r_{p12(3)} = \frac{r_{12} - r_{13}r_{23}}{\sqrt{1 - r_{13}^2}\sqrt{1 - r_{23}^2}} \tag{9}$$

式子中的 r_{12} 代表 X_1 和 X_2 的相关系数，以此类推。容易发现上式排除了 X_1、X_2 和 X_3 的相关关系。

7.2 模型求解

我们求解了温度与降水量在排除另一者影响后与 NDVI 指数的相关系数和偏相关系数，得到的结果如下表：

表 1　参量间的相关系数

方法	变量 1	变量 2	偏相关系数 r_p
Pearson	平均温度	NDVI 指数	0.8063
Pearson	降水量	NDVI 指数	0.3061
Spearman	平均温度	NDVI 指数	0.8027
Spearman	降水量	NDVI 指数	0.3439
Kendall	平均温度	NDVI 指数	0.5879
Kendall	降水量	NDVI 指数	0.2300

表 2　参量间的偏相关系数

方法	变量 1	变量 2	控制量	偏相关系数 r_p
Pearson	平均温度	NDVI 指数	降水量	0.7892
Pearson	降水量	NDVI 指数	平均温度	-0.0753
Spearman	平均温度	NDVI 指数	降水量	0.7783
Spearman	降水量	NDVI 指数	平均温度	-0.1162
Kendall	平均温度	NDVI 指数	降水量	0.5577
Kendall	降水量	NDVI 指数	平均温度	0.0355

我们发现温度，降水和人类的植树造林活动都与 NDVI 成正相关关系，其中气温对 NDVI 指数的影响较大。因而可以通过增加降水量，提高平均气温，在水分允许的条件下植树造林的方式增加 NDVI。具体措施包括：

- 在植树种草的早期使用地膜，改进植物的热量情况
- 通过人工增雨等手段改善当地的降水条件
- 合理进行植树造林等措施

八、 模型评价与推广

8.1 模型优点

- 基于 DTW 算法的聚类过程能够较好的考虑到时间序列上的相似性
- 三次指数平滑法能够较好的考虑到 NDVI 的季节性变化因素
- 在问题三和问题四的分析中有一定的气候数据支撑，能较好的反映相关性

8.2 模型不足

- 基于 DTW 算法的聚类过程的时间复杂度较高。
- 模型缺乏象元的具体地理位置信息，在问题 3 的相关性分析中象元的地理位置可能与观测站的地理位置有一定偏差。

8.3 模型改进

文献 [**?**] 提出了一种基于的符号化时间序列聚类算法对降维后得到的不等长符号时间序列进行聚类的方法。方法首先对时间序列进行降维处理，提取时间序列的关键点，

并对其进行符号化；其次利用方法进行相似度计算；最后进行聚类分析。我们可以通过提取时间序列关键点（如下图）的方法来降低聚类算法的时间复杂度。要判断时间序列中的一个点是否是关键点可以通过时间序列上连续的三个数据形成的夹角和预先设定的阈值之间的关系来进行判断。例如下图中的左图的点是关键点，而右图不是。这种选取关键点的方法能够进行聚类时的算法效率。

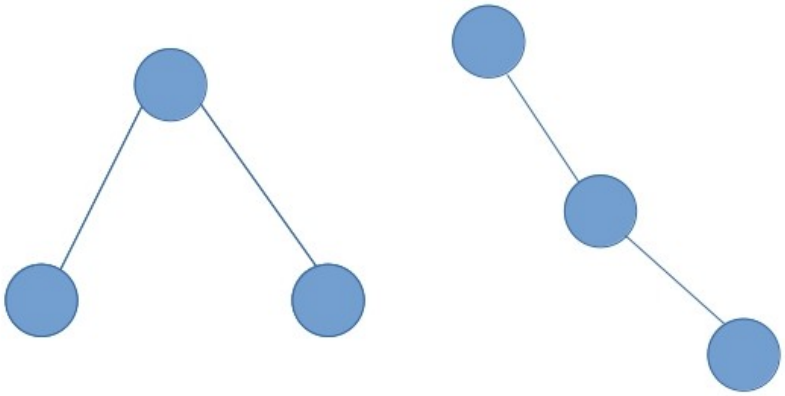


图 3　关键点情况示意

在确定人类活动对植被覆盖变化的贡献情况时，文献 [**?**] 采用了 BP 神经网络来构建构建植被与气候的响应关系，相对我们的残差趋势模型更好的考虑到了降水量等气候因素带来的影响。我们可以在问题四中搜集更多的气候要素并通过构建神经网络的方法进行处理，从而更好的排除自然环境的影响。

参考文献

[1] Liao T W. Clustering of time series data—a survey[J]. Pattern recognition, 2005, 38(11): 1857-1874.

[2] He J , Yang K , Tang W , et al. The first high-resolution meteorological forcing dataset for land process studies over China[J]. Scientific Data, 2020, 7.

[3] 张乐艺，李霞，冯京辉，饶日光，何天英，陈瑜. 黄河流域 NDVI 时空变化及其对气候和人类活动的双重响应 [J/OL]. 水土保持通报:1-11[2021-08-30].https://doi.org/10.13961/j.cnki.stbctb.20210809.001.

[4] 赵安周, 张安兵, 刘海新, 刘焱序, 王贺封, 王冬利. 退耕还林 (草) 工程实施前后黄土高原植被覆盖时空变化分析 [J]. 自然资源学报,2017,32(03):449-460.

[5] 李迎. 时间序列聚类方法研究 [D]. 辽宁师范大学, 2012.

[6] 陈鹏. 1982-2015 年黄土高原植被覆盖变化中气候和人类活动的贡献率研究 [D]. 兰州交通大学,2017.

[7] Maciej S , A Sławomir, Sebastian B , et al. Implementation of Brutlag's algorithm in Anomaly Detection 3.0[C]// Computer Science & Information Systems. IEEE, 2012.

附录 A　KMeans 算法–Python 源程序

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN,KMeans,AgglomerativeClustering,MeanShift
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import silhouette_score
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller

def adf_test(timeseries):
    print ('ADF检验结果:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','Number of Lags
        Used','Number of Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

def calClusterAve(dataf,cluster):
    tar=[]
    for item in cluster:
        tar.append(dataf[item,:])
    tar=np.array(tar)
    ave=np.average(tar,axis=0)
    return ave

dataf=pd.read_csv('聚类样本改.csv')
cols=dataf.columns.values

coll=dataf.shape[0]
rowl=dataf.shape[1]

dataf=np.array(dataf)
means=[]
vars=[]
for item in dataf:
    means.append(np.mean(item))
    vars.append(np.var(item))

means=np.array(means)
vars=np.array(vars)
```

```
x=[]
for i in range(means.shape[0]):
    x.append([means[i],vars[i]])

x=np.array(x)
y_pred=KMeans(n_clusters=5,random_state=9).fit_predict(x)
print(silhouette_score(x,y_pred))
plt.scatter(np.array(x)[:,0],np.array(x)[:,1],s=0.1,c=y_pred)
plt.show()

month=np.array([pd.to_datetime(item+'01') for item in cols])

clunum=5
clusters=[]
aves=[]
for clun in range(clunum):
    clus=[]
    for i in range(coll):
        if y_pred[i]==clun: clus.append(i)
    clusters.append(clus)
    ave=calClusterAve(dataf,clus)
    aves.append(ave)
    #ave=ave[12:len(ave)]-ave[0:len(ave)-12]
    adf_test(ave)
    win=12
    avedf=pd.DataFrame({'ave':ave},index=pd.date_range('1982-01-01', periods=len(ave),
        freq='M'))
    rollingmean=avedf.rolling(window=win).mean()
    decos=seasonal_decompose(avedf['ave'])
    seasonal_decompose(avedf['ave']).plot()
    plt.show()
    plt.close()
    # plt.plot(month[win:],rollingmean[win:],linewidth=2)
    # plt.plot(month[win:], ave[win:], linewidth=1) # [0:396]
    # plt.show()
    # plt.close()
    # plot_acf(ave)
    # plt.show()
    # plt.close()

res=pd.DataFrame(aves)
#res.to_csv('平均数.csv',index=False)

print(y_pred.tolist())
print(1)
```

附录 B　指数平滑分析–Python 源程序

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN,KMeans,AgglomerativeClustering,MeanShift
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import KNeighborsClassifier
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import
    mean_squared_error,mean_squared_log_error,mean_absolute_percentage_error
from scipy.optimize import minimize
from sklearn.model_selection import TimeSeriesSplit

class HoltWinters:
    def __init__(self, series, slen, alpha, beta, gamma, n_preds, scaling_factor=1.96):
        self.series = series
        self.slen = slen
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.n_preds = n_preds
        self.scaling_factor = scaling_factor

    def initial_trend(self):
        sum = 0.0
        for i in range(self.slen):
            sum += float(self.series[i + self.slen] - self.series[i]) / self.slen
        return sum / self.slen

    def initial_seasonal_components(self):
        seasonals = {}
        season_averages = []
        n_seasons = int(len(self.series) / self.slen)
        for j in range(n_seasons):
            season_averages.append(
                sum(self.series[self.slen * j : self.slen * j + self.slen])
                / float(self.slen)
            )
        for i in range(self.slen):
            sum_of_vals_over_avg = 0.0
            for j in range(n_seasons):
                sum_of_vals_over_avg += (
                    self.series[self.slen * j + i] - season_averages[j]
```

```
            )
        seasonals[i] = sum_of_vals_over_avg / n_seasons
    return seasonals

def triple_exponential_smoothing(self):
    self.result = []
    self.Smooth = []
    self.Season = []
    self.Trend = []
    self.PredictedDeviation = []
    self.UpperBond = []
    self.LowerBond = []
    seasonals = self.initial_seasonal_components()
    for i in range(len(self.series) + self.n_preds):
        if i == 0:
            smooth = self.series[0]
            trend = self.initial_trend()
            self.result.append(self.series[0])
            self.Smooth.append(smooth)
            self.Trend.append(trend)
            self.Season.append(seasonals[i % self.slen])
            self.PredictedDeviation.append(0)
            self.UpperBond.append(
                self.result[0] + self.scaling_factor * self.PredictedDeviation[0]
            )
            self.LowerBond.append(
                self.result[0] - self.scaling_factor * self.PredictedDeviation[0]
            )
            continue
        if i >= len(self.series):
            m = i - len(self.series) + 1
            self.result.append((smooth + m * trend) + seasonals[i % self.slen])
            self.PredictedDeviation.append(self.PredictedDeviation[-1] * 1.01)
        else:
            val = self.series[i]
            last_smooth, smooth = (
                smooth,
                self.alpha * (val - seasonals[i % self.slen])
                + (1 - self.alpha) * (smooth + trend),
            )
            trend = self.beta * (smooth - last_smooth) + (1 - self.beta) * trend
            seasonals[i % self.slen] = (
                self.gamma * (val - smooth)
                + (1 - self.gamma) * seasonals[i % self.slen]
            )
            self.result.append(smooth + trend + seasonals[i % self.slen])
            self.PredictedDeviation.append(
```

```
                self.gamma * np.abs(self.series[i] - self.result[i])
                + (1 - self.gamma) * self.PredictedDeviation[-1]
            )
        self.UpperBond.append(
            self.result[-1] + self.scaling_factor * self.PredictedDeviation[-1]
        )
        self.LowerBond.append(
            self.result[-1] - self.scaling_factor * self.PredictedDeviation[-1]
        )
        self.Smooth.append(smooth)
        self.Trend.append(trend)
        self.Season.append(seasonals[i % self.slen])

def timeseriesCVscore(params, series, loss_function=mean_squared_error, slen=12):
    errors = []
    values = series.values
    alpha, beta, gamma = params
    tscv = TimeSeriesSplit(n_splits=3)
    for train, test in tscv.split(values):
        model = HoltWinters(
            series=values[train],
            slen=slen,
            alpha=alpha,
            beta=beta,
            gamma=gamma,
            n_preds=len(test),
        )
        model.triple_exponential_smoothing()
        predictions = model.result[-len(test) :]
        actual = values[test]
        error = loss_function(predictions, actual)
        errors.append(error)
    return np.mean(np.array(errors))

def plotHoltWinters(series, plot_intervals=False, plot_anomalies=False):
    plt.rcParams['font.sans-serif']=['SimHei']
    plt.figure(figsize=(15, 4))
    plt.plot(model.result, linewidth=1, label="模型拟合")
    plt.plot(series.values, linewidth=1, label="实际数据")
    error = mean_absolute_percentage_error(series.values, model.result[: len(series)])
    plt.title("平均绝对误差百分比: {0:.2f}%".format(error))
    if plot_anomalies:
        anomalies = np.array([np.NaN] * len(series))
        anomalies[series.values < model.LowerBond[: len(series)]] = series.values[
            series.values < model.LowerBond[: len(series)]
        ]
        anomalies[series.values > model.UpperBond[: len(series)]] = series.values[
```

```
            series.values > model.UpperBond[: len(series)]
        ]
        plt.plot(anomalies, "o", markersize=10, label="Anomalies")

    if plot_intervals:
        plt.plot(model.UpperBond, "r--", alpha=0.5, label="Up/Low confidence")
        plt.plot(model.LowerBond, "r--", alpha=0.5)
        plt.fill_between(
            x=range(0, len(model.result)),
            y1=model.UpperBond,
            y2=model.LowerBond,
            alpha=0.2,
            color="grey",
        )

    plt.vlines(
        len(series),
        ymin=min(model.LowerBond),
        ymax=max(model.UpperBond),
        linestyles="dashed",
    )
    plt.axvspan(len(series)-20, len(model.result), alpha=0.3, color="lightgrey")
    plt.grid(True)
    plt.axis("tight")
    plt.legend(loc="best", fontsize=13)
    plt.show()

df=pd.read_csv('平均数.csv')
data=np.array(df)
for item in data:
    curdata=pd.DataFrame({'ave':item[:-20]})
    x = [0, 0, 0]
    opt = minimize(
        timeseriesCVscore,
        x0=x,
        args=(curdata, mean_squared_error),
        method="TNC",
        bounds=((0, 1), (0, 1), (0, 1)),
    )
    alpha_final, beta_final, gamma_final = opt.x
    print(alpha_final, beta_final, gamma_final)
    model = HoltWinters(
        np.array(curdata),
        slen=12,
        alpha=alpha_final,
        beta=beta_final,
        gamma=gamma_final,
```

```
        n_preds=50,
        scaling_factor=3,
    )
    model.triple_exponential_smoothing()
    plotHoltWinters(pd.DataFrame(item))
```

附录 C　偏相关系数求解–python 源程序

```
import numpy as np
import pandas as pd
import glob
import matplotlib.pyplot as plt
import math

def genMonthData(resource,month):
    tlist=[]
    for item in resource:
        if int(item[2])==month: tlist.append(item[4])
    while len(tlist)==0:
        month-=1
        for item in resource:
            if int(item[2]) == month - 1: tlist.append(item[4])
    return np.average(tlist)

def calAveByMonth(target,resource):
    for month in range(12):
        temsum=0
        for matr in resource:
            temp=genMonthData(matr,month+1)
            temsum+=temp
        target.append(temp/5)
    return target

pfolderlist=glob.glob('predata/*')
pf0=glob.glob(pfolderlist[0]+'/*.xlsx')
pf1=glob.glob(pfolderlist[1]+'/*.xlsx')
pf2=glob.glob(pfolderlist[2]+'/*.xlsx')
pf3=glob.glob(pfolderlist[3]+'/*.xlsx')
pf4=glob.glob(pfolderlist[4]+'/*.xlsx')
ptotnum=len(pf0)
tfolderlist=glob.glob('tempData/*')
tf0=glob.glob(tfolderlist[0]+'/*.xlsx')
tf1=glob.glob(tfolderlist[1]+'/*.xlsx')
tf2=glob.glob(tfolderlist[2]+'/*.xlsx')
tf3=glob.glob(tfolderlist[3]+'/*.xlsx')
```

```
tf4=glob.glob(tfolderlist[4]+'/*.xlsx')
ttotnum=len(tf0)
ptarget=[]
ttarget=[]
for i in range(ttotnum):
    if '2016' in tf0[i]: break
    df0=np.array(pd.read_excel(tf0[i],engine='openpyxl'))
    df1=np.array(pd.read_excel(tf1[i],engine='openpyxl'))
    df2=np.array(pd.read_excel(tf2[i],engine='openpyxl'))
    df3=np.array(pd.read_excel(tf3[i],engine='openpyxl'))
    df4=np.array(pd.read_excel(tf4[i],engine='openpyxl'))
    calAveByMonth(ttarget,[df0,df1,df2,df3,df4])

for i in range(ptotnum):
    if '2016' in pf0[i]: break
    df0=np.array(pd.read_excel(pf0[i],engine='openpyxl'))
    df1=np.array(pd.read_excel(pf1[i],engine='openpyxl'))
    df2=np.array(pd.read_excel(pf2[i],engine='openpyxl'))
    df3=np.array(pd.read_excel(pf3[i],engine='openpyxl'))
    df4=np.array(pd.read_excel(pf4[i],engine='openpyxl'))
    calAveByMonth(ptarget,[df0,df1,df2,df3,df4])

df=pd.read_csv('聚类样本改.csv')
data=np.array(df).tolist()
tresp=[]
tress=[]
tresk=[]
presp=[]
press=[]
presk=[]

parxt=[]
parxp=[]

sarxt=[]
sarxp=[]

karxt=[]
karxp=[]

for row in data:
    rowseries=pd.Series(row[-384:])
    ttarseries=pd.Series(ttarget)
    ptarseries=pd.Series(ptarget)
    tcorefp=rowseries.corr(ttarseries,method='pearson');tresp.append(tcorefp)
    tcorefs=rowseries.corr(ttarseries,method='spearman');tress.append(tcorefs)
    tcorefk=rowseries.corr(ttarseries,method='kendall');tresk.append(tcorefk)
```

```
    pcorefp=rowseries.corr(ptarseries, method='pearson');presp.append(pcorefp)
    pcorefs=rowseries.corr(ptarseries, method='spearman');press.append(pcorefs)
    pcorefk=rowseries.corr(ptarseries, method='kendall');presk.append(pcorefk)

    corftp=ttarseries.corr(ptarseries,method='pearson')
    parxp.append((pcorefp-tcorefp*corftp)/(math.sqrt((1-tcorefp**2)*(1-corftp**2))))
    parxt.append((tcorefp-pcorefp*corftp)/(math.sqrt((1-pcorefp**2)*(1-corftp**2))))

    corfts = ttarseries.corr(ptarseries, method='spearman')
    sarxp.append((pcorefs - tcorefs * corfts) / (math.sqrt((1 - tcorefs ** 2) * (1 - corfts **
        2))))
    sarxt.append((tcorefs - pcorefs * corfts) / (math.sqrt((1 - pcorefs ** 2) * (1 - corfts **
        2))))

    corftk = ttarseries.corr(ptarseries, method='kendall')
    karxp.append((pcorefk - tcorefk * corftk) / (math.sqrt((1 - tcorefk ** 2) * (1 - corftk **
        2))))
    karxt.append((tcorefk - pcorefk * corftk) / (math.sqrt((1 - pcorefk ** 2) * (1 - corftk **
        2))))

print(np.average(tresp))
print(np.average(presp))
print(np.average(tress))
print(np.average(press))
print(np.average(tresk))
print(np.average(presk))
print(np.average(parxt))
print(np.average(parxp))
print(np.average(sarxt))
print(np.average(sarxp))
print(np.average(karxt))
print(np.average(karxp))
```