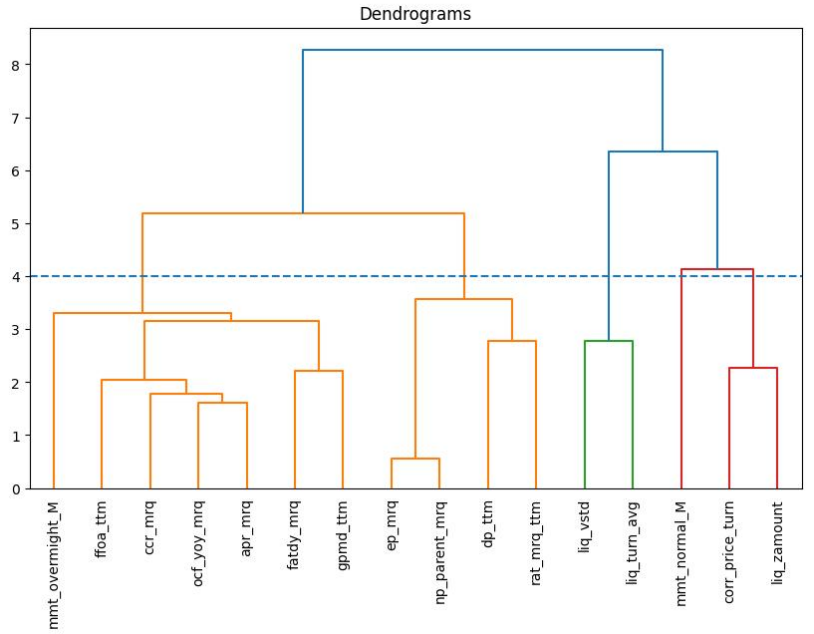
ALPHA因子专题

因子特征聚类
最大化ICIR加权

科大财经



大纲：

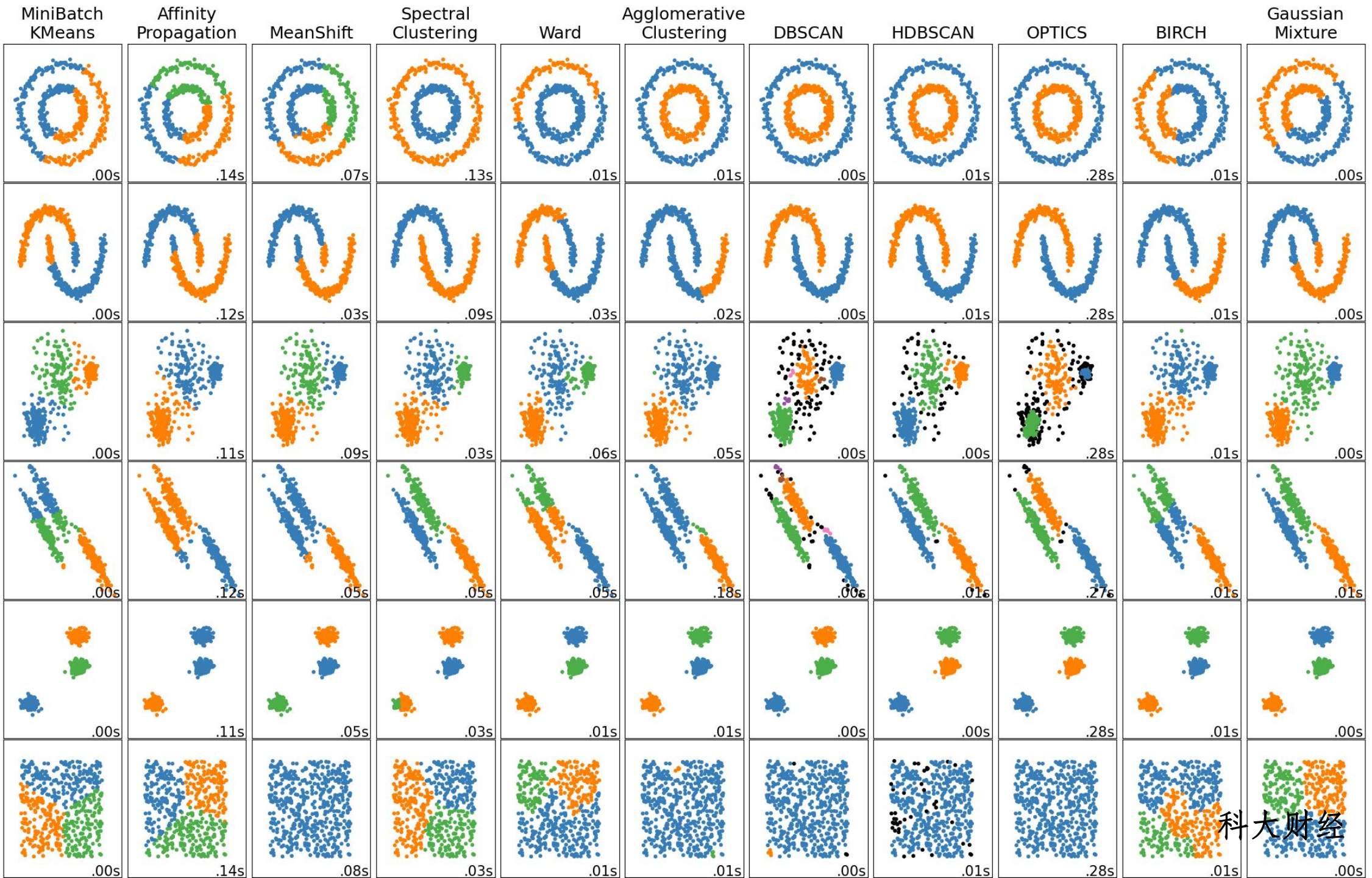
1. 特征聚类方法（Kmeans，图谱聚类，高斯聚类）
2. 聚类性能评估系数（确认最优簇数）
3. 层次聚类的应用&因子自动化合成
4. 因子合成方法优化（动态ICIR加权、最大化ICIR加权）
5. 最大化ICIR加权压缩矩阵估计及二次规划

聚类合成方法

通常我们对于多因子模型解决多重共线性时，一般通过传统的计量方法来判断因子间是否存在共线性问题，常用的方法有相关性检验以及VIF检验。这部分我们在基础班的内容里面已经讲过。高阶班将使用机器学习中常用的无监督学习聚类算法应用从另一个维度解释因子间的共线性问题。

我们的任务会分为以下几步：

1. 解释聚类算法是什么，如何工作?
2. 选择不同的聚类算法对因子进行分类（并对比聚类算法中的区别）
3. 选择不同的聚类评估算法对于聚类结果的稳定性进行确认
4. 引入层次聚类的应用全局理解因子聚类的优势


MiniBatch KMeans
Affinity Propagation
MeanShift
Spectral Clustering
Ward
Agglomerative Clustering
DBSCAN
HDBSCAN
OPTICS
BIRCH
Gaussian Mixture
.00s .14s .07s .13s .01s .01s .00s .01s .28s .01s .00s
.00s .12s .03s .09s .03s .02s .00s .01s .28s .01s .00s
.00s .11s .09s .03s .06s .05s .00s .00s .28s .01s .00s
.00s .12s .05s .05s .05s .18s .00s .01s .27s .01s .01s
.00s .11s .05s .03s .01s .01s .00s .00s .28s .01s .00s
.00s .14s .08s .03s .01s .01s .00s .01s .28s .01s .00s

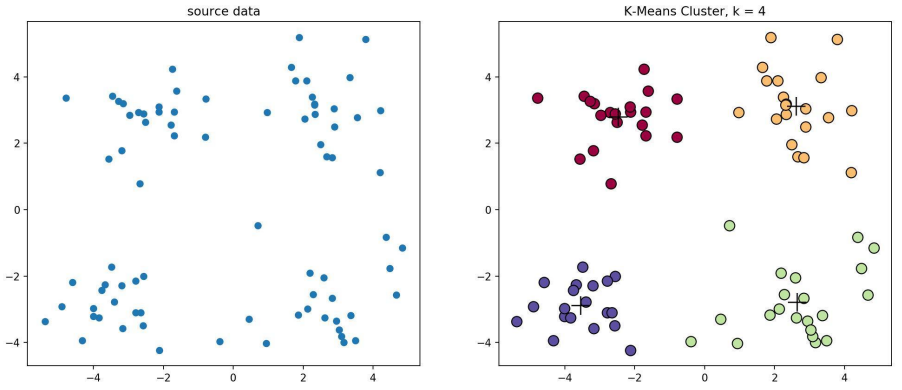
1. 聚类方法（KMEANS，图谱聚类，高斯混合）

聚类（cluster）定义：按照某个特定标准 (如距离) 把一个数据集分割成不同的类或簇，使得同一个簇内的数据对象的相似性尽可能大，同时不在同一个簇中的数据对象的差异性也尽可能地大。也即聚类后同一类的数据尽可能聚集到一起，不同类数据尽量分离。

相似度度量准则	相似度度量函数
Euclidean 距离	$d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$
Manhattan 距离	$d(x,y)=\sum_{i=1}^{n}\|x_i-y_i\|$
Chebyshev 距离	$d(x,y)=\max_{i=1,2,\ldots,n}^{n}\|x_i-y_i\|$
Minkowski 距离	$d(x,y)=[\sum_{i=1}^{n}(x_i-y_i)^p]^{\frac{1}{p}}$

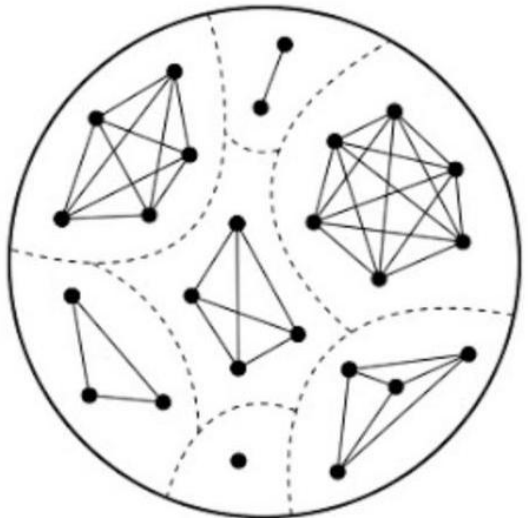
Kmeans聚类流程

1. 创建k个点作为初始质心(通常是随机选择)
2. 当任意一个点的簇分配结果发生改变时
    1. 对数据集中的每个数据点
        1. 对每个质心
            1. 计算质心与数据点之间的距离
        2. 将数据点分配到距其最近的族
    2. 对每个簇，计算簇中所有点的均值并将均值作为质心



谱聚类（SPECTRAL CLUSTERING）

谱聚类（*spectral clustering*）是广泛使用的聚类算法，比起传统的K-Means算法，谱聚类对数据分布的适应性更强。谱聚类是从图论中演化出来的算法，后来在聚类中得到了广泛的应用。**聚类过程**是把所有的数据看做空间中的点，这些点之间可以用边连接起来。距离较远的两个点之间的边权重值较低，而距离较近的两个点之间的边权重值较高，通过对所有数据点组成的图进行切图，**让切图后不同的子图间边权重和尽可能的低，而子图内的边权重和尽可能的高**，从而达到聚类的目的。把聚类问题转化为图分割的问题，转化之后就存在两个问题：（1）数据点与数据点之间的相似度的定义；（2）建立最邻近图谱之后要从哪条边或者从哪些边分割最优。



谱聚类算法的主要**优点**有：

1）谱聚类只需要数据之间的相似度矩阵，因此对于处理稀疏数据的聚类很有效。这点传统聚类算法比如K-Means很难做到。

2）由于使用了降维，因此在处理高维数据聚类时的复杂度比传统聚类算法好。

谱聚类算法的主要**缺点**有：

1）如果最终聚类的维度非常高，则由于降维的幅度不够，谱聚类的运行速度和最后的聚类效果均不好。

2) 聚类效果依赖于相似矩阵，不同的相似矩阵得到的最终聚类效果可能很不同。

科大财经

高斯混合模型（GMM）

2维k-means模型的本质是，它以每个簇的中心为圆心，簇中点到簇中心点的欧氏距离最大值为半径画一个圆。这个圆硬性的将训练集进行截断。而且，k-means要求这些簇的形状必须是圆形的。因此，k-means模型拟合出来的簇（圆形）与实际数据分布（可能是椭圆）差别很大，经常出现多个圆形的簇混在一起，相互重叠。GMM用多个高斯分布函数去近似任意形状的概率分布，所以GMM就是由多个单高斯密度分布（Gaussian）组成的，每个Gaussian叫一个"Component"，这些"Component"线性加和即为 GMM 的概率密度函数。将待聚类的数据点看成是分布的采样点，通过采样点利用类似极大似然估计的方法估计高斯分布的参数，求出参数（用EM算法求解）即得出了数据点对分类的隶属函数.

聚类过程：

1) 设置k的个数，即初始化高斯混合模型的成分个数。（随机初始化每个簇的高斯分布参数（均值和方差）。也可观察数据给出一个相对精确的均值和方差）

2) 计算每个数据点属于每个高斯模型的概率，即计算后验概率。（点越靠近高斯分布的中心，则概率越大，即属于该簇可能性更高）

3) 计算 α,μ,Σ 参数使得数据点的概率最大化，使用数据点概率的加权来计算这些新的参数，权重就是数据点属于该簇的概率。

4) 重复迭代2和3直到收敛。

模型代码

```
num = int(data.shape[0]/2)
for k in range(2,num + 1):
    model = KMeans(n_clusters=k).fit(data)  # 构造聚类器
    y_pred = model.predict(data)
    SSE.append(model.inertia_)                    # estimator.inertia_获取聚类准则的总和
    sil_score.append(silhouette_score(data,y_pred,metric='euclidean'))
    calinski_harabasz_score.append(metrics.calinski_harabasz_score(data,y_pred))          # 越大越好
```

```
from sklearn.cluster import SpectralClustering

for index, gamma in enumerate((0.01,0.1,1,10)):
    for index2, k in enumerate(range(2,num+1,1)):
        y_pred = SpectralClustering(n_clusters=k, gamma=gamma).fit_predict(data)
        print("gamma", gamma, "n_clusters", k,"score:", metrics.calinski_harabasz_score(data, y_pred))
```

```
from sklearn.mixture import GaussianMixture

# calinski_harabasz_score
calinski_harabasz_score = []
for k in tqdm(range(2,num+1)):
    Gaussian = GaussianMixture(n_components=k,random_state=0).fit(data)
    y_pred = Gaussian.predict(data)
    calinski_harabasz_score.append(metrics.calinski_harabasz_score(data,y_pred))          # 越大越好
```

模型代码

```
""" 因子ic序列相关性聚类 """
from sklearn.cluster import KMeans
from sklearn import metrics

def corr_analysis(factor_pair,low,high = 1):
    corr_group = {}
    ## 高相关性分析
    factor_corr = pd.DataFrame(ic_df[factor_pair].dropna().corr().stack())
    factor_corr.columns = ['CORRELATION']
    factor_corr.index.names = ['factor_a','factor_b']
    factor_corr = factor_corr[(factor_corr > low) & (factor_corr < high)].dropna().drop_duplicates()
    data = ic_df[list(set(factor_corr.index.get_level_values(0)) | set(factor_corr.index.get_level_values(1)))].dropna().T

    # calinski_harabasz_score
    calinski_harabasz_score = []
    num = int(data.shape[0]/2) + 1
    try:
        for k in range(2,num):
            model = KMeans(n_clusters=k,random_state=1).fit(data)  # 构造聚类器
            y_pred = model.predict(data)
            calinski_harabasz_score.append(metrics.calinski_harabasz_score(data,y_pred))          # 越大越好

        best = calinski_harabasz_score.index(max(calinski_harabasz_score))+2
        kmeans_cat = KMeans(n_clusters=best,random_state=1).fit(data).predict(data)
        cluster_compoare = pd.DataFrame()
        cluster_compoare.index = data.index
        cluster_compoare['kmeans'] = list(kmeans_cat)
        cluster_compoare = cluster_compoare.sort_values(by = 'kmeans')

        for i in tqdm(range(0,best)):
            corr_group[i] = cluster_compoare[cluster_compoare.kmeans == i].index.tolist()
    except:
        best = 1
        corr_group[0] = data.index.tolist()

    return corr_group,calinski_harabasz_score
```

科大财经

聚类性能评估

Silhouette coefficient（轮廓系数）

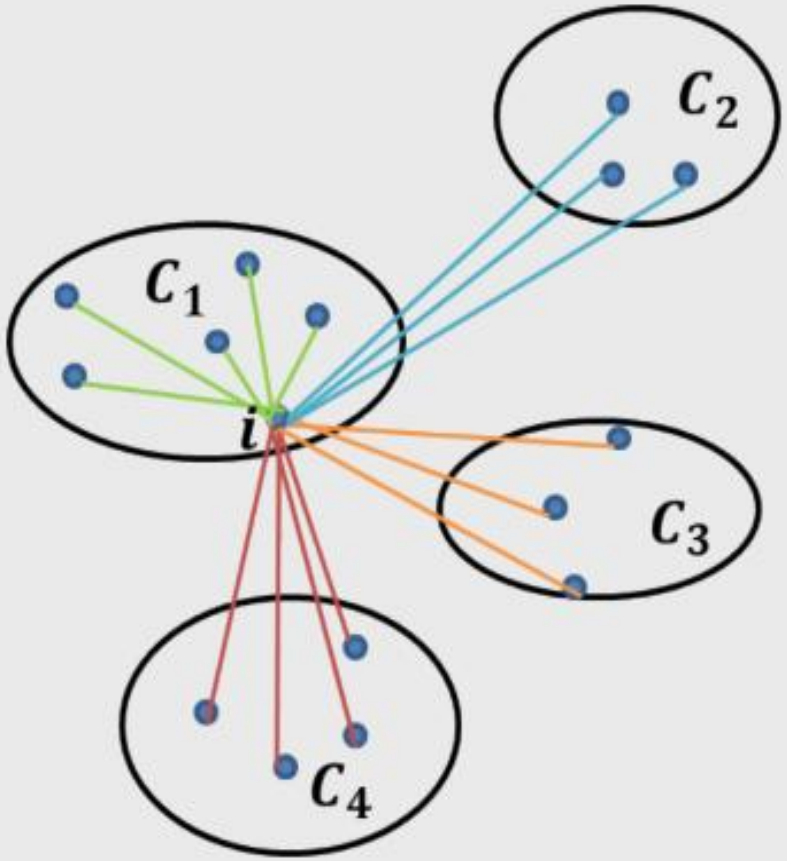
a(i)：本簇中到其它所有样本点的距离的平均

b(i)：到其它簇的所有样本点的平均距离的最小值

$$s(i)=\frac{b(i)-a(i)}{\max\{a(i),b(i)\}} \quad or \quad s(i)=\begin{cases}1-\frac{a(i)}{b(i)} & if \quad a(i)<b(i)\\ 0 & if \quad a(i)=b(i)\\ \frac{b(i)}{a(i)}-1 & if \quad a(i)>b(i)\end{cases}$$

$$-1 \le s(i) \le 1 \qquad \bar{s}=\frac{1}{n}\sum_{i=1}^{n}s(i)$$

指标值越大证明聚类效果越好



假设数据集被拆分为4个簇，样本i对应的$a(i)$值就是所有C_1中其他样本点与样本i的距离平均值；样本i对应的$b(i)$值分两步计算，首先计算该点分别到C_2、C_3和C_4中样本点的平均距离，然后将三个平均值中的最小值作为$b(i)$的度量。

聚类性能评估

簇内平方和 inertia

而将一个数据集中的所有簇的簇内平方和相加，就得到了整体平方和(Total Cluster Sum of Square)，又叫做total inertia，TSSE。Total Inertia越小，代表着每个簇内样本越相似，聚类的效果就越好。因此KMeans追求的是，求解能够让Inertia最小化的质心。

$$Cluster\ Sum\ of\ Square\ (CSS) = \sum_{j=0}^{m}\sum_{i=1}^{n}(x_i - \mu_i)^2$$

$$Total\ Cluster\ Sum\ of\ Square = \sum_{l=1}^{k} CSS_l$$

Calinski-Harabaz（CH）

CH指标通过计算类中各点与类中心的距离平方和来度量类内的紧密度，通过计算各类中心点与数据集中心点距离平方和来度量数据集的分离度，CH指标由分离度与紧密度的比值得到。从而，CH越大代表着类自身越紧密，类与类之间越分散，即更优的聚类结果。

组内离散度：

$$W(K) = \sum_{k=1}^{K}\sum_{C(j)=k} ||\mathbf{x_j} - \overline{\mathbf{x}}_\mathbf{k}||^2$$

类内离散度：

$$B(K) = \sum_{k=1}^{K} a_k ||\overline{\mathbf{x}}_\mathbf{k} - \overline{\mathbf{x}}||^2$$

CH评估值：

$$CH(K) = \frac{B(K)(N-K)}{W(K)(K-1)}$$

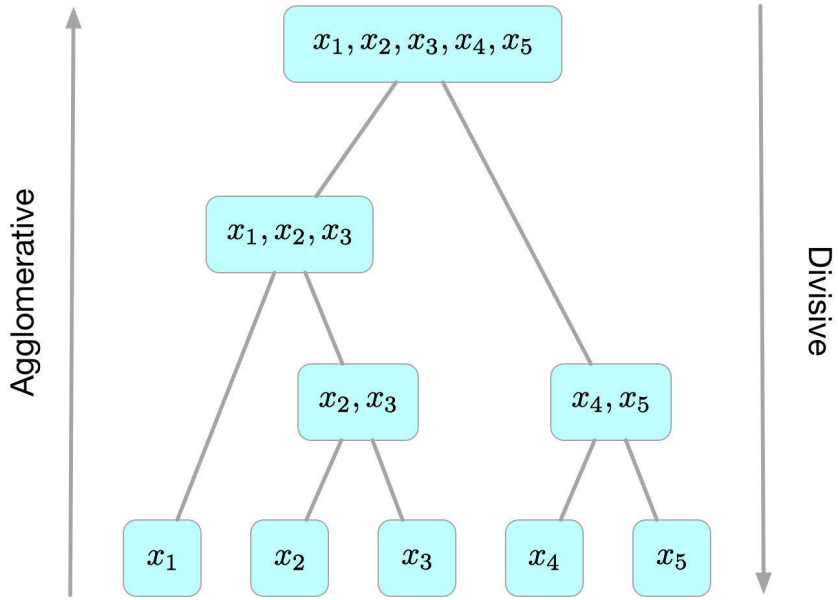
科大财经

层次聚类算法

层次聚类算法 (hierarchical clustering) 将数据集划分为一层一层的 clusters，后面一层生成的 clusters 基于前面一层的结果。层次聚类算法是一种贪心算法（*greedy algorithm*），因其每一次合并或划分都是基于某种局部最优的选择。

» Divisive 层次聚类：又称自顶向下（top-down）的层次聚类，最开始所有的对象均属于一个 cluster，每次按一定的准则将某个 cluster 划分为多个 cluster，如此往复，直至每个对象均是一个 cluster。

» Agglomerative 层次聚类：又称自底向上（bottom-up）的层次聚类，每一个对象最开始都是一个 cluster，每次按一定的准则将最相近的两个 cluster 合并生成一个新的 cluster，如此往复，直至最终所有的对象都属于一个 cluster。我们主要关注此类算法。



科大财经

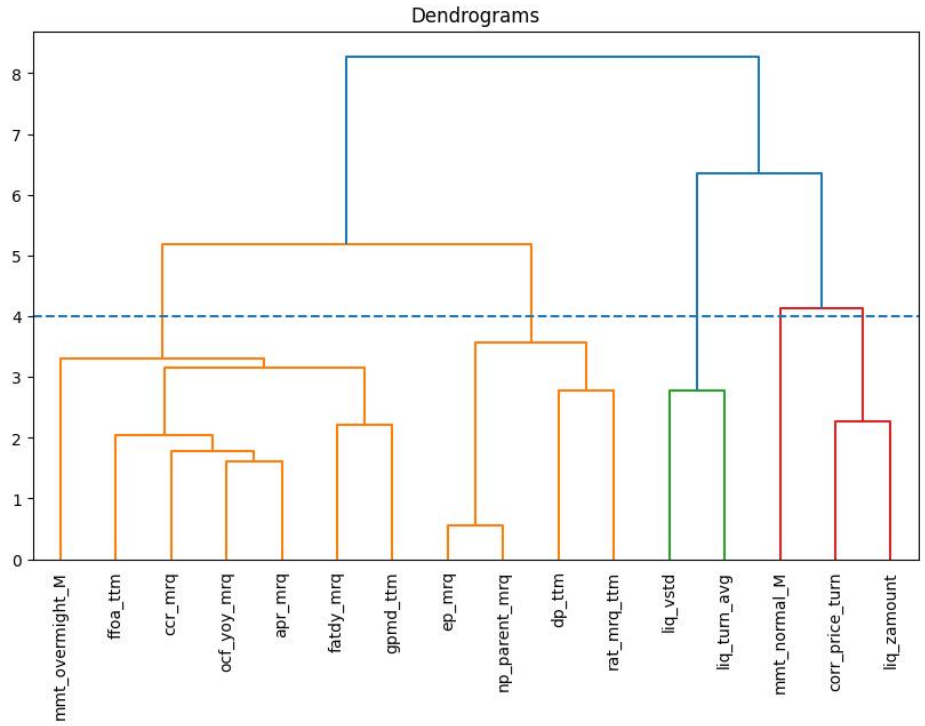
AGGLOMERATIVE层次聚类算法

能够从全局帮助我们了解到因子和因子之间的关系，省去了先切分后聚类时，对于切分点的寻找。

1.初始时每个样本为一个 cluster，计算距离矩阵 D，其中元素D为样本点D和 D之间的距离。

2.遍历距离矩阵 D，找出其中的最小距离(对角线上的除外)，并由此得到拥有最小距离的两个 cluster的编号，将这两个 cluster 合并为一个新的 cluster 并依据 cluster 距离度量方法更新距离矩阵D(删除这两个 cluster 对应的行和列，并把由新 cluster 所算出来的距离向量插入 D中)，存储本次合并的相关信息

3. 重复2的过程，直至最终只剩下一个 cluster



簇内因子自动合成

```python
if len(corr_group[0]) != 0 :
    for i in list(corr_group.keys()):
        print(f'{corr_group[i]} 进入合成队列')
        del_list = []
        # 中高相关组内因子合成
        orth_df = orth_quick(corr_group[i],factor_dict,value_dict_ = True)
        orth_df = data_clean(orth_df)
        df,ic = Quick_Factor_Return_N_IC(orth_df,20,'orth')

        #对比原有因子 （如果复合因子强，则保留复合因子）
        corr_group_compare = pd.concat([compare.loc[list(set(corr_group[i])),'IR'],ic.set_index('name').IR],axis = 0).sort_values(ascending
        = False)
        if corr_group_compare.abs().sort_values(ascending=False).index[0] == 'orth':
            factor_dict[",".join(corr_group[i])] = orth_df
            ic_df[",".join(corr_group[i])] = df
            del_list = corr_group_compare.index[1:].tolist()
            for j in del_list:
                try:
                    pass_icir_factor.remove(j)
                except:
                    pass
            pass_icir_factor.append(",".join(corr_group[i]))
        else:
            del_list = corr_group_compare.index[1:].tolist()
            del_list.remove('orth')
            for j in del_list:
                try:
                    pass_icir_factor.remove(j)
                except:
                    pass

        # 高相关性下保留表现较好的因子的因子
        print('{}因子值中选择{}'.format(corr_group_compare.to_dict(),corr_group_compare.index[0]))

    # 因子过滤
    compare = ic_analysis(ic_df[pass_icir_factor])
else:
    print('无中相关项')
    pass
```

Python

科大财经

因子合成（最大化IC_IR加权法）

20190104-华泰证券-华泰多因子系列之十：因子合成方法实证分析

最大化 IC_IR 加权法

Qian 在《Quantitative Equity Portfolio Management》一书中提出最大化复合因子 IC_IR 的方法。其基本思想是，以历史一段时间的复合因子平均 IC 值作为对复合因子下一期 IC 值的估计，以历史 IC 值的协方差矩阵作为对复合因子下一期波动率的估计，根据 IC_IR 等于 IC 的期望值除以 IC 的标准差，可以得到最大化复合因子 IC_IR 的最优权重解。以$\vec{w} = (w_1, w_2, \dots, w_N)^T$表示因子合成时所使用的权重，$\overrightarrow{IC} = (\overline{ic_1}, \overline{ic_2}, \dots, \overline{ic_N})^T$表示因子 IC 均值向量，其中$\overline{ic_k}(k = 1,2, \dots, N)$表示第 k 个因子在历史一段时间内的 IC 均值，$\sum$为因子 IC 的协方差矩阵。则最优化复合因子 IC_IR 的问题可以表示为：

$$\max IC\_IR = \frac{\vec{w}^T * \overrightarrow{IC}}{\sqrt{\vec{w}^T \sum \vec{w}}}$$

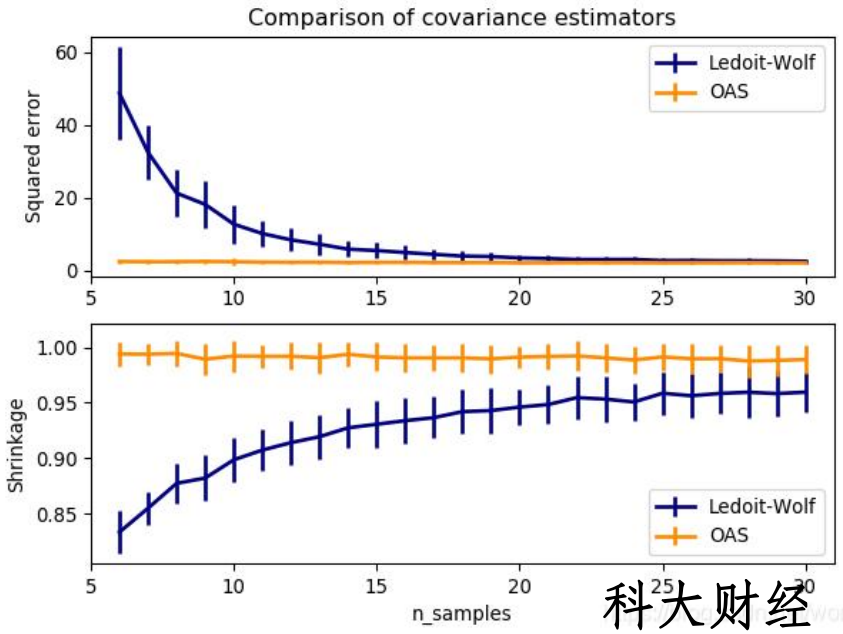
上述优化问题具有显式解$\vec{w} = \sum^{-1} * \overrightarrow{IC}$，对计算出的$\vec{w}$需进行归一化。实际上，我们仍然使用因子的 RankIC 而非简单 IC（Pearson IC）参与上述计算，后文中若未明确指出，则所有的 IC 均指代 RankIC。

科大财经

压缩矩阵估计协方差矩阵

该方法在运用中值得注意的有两点。首先，对协方差矩阵的估计常常有偏差。统计学中以样本协方差矩阵代替总体协方差矩阵，但在样本量不足时，样本协方差矩阵与总体协方差矩阵差异过大，另外估计出的协方差矩阵可能是病态的，造成上述优化问题难以求解。因此，在求解权重的过程中，协方差矩阵的估计也是一个重要的问题。

通常的协方差最大似然估计能够使用缩水(shrinkage)的方法正则化。Ledoit and Wolf在2004年提出，通过最小化MSE的准则，计算渐近最优的缩水参数，产生了以他们的名字命名的Ledoit-Wolf协方差估计法。chen等人进一步在2010年提出了Ledoit-Wolf缩水参数的改进，即，OAS系数，在正态假设下，它的收敛性更好。



科大财经

线性回归（约束权重）

```python
# 最大化动态ICIR加权(压缩矩阵)

import numpy as np
from scipy.optimize import minimize

def maxir_weight(Ic_df_r,cov_matrix):
    # 定义目标函数
    def objective(x):
        return -(x.dot(np.array(Ic_df_r)))/(np.sqrt(x.dot(cov_matrix).dot(x.transpose())))

    # 定义初始值
    x0 = np.array([[1]*Ic_df_r.shape[0]])

    # 求解优化问题
    result = minimize(objective,x0,method='SLSQP',bounds=[[0,None]]*Ic_df_r.shape[0])

    # 输出结果
    return result.x
```

其次，因协方差矩阵估计不准确或存在其它干扰因素，由显式解解出的权重常常出现负数，这与因子本身的逻辑相反，违反了因子的实际意义。我们推荐直接求解上述优化问题，并加上权重为正的约束条件，即求解以下优化问题：

$$\max IC\_IR = \frac{\vec{w}^T * \overrightarrow{IC}}{\sqrt{\vec{w}^T \sum \vec{w}}}$$

$$s.t. \qquad \vec{w} \geq 0$$

回测函数（含交易成本）

```
1  BACKTEST?
✓ 0.0s
```

```
Signature:
BACKTEST(
    df,
    name='',
    n=50,
    change_days=5,
    tax=0.001,
    commission=0.0002,
    benchmark='000906.XSHG',
    fix=False,
    limit_n=1,
    filter=False,
)
Docstring:
:param df: 因子值 -> unstack
:param name: 因子名称 -> str
:param n: 最多选取因子值最大的n只标的 -> int
:param change_days: 调仓周期 -> int
:param tax: 印花税（默认千一） -> float
:param commission: 交易佣金（默认万二） -> int
:param benchmark: 基准 -> str
:param fix: 是否修正买入队列 -> bool
:param limit_n: 数据修正填充数量 -> int
:param filter: 是否过滤ST涨停停牌 -> bool
:return net: 净值序列 -> dataframe
:return performance: 绩效分析 -> dataframe
File:      c:\users\nick_ni\documents\nick_ni\courses\课程\专题课程\2. alpha因子专题\因子合成方法研究\init.py
Type:      function
```