```python
import json
import math
import random
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from itertools import *


#裂类个数确定算法 1，依据各种类的平均值，按概率确定裂类数，效果不是很理想
def divide_cluster_num(node_term, cluster_aver):
    more_than = -1  #保存大于或小于 temr 数的类的索引
    less_than = -1  ###########有问题
    more_temp = 0  #保存确定大于的 term 数
    less_temp = INFINITE  # 保存确定小于的 term 数
    for i in range(len(cluster_aver)):
        if len(node_term) > cluster_aver[i] and more_temp < cluster_aver[i]:
            more_temp = cluster_aver[i]
            more_than = i
        elif len(node_term) < cluster_aver[i] and less_temp > cluster_aver[i]:
            less_temp = cluster_aver[i]
            less_than = i
```

```
    if more_temp == 0:   #表示在 term 数在最小区间
        more_likely = 0  #即裂类数为最小数的概率为 1
        less_likely = 1
        return less_than
    elif less_temp == INFINITE:
        more_likely = 1  # 即裂类数为最大数的概率为 1
        less_likely = 0
        return more_than
    else:
        more_likely = 1-(len(node_term) - more_temp)/(less_temp-more_temp)  #即根据距离确定属于哪个裂类数的概率
        less_likely = 1 - more_likely
        ran = random.random()
        if ran <= more_likely:
            return more_than
        else:
            return less_than

#裂类个数确定算法 2，依据到各个实例的距离，确定裂类个数，效果较好一点
def divide_cluster_num2(node_term, nodes, nodes_cluster, pos):
    temp = INFINITE
    temp_cluster = -1
    for i in nodes:
        if i['pos'] <= pos:
            if abs(len(node_term)-len(i['key'])) < temp:
                temp = abs(len(node_term)-len(i['key']))
                temp_cluster = nodes_cluster[nodes.index(i)]
```

```
    return temp_cluster

#从 terms 集中找到 term 的下标
def find_idx(name, terms):
    for i in terms:
        if name == i['t']:
            return i['idx'] - 1
    return -1

#计算单术语到结点术语集的距离
def dist_to_cluster(solo_term, poly_terms):
    aver_dist = 0
    for i in poly_terms:
        aver_dist += dist[find_idx(solo_term,terms)][find_idx(i,terms)]/len(poly_terms)
    return aver_dist


#KMEAN++对单个结点裂类,max_dist 是在选择中心时计算中心之间的距离，min_dist 是在将术语并入类时的距离
def split_cluster(number, key, dist, ch_rate, ch_dist):
    #KMEAN++方法初始化聚类中心
    cluster = ['' for i in range(number)]
    cluster[0] = (key[0]['term'])  #初始化第一个中心
    for i in range(1,number):  #挨个对其他中心初始化,找剩余术语中离已选择中心最远的
        max_dist = [0 for j in range(len(key))]
        for j in range(len(key)):   #遍历术语,找术语到已知中心最近距离
            max_dist[j] = INFINITE
```

```
            for i2 in range(0,i):
                if dist[find_idx(key[j]['term'],terms)][find_idx(cluster[i2],terms)] < max_dist[j]:
                    max_dist[j] = dist[find_idx(key[j]['term'],terms)][find_idx(cluster[i2],terms)]
        cluster[i] = key[max_dist.index(max(max_dist))]['term']
    print("init cluster center:",cluster)
    # KMean 迭代，到质心的距离看做是到类内各个点的平均距离
    new_cluster = [[i] for i in cluster]
    for round in range(ITER_TIME):
        sort = [0 for i in range(len(key))]
        for i in key:
            min_dist = INFINITE
            for j in range(number):
                if dist_to_cluster(i['term'],new_cluster[j]) < min_dist:
                    min_dist = dist_to_cluster(i['term'],new_cluster[j])
                    sort[key.index(i)] = j
        new_cluster = [[] for i in range(number)]
        for i in range(len(key)):
            new_cluster[sort[i]].append(key[i]['term'])
    return new_cluster

#结点间变异，term_tuple 为一个三层嵌套数组，[ [[],[],[]], [[],[]] ], 最里层为一个裂子类，第二层为一个结点，最外层为一个年代
def change_term(term_tuple, ch_rate, ch_dist, terms):
    for kkk in term_tuple:
        for i in kkk:  #对元组中每个裂类结点遍历术语集
            for j in i:
                ran = random.random()
```

```python
            if ran < ch_rate[find_idx(j, terms)]:
                #变异,temp_dist 保存该结点到所有结点的变异距离,根据相关性，对距离取倒数再用 exp 函数修改梯度
                ran_dist = ch_dist[find_idx(j, terms)]
                for every in ran_dist:
                    every = math.exp(1/every)
                ran_prob = [0 for i in range(TERM)]
                ran_prob[0] = ran_dist[0]/sum(ran_dist)
                for k in range(1,TERM):
                    ran_prob[k] = ran_dist[k]/sum(ran_dist) + ran_prob[k-1]
                    #选择变异为哪一个
                ran2 = random.random()
                for k in range(TERM):
                    if ran2 <= ran_prob[k]:
                        print(i)
                        print(j)
                        i.remove(j)
                        for kkk2 in term_tuple:
                            for i2 in kkk2:
                                if terms[k]['t'] in i2:
                                    i2.remove(terms[k]['t'])
                                    i2.append(j)
                        i.append(terms[k]['t'])
                        break
    return term_tuple
```

```
TERM = 99 #术语数量
NODE = 40 #结点数量
YEAR = 39 #年代数
INFINITE = 1000000000000 #无穷大代称
ITER_TIME = 1000
NODE_SHRINK_TIMES = 4  #结点大小在可视化中的收缩倍数
MERGE_TIMES = 20 #子类合并时需要保证小于的距离倍数
DIST_TRAIN_RATE = 0.8 #训练时每次距离更新的倍数
PENALTY = [math.log(i,10) for i in range(1,11)]

f = open(r"C:\Users\lenovo\Desktop\实习项目\2017 8-13 technicaltrend\带梯度处理\technicaltrend\technicaltrend\WebContent\data\trend_out.json", encoding='utf-8')
record = json.load(f)

documents = record['documents']
terms = record['terms'] #对 terms 按索引号排序
terms = sorted(terms, key=lambda x:x['idx'] )
people = record['people']
time_slides = record['time_slides']
nodes = record['nodes']
links = record['links']
for i in links: #对 link 按年代分组排序,且统计结点的裂类个数,
    i['src_year'] = nodes[i['source']]['pos']
    i['tar_year'] = nodes[i['target']]['pos']
links.sort(key = lambda x:(x['src_year'],x['tar_year']))
for m,n in groupby(links, key = lambda x:(x['src_year'],x['tar_year'])):
    print(m)
```

```
    print(list(n))
print(links)


# 初始化 #
########
#year_node 将所有的结点按年代划分
year_node = [[],[],[],[],[],[],[],[],[],[],[]]
print(year_node)
for i in nodes:
    year_node[i['pos']].append(i)
#术语之间的 dist，取值 0~1，决定裂类和聚类的概率，它是对称矩阵
dist = [[1 for i in range(TERM)] for j in range(TERM)]
for i in range(TERM):
    dist[i][i] = 0
#演化度 ch_rate，取值 0~1，决定术语变异的概率
ch_rate = [0.01 for i in range(TERM)]
#变异距离 ch_dist，他是非对称矩阵
ch_dist = [[1 for i in range(TERM)] for j in range(TERM)]
for i in range(TERM):
    ch_dist[i][i] = INFINITE
#术语大小 value，即趋势图数据
term_year = [i for i in range(YEAR)]
value = [[0 for i in range(YEAR)] for j in range(TERM)]
for i in terms:
    for j in range(YEAR):
```

```
        value[i['idx']-1][j] = i['year'][j]['d']
print(value)
#从技术正式开始兴起，即值不为0时开始拟合，得设置偏移！！！！使用罚函数来拟合曲线，1974年到2012年，1次到10次函数
offset = [0 for i in range(TERM)]
for i in terms:
    temp = YEAR #表示没找到不为0的点
    for j in range(YEAR):
        if i['year'][j]['d'] != 0:
            temp = j
            break
    offset[i['idx']-1] = temp
value_for_fitting = [[] for i in range(TERM)]
for i in range(TERM):
    for j in range(offset[i],YEAR):
        value_for_fitting[i].append(value[i][j])
print(value_for_fitting)


#对每个术语趋势岭回归,fit_parameter保存对每个术语的拟合曲线参数，esti_offset表示要预测多少个点，即排除后面多少个点
fit_parameter = [[] for i in range(TERM)]
esti_offset = 0
for i in range(TERM):
    x = np.linspace(1, YEAR - offset[i] + 1, YEAR - offset[i])
    y = np.array(value_for_fitting[i])

    x_train = np.linspace(1, YEAR - offset[i] + 1 - esti_offset, YEAR - offset[i] - esti_offset)
```

```
    y_train = np.array(value_for_fitting[i][:YEAR - offset[i] - esti_offset])
    X_TRAIN = x_train[:, np.newaxis]
    model = Pipeline([('poly',PolynomialFeatures(5)),('ridge',Ridge(alpha=10000000000000000*PENALTY[5]))])  # 使用岭回归来进行多项式的特征输出,PENALTY 表示惩罚度,依据训练好坏选择惩罚度
    model.fit(X_TRAIN, y_train)
    fit_parameter[i] = list(model.named_steps['ridge'].coef_)
    fit_parameter[i].reverse()   #将系数按幂数降序排列

    #显示预测效果
    # x_esti = np.linspace(1, YEAR - offset[i] + 1, YEAR - offset[i])
    # X_ESTI = x_esti[:, np.newaxis]
    # y_esti = model.predict(X_ESTI)
    # plt.plot(X_ESTI+1973, y_esti, label="degree %d" % 10)
    # plt.scatter(X_ESTI+1973, y, label="training points" + terms[i]['t'])#画出散点图
    # plt.legend(loc='lower left')  # 画出画线标签
    # plt.show()

#训练裂类个数，这里对每种取平均值,判断它在哪个区间内
link_sum = [0 for i in range(40)]
cluster_termnum = [0 for i in range(4)]
cluster_nodenum = [0 for i in range(4)]
for i in links:
    link_sum[i['source']] += 1
print(link_sum)
for i in range(40):
    if link_sum[i] != 0:
```

```
        cluster_termnum[link_sum[i]-1] = (cluster_termnum[link_sum[i]-1]*cluster_nodenum[link_sum[i]-1] + len(nodes[i]['key']))/(cluster_nodenum[link_sum[i]-1] + 1)
        cluster_nodenum[link_sum[i]-1] += 1
print(cluster_termnum)
print(cluster_nodenum)
for i in range(40):
    print((divide_cluster_num(nodes[i]['key'],cluster_termnum) +1),end=" ")
print()
for i in range(30,40):
    print((divide_cluster_num2(nodes[i]['key'],nodes, link_sum, 6) ),end=" ")

#训练部分，训练术语间距离，变异概率，变异距离
for m,n in groupby(links, key = lambda x:(x['src_year'],x['tar_year'])):
    print(m)
    print(list(n))

set1 = [] #set1 存放临时的 source 术语集
set2 = [] #set2 存放临时的 target 术语集
set3 = [] #set3 存放临时的 source 术语集中正常继承的部分
mutate = [] #mutate 存放临时的 source 术语集中变异的部分
for j in range(9):  # 按年代进行训练
    for i in nodes:
        if i['pos'] == j:
            set1 = []  # 清空 set1
            for k1 in i['key']:
                set1.append(k1['term'])
            set3 = []
```

```
for k in links:   #对 list 里面每个元素遍历
    if nodes[k['source']] == i:  #找到对应年代的结点和连接
        set2 = [] #清空 set2
        for k2 in nodes[k['target']]['key']:
            set2.append(k2['term'])
        for term_item in set2:
            for term_item2 in set2:
                if term_item != term_item2:
                    term_idx1 = 0
                    term_idx2 = 0
                    for term in terms:
                        if term_item == term['t']:
                            term_idx1 = term['idx'] - 1
                        if term_item2 == term['t']:
                            term_idx2 = term['idx'] - 1
                    dist[term_idx1][term_idx2] *= DIST_TRAIN_RATE   #即每次训练，在同一个子类的术语间距离变为 DIST_TRAIN_RATE 倍，结合时距离进一步减小
        set3 = set(set3) | (set(set1) & set(set2))
mutate = set(set1) - set(set3)
for term_mutate in mutate:
    mutate_idx = 0
    for term in terms:
        if term_mutate == term['t']:
            mutate_idx = term['idx'] - 1
            ch_rate[mutate_idx] *= 1.1 #即术语每次变异它的演化率会变为 1.1 倍
    for i2 in nodes:
        if i2['pos'] == (j+1): #判断变异术语在下一代哪个结点中
```

```
                    for k3 in i2['key']:
                        for k4 in i2['key']:
                            dist[find_idx(k3['term'], terms)][find_idx(k4['term'], terms)] *= 0.95    #即在下一代中合在一起的术语，距离进一步缩小为 0.95 倍
                    for k3 in i2['key']:
                        if term_mutate == k3['term']:  #如果术语在结点 i2 内
                            for k4 in i2['key']:
                                ch_dist[mutate_idx][find_idx(k4['term'],terms)] *= 0.9  #即术语每次变异它与变异后结点内其他术语的演化距离会变为 0.9 倍
print(dist)
print(ch_rate)
print(ch_dist)

#聚类部分,Kmean++
        # test
        # test_cluster = split_cluster(4, nodes[12]['key'], dist, ch_rate, ch_dist)
        # for i in range(4):
        #     print(test_cluster[i])

#第一年初始化结点
split_cluster_set = []    #一个三层嵌套数组，[ [[],[],[]], [[],[]] ], 最里层为一个裂子类，第二层为一个结点，最外层为一个年代
for i in nodes:
    if i['pos'] == 0:
        split_cluster_set.append( split_cluster(divide_cluster_num2(i['key'], nodes, link_sum, i['pos']), i['key'], dist, ch_rate, ch_dist) )

cur_year_nodes = []   #年代更迭时使用
total_year_nodes = []   #保存所有年代的结点，用于导出到 json 文件,这里给它初始化第一个年代的结点
for i in nodes:
```

```
    if i['pos'] == 0:
        total_year_nodes.append(i)
cur_nodes_series = [i for i in range( len(split_cluster_set) )]   #保存当前年代结点的序号，生成 link 时用到
total_links = []    #保存所有 link 信息

for year_pos in range(10):   #按年代进行整合
    if year_pos != 0:
        split_cluster_set = []   #一个三层嵌套数组，[ [[],[],[]], [[],[]] ], 最里层为一个裂子类，第二层为一个结点，最外层为一个年代
        for i in cur_year_nodes:
            split_cluster_set.append(split_cluster( divide_cluster_num2(i['key'], nodes, link_sum, i['pos']), i['key'], dist, ch_rate, ch_dist))
    print("split_cluster_set: ",split_cluster_set)
    change_cluster_set = change_term( split_cluster_set, ch_rate, ch_dist, terms)   #对数组变异操作

    cur_nodes_index = cur_nodes_series[-1]
    next_nodes_series = []   #保存新生成一代的结点序号

    #重新聚合，每次从每个结点内抽取 1 个或 0 个子类，直到所有结点都变为空，如果结合后类内距离减小，就并入？通过计算类间距离？
    final_set = []
    for i in change_cluster_set:   #对于每个结点
        ##########这个判断语句不一定正确
        while i != []:
            #取出新类时新建连接信息
            cur_nodes_index += 1
            next_nodes_series.append(cur_nodes_index)
            next_link = {}
            next_link['source'] = cur_nodes_series[ change_cluster_set.index(i) ]
```

```
next_link['target'] = cur_nodes_index
next_link['w1'] = random.random()
next_link['w2'] = random.random()
total_links.append(next_link)

# 取出当前结点第一个子类
temp_set = i[0]
i.remove(i[0])

for j in change_cluster_set:   #判断是否从其他结点取出元素
    if j != i and j != []:
        #计算当前类内距离
        temp_dist_set = []
        for k in temp_set:
            for k2 in temp_set:
                if k != k2:
                    temp_dist_set.append( dist[ find_idx(k,terms) ][ find_idx(k2,terms) ] )
        temp_aver_dist = sum(temp_dist_set) / len(temp_dist_set)
        #print("len temp_dist_set:",len(temp_dist_set))
        #对当前结点内每个类计算假设加入后的距离
        for test_set in j:
            test_dist_set = []
            for k in list(set(temp_set).union(set(test_set))):
                for k2 in list(set(temp_set).union(set(test_set))):
                    if k != k2:
                        test_dist_set.append( dist[ find_idx(k,terms) ][ find_idx(k2,terms) ] )
```

```
                    test_aver_dist = sum(test_dist_set) / len(test_dist_set)
                    #print("len test_dist_set:", len(test_dist_set))
                    #如果合并后类内距离更小,就将它加入
                    print(temp_aver_dist)
                    print(test_aver_dist)
                    if test_aver_dist < MERGE_TIMES*temp_aver_dist:
                        #将其他类并入新类时新建连接
                        next_link = {}
                        print("added j = ",j)
                        print("added j index in year = ",change_cluster_set.index(j))
                        print("added j index in total = ", cur_nodes_series[change_cluster_set.index(j)])
                        next_link['source'] = cur_nodes_series[change_cluster_set.index(j)]
                        next_link['target'] = cur_nodes_index
                        next_link['w1'] = random.random()
                        next_link['w2'] = random.random()
                        total_links.append(next_link)
                        #将子类并入新结点
                        temp_set = list(set(temp_set).union(set(test_set)))
                        j.remove(test_set)
                        break
            final_set.append(temp_set)
print("pos: ",year_pos + 1)

cur_nodes_series = next_nodes_series
cur_year_nodes = []
for every_node in final_set:
```

```
        print(every_node)
        generate_node = {}
        generate_node['key'] = []
        generate_node['pos'] = year_pos + 1   #生成的是下一代结点
        generate_node['n'] = random.random()
        generate_node['cluster'] = random.randint(0, 4)
        #根据 key 的所有 term 得到预测权值
        for every_term in every_node:
            term_tuple = {}
            term_tuple['term'] = every_term
            #根据拟合曲线预测 term 的值
            term_value = 0
            if 4* (year_pos + 1) - offset[find_idx(every_term,terms)] >= 0:
                term_value = fit_parameter[find_idx(every_term, terms)][0]  #先将最高次幂参数赋给它，用作计算的 x 值其实是考虑偏移后的：4*(year_pos+1) - offset[find_idx(every_term,terms)]
                for order in range( 1, len( fit_parameter[find_idx(every_term,terms)] ) ):
                    term_value = term_value * ( 4*(year_pos + 1) - offset[find_idx(every_term,terms)] ) + fit_parameter[find_idx(every_term,terms)][order]
            else:
                term_value = 0   #小于偏移量，即术语还未出现，值为 0

            if term_value < 0:  #防止曲线拟合后得到函数值小于 0
                term_value = 0
            term_tuple['w'] = term_value
            generate_node['key'].append(term_tuple)
        #根据结点所有 term 的权值给结点命名
        term_for_name = []
```

```
        value_for_name = 0
        for every_key in generate_node['key']:
            if every_key['w'] > value_for_name:
                value_for_name = every_key['w']
                term_for_name = every_key['term']
        generate_node['name'] = term_for_name
        # 结点大小根据结点术语集内的权值，NODE_SHRINK_TIMES 为缩小倍数
        generate_node['w'] = 0
        for i in generate_node['key']:
            generate_node['w'] += i['w']/NODE_SHRINK_TIMES
        cur_year_nodes.append(generate_node)
        total_year_nodes.append(generate_node)

#将对象写入 json 文件

#将 node 写入 json 文件
print( "the number of nodes: ", len(total_year_nodes) -1 )   #打印结点序号最大的下标
fileObject = open('NodesFile.json', 'w')
jsonStr = json.dumps(total_year_nodes, ensure_ascii=False, indent=2)
fileObject.write(jsonStr)
fileObject.close()

#将 link 写入 json 文件

#找 link 的 target 结点中的最大下标
total_links_target_print = []
```

```
for i in total_links:
    total_links_target_print.append(i['target'])
print( "find the biggest target: ", max(total_links_target_print) )

fileObject = open('LinksFile.json', 'w')
jsonStr = json.dumps(total_links, ensure_ascii=False, indent=2)
fileObject.write(jsonStr)
fileObject.close()
```