教学上机实验报告

课程名称： 自然语言处理

任课教师姓名： 林忠华

学生学号： 311809000608

学生姓名： 王荣胜

学生专业班级： 计实验 1801

2020 ～ 2021 学年 第 二 学期

河南理工大学

教学上机实验报告评价分值标准

序号	评价指标	分值	评价等级及参考分值					评价分
			优	良	中	合格	差	
1	实验报告内容完整充实	10	10	8	7	6	3	
2	实验内容书写规范、字迹工整认真	10	10	8	7	6	3	
3	实验过程叙述详细、概念正确，语言表达准确，结构严谨，调理清楚，逻辑性强，自己努力完成，没有抄袭。	30	30	26	23	20	10	
4	对实验过程中存在的问题分析详细透彻、深刻、全面、规范、，结合实验内容，有自己的个人见解和想法，并能结合该实验提出相关问题，给出解决方法。	30	30	26	23	20	10	
5	实验结果、分析和结论正确无误	20	20	17	15	13	6	
总得分								

签名（签章）：

日期：　　年　月　日

河南理工大学教学上机实验报告

上机时间　　2021 年 5 月 27 日

实验题目：

实验一　基于规则的分词算法

实验目的和要求：

目的：分别使用完全切分算法、正向最长匹配、逆向最长匹配、双向最长匹配算法对句子进行切分

要求：比较四种算法的优缺点，用书上 41 页的表格进行分析。

实验过程：

算法一：完全切分

算法二：正向最长匹配算法

1. 实验原理

切分出单字串，然后和词库进行比对，如果是一个词就记录下来， 否则通过增加或者减少一个单字，继续比较，一直还剩下一个单字则终止。

2.算法描述：

设 MaxLen 表示最大词长，D 为分词词典。

①待切分语料中按正向取长度为 MaxLen 的字串 str，令 Len=MaxLen;

②把 str 与 D 中的词相匹配；

③若匹配成功，则认为该字串为词，指向待切分语料的指针向前移 Len 个汉字（字节），返回到①；

④若不成功：如果 Len>1，则将 Len 减 2 个字节，从待切分语料中取长度为 Len 的字串 str，返回到②。否则，得到长度为 2 的单字词，指向待切分语料的指针向前移 1 个汉字，返回①。

3.算法实现

```
class MaximumMatching(object):

    def __init__(self):
        self.window_size = 6     # 定义词典中的最长单词的长度


    def tokenize(self, text):
        tokens = []                 # 定义一个空列表来保存切分的结果
        index = 0                   # 切分
        text_length = len(text)

        # 定义需要被维护的词典，其中词典最长词的长度为 6
        maintained_dic = ['研究', '研究生', '自然', '自然语言', '语言','自然语言处理',
'处理', '是', '一个', '不错', '的', '科研', '方向']
```

```
                while text_length > index:      # 循环结束判定条件
        #                 print(text_length)
                    print('index 4: ', index)
                    for size in range(self.window_size + index, index, -1):     # 根据窗口大小循环，直到找到符合的进行下一次循环
                        print('index 1: ', index)
                        print('window size: ', size)
                        piece = text[index: size]          # 被匹配字段
                        if piece in maintained_dic:       # 如果需要被匹配的字段在词典中的话匹配成功，新的 index 为新的匹配字段的起始位置
                            index = size - 1
                            print('index 2: ', index)
                            break
                    index += 1
                    print('index 3', index, '\n')
                    tokens.append(piece)                # 将匹配到的字段保存起来
                return tokens

        if __name__ == '__main__':
            text = '研究生研究自然语言处理是一个不错的研究方向'
            tokenizer = MaximumMatching()
        print(tokenizer.tokenize(text))
```

算法三：逆向最长匹配算法

1.实验原理

逆向最大匹配算法的实现过程基本跟正向最大匹配算法无差，唯一不同的点是分词的切分是从后往前，跟正向最大匹配方法刚好相反。也就是说逆向是从字符串的最后面开始扫描，每次选取最末端的 i 个汉字字符作为匹配词段，若匹配成功则进行下一字符串的匹配，否则则移除该匹配词段的最前面一个汉字，继续匹配。需要注意的是，分词词典为逆向词典，即每个词条都以逆序的方式存放。

2.算法描述：

设 MaxLen 表示最大词长，D 为分词词典。

(1)将文章分成句子(通过标点符号来实现)；

(2)循环的读入每一个句子 S，设句子中的字数为 n；

(3)设置一个最大词长度，就是我们要截取的词的最大长度 max；

(4)从句子中取 n-max 到 n 的字符串 subword，去字典中查找是否有这个词。如果有就走(5),没有就走(6)；

(5)记住 subword，从 n-max 付值给 n，继续执行(4)，直到 n=0。

(6)将 max-1，再执行(4)。

3.算法实现

算法四：双向最长匹配算法

1.实验原理

切分出单字串，然后和词库进行比对，如果是一个词就记录下来， 否则通过增加或者减少一个单字，继续比较，一直还剩下一个单字则终止。

2.算法描述

双向最大匹配算法其实是在正向最大匹配和逆向最大匹配两个算法的基础上延伸出来的，基本的思想很简单，主要可以分为如下两大步骤：

①当正向和反向的分词结果的词语数目是不一样的，那么这个时候就选取分词数量较少的那组分词结果；

②倘若分词结果词语数量是一样的，又可以分为两种子情况去考虑：

分词结果完全一样，那么就不具备任何歧义，即表示正向反向的分词结果皆可得到满足；

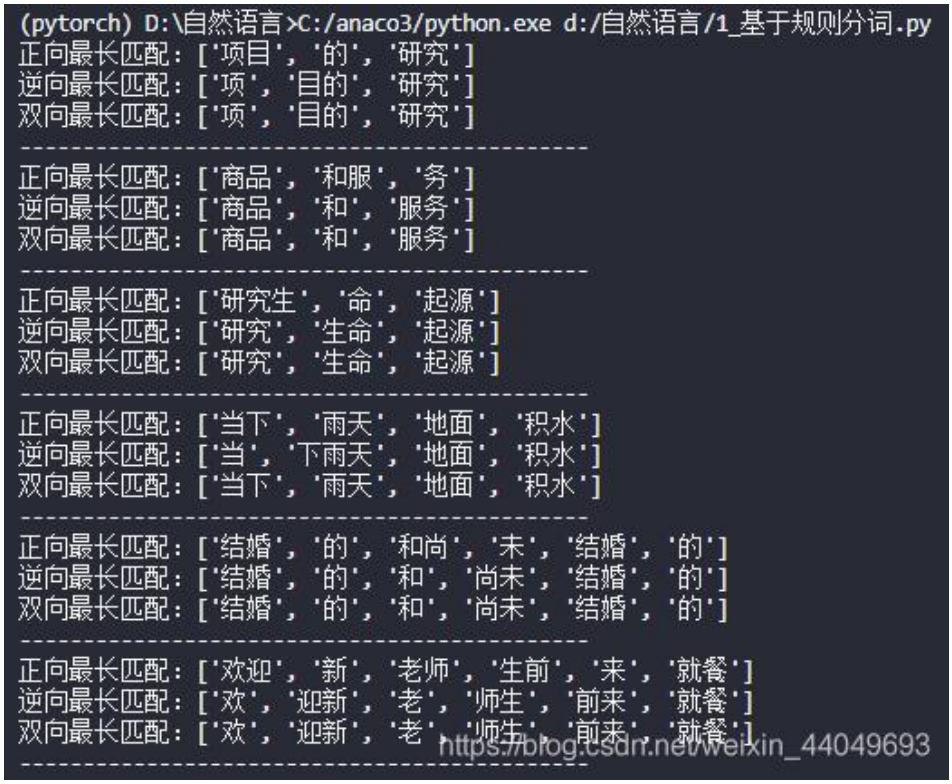
如果不一样，那么就选取分词结果中单个汉字数目较少的那一组。

3.算法实现

```
import operator


class BiDirectionMatching(object):

    def __init__(self):
        self.window_size = 6
        self.dic = ['研究', '研究生', '生命', '命', '的', '起源']


    def mm_tokenize(self, text):
        tokens = []                  # 定义一个空列表来保存切分的结果
        index = 0                    # 切分
        text_length = len(text)

        while text_length > index:      # 循环结束判定条件
            for size in range(self.window_size + index, index, -1):    # 根据窗口大小循环，直到找到符合的进行下一次循环
                piece = text[index: size]          # 被匹配字段
                if piece in self.dic:       # 如果需要被匹配的字段在词典中的话匹配成功，新的 index 为新的匹配字段的起始位置
                    index = size - 1
                    break
            index += 1
            tokens.append(piece)               # 将匹配到的字段保存起来

        return tokens

    def rmm_tokenize(self, text):
        tokens = []
        index = len(text)

        while index > 0:
```

```
                for size in range(index - self.window_size, index):
                    w_piece = text[size: index]
                    if w_piece in self.dic:
                        index = size + 1
                        break
                index -= 1
                tokens.append(w_piece)
        tokens.reverse()

        return tokens

    def bmm_tokenize(self, text):
        mm_tokens = self.mm_tokenize(text)
        print('正向最大匹配分词结果:', mm_tokens)
        rmm_tokens = self.rmm_tokenize(text)
        print('逆向最大匹配分词结果:', rmm_tokens)

        if len(mm_tokens) != len(rmm_tokens):
            if len(mm_tokens) > len(rmm_tokens):
                return rmm_tokens
            else:
                return mm_tokens
        elif len(mm_tokens) == len(rmm_tokens):
            if operator.eq(mm_tokens, rmm_tokens):
                return mm_tokens
            else:
                mm_count, rmm_count = 0, 0
                for mm_tk in mm_tokens:
                    if len(mm_tk) == 1:
                        mm_count += 1
                for rmm_tk in rmm_tokens:
                    if len(rmm_tk) == 1:
                        rmm_count += 1
                if mm_count > rmm_count:
                    return rmm_tokens
                else:
                    return mm_tokens


if __name__ == '__main__':
    text = '研究生命的起源'
    tokenizer = BiDirectionMatching()
    print('双向最大匹配得到的结果：', tokenizer.bmm_tokenize(text))
```

实验结果：



实验分析：

1. 完全切分输出的不是有意义的词语序列，而是出现在所有词典中的链表，分析结果不是我们所要的，而且效率低。

2. 正向最长匹配：

优点：原理简单，程序容易实现，复杂度低。

缺点：（1）忽视“词中有词”的现象，导致切分错误；

（2）最大词长难以确定：

①太长：匹配所花时间多，算法时间复杂度高；

②太短：不能切分长度超过它的词，导致切分正确率降低。

3. 逆向最长匹配：与正相匹配相同，程序的执行效率并不高，因为算法需要不断的去检测匹配的字段。

4. 双相匹配：

优点：克服了 MM 方法里忽视“词中有词”的现象。

缺点：

① 算法复杂度提高。

② 为了支持正反向匹配算法，词典设置要更复杂。

③ 对某些句子仍然无法发现歧义。

实验成绩：

日期：_____年____月___日

河南理工大学教学上机实验报告

上机时间　　2021 年 5 月 27 日

实验题目：

实验二 隐马尔可夫模型在分词中的应用（1）

实验目的和要求：

目的：通过实现隐马尔科夫模型，了解隐马尔科夫模型的实际应用，对中文分词有进一步的学习和了解。

要求：1.如何中文分词序列标注的 BMES

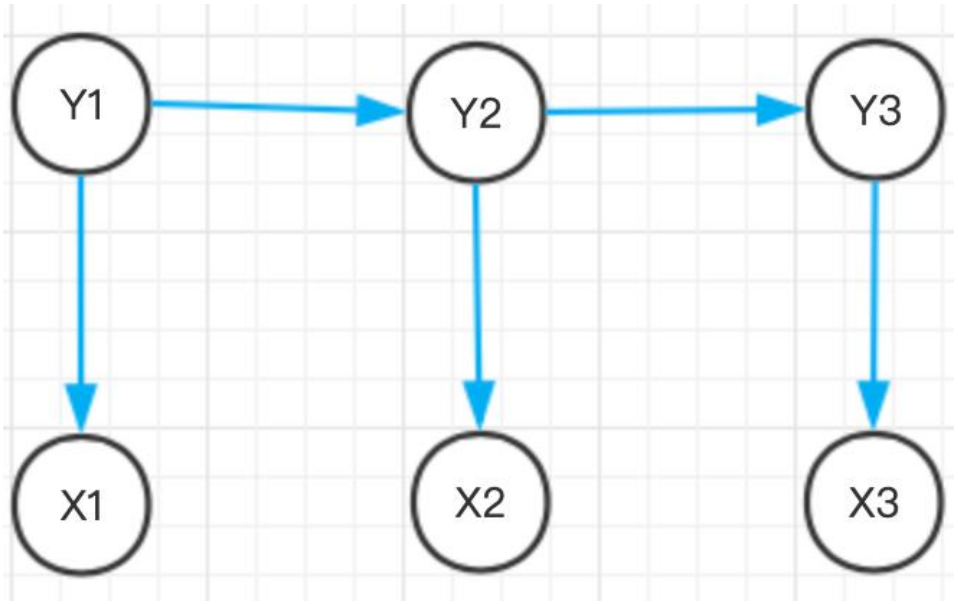
2.序列标准初始状态概率向量，状态转移概率矩阵，发射概率矩阵如何求出。

实验过程：

1.对隐马尔科夫模型的了解

隐马尔可夫模型（Hidden Markov Model，HMM）是统计模型，它用来描述一个含有隐含未知参数的马尔可夫过程。它的状态不能直接观察到，但能通过观测向量序列观察到，每个观测向量都是通过某些概率密度分布表现为各种状态，每一个观测向量是由一个具有相应概率密度分布的状态序列产生。所以，隐马尔可夫模型是一个双重随机过程----具有一定状态数的隐马尔可夫链和显示随机函数集。

两个基本假设：齐次马尔可夫性假设（当前隐状态只依赖前一状态）、观测独立性假设（观测只依赖当前状态）。



2. .如何中文分词序列标注的 BMES

思路：本质上是一个序列标注问题，将语句中的字，按照他们在词中的位置进行标注。

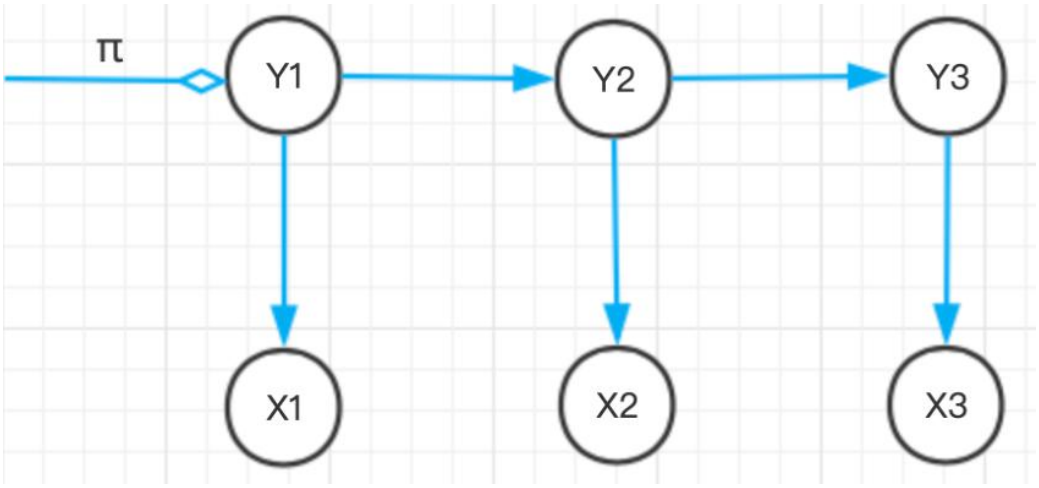
标注主要有：B（词开始的一个字），E（词最后一个字），M（词中间的字，可能多个），S

（由一个字表示的词）

语句是观测序列，而序列标注结果（B、E、M、S）是隐藏序列。任何一个 HMM 都可以由一个五元组来描述：观测序列，隐藏序列，隐藏态起始概率，隐藏态之间转换概率（转移概率），隐藏态表现为观测值的概率（发射概率）。从隐藏态初始状态出发，计算下一个隐藏态的概率，并依次计算后面所有的隐藏态转移概率。序列标注问题就转化为了求解概率最大的隐藏状态序列问题。

2. 初始状态概率矩阵 π

表示隐含状态在初始时刻 t=1 的概率矩阵，(例如 t=1 时，P(S1)=p1、P(S2)=P2、P(S3)=p3，则初始状态概率矩阵 π=[p1 p2 p3].



3.

隐含状态转移概率矩阵 A。

描述了 HMM 模型中各个状态之间的转移概率。

其中 aij = P(Sj | Si),1≤i,j≤N. 表示在 t 时刻、状态为 Si 的条件下，在 t+1 时刻状态是 Sj 的概率。

A 和 π 是一阶马尔可夫链的两个参数。

4. 观测状态转移概率矩阵(发射概率矩阵) B

令 N 代表隐含状态数目，M 代表可观测状态数目，则：

Bqj(oi) = P(Ot=Oi | qt=Sj), 1≤i≤M,1≤j≤N

表示在 t 时刻、隐含状态是 Sj 条件下，观察状态为 Oi 的概率。

总结：一般的，可以用 λ=(A,B,Π)三元组来简洁的表示一个隐马尔可夫模型。隐马尔可夫模型实际上是标准马尔可夫模型的扩展，添加了可观测状态集合和这些状态与隐含状态之间的概率关系。

3.核心代码

```
# 定义工具函数，取出一个词语中每个字的标记
def get_tag(word):
    tag = []
    if len(word) == 1:
```

```
            tag = ['S']
        elif len(word) == 2:
            tag = ['B', 'E']
        else:
            num = len(word) - 2
            tag.append('B')
            tag.extend(['M'] * num)
            tag.append('E')
        return tag

    #定义训练程序，得到初始概率向量，状态转移矩阵，和发射矩阵
    def get_Mats_fenci(filename):
        word_file = open(filename,'r', encoding='utf-8').readlines()
        seg_stop_words = {" ","，","。","“","”",'‘',"？","！","：","《","》","、","；","•",
" ‘ ","’ ",
                          "——",",",".","?","!","`","~","@","#","$","%","^","&","*","(",")","-",
                          "_","+","=","[","]","{","}","'",'"',"<",">","\\","|" "\r","\n","\t"}
        trans_mat, emit_mat, init_vec, state_count = mats_setup(['B', 'M', 'E', 'S'])
        for line in word_file:
            line = line.strip()
            if not line:
                continue
            # 获取观测序列
            observes = []
            for i in range(len(line)):
                if line[i] not in seg_stop_words:
                    observes.append(line[i])
            words = line.split(" ")
            # 获取实际状态序列
            states = []
            for word in words:
                if word not in seg_stop_words:
                    states.extend(get_tag(word))
            # 计数，记频率
            if(len(observes) >= len(states)):
                for i in range(len(states)):
                    if i == 0:
                        init_vec[states[0]] += 1
                        state_count[states[0]] += 1
                    else:
                        trans_mat[states[i - 1]][states[i]] += 1
                        state_count[states[i]] += 1
                    if observes[i] not in emit_mat[states[i]]:
                        emit_mat[states[i]][observes[i]] = 1
```

```
                    else:
                        emit_mat[states[i]][observes[i]] += 1
            else:
                pass
        return init_vec,trans_mat,emit_mat,state_count
```

实验结果：

实验分析：

先定义状态映射字典，方便我们定位状态在列表中对应位置。再定义状态转移矩阵 A。总共 4 个状态，所以 4x4。在 ord 中，中文编码大小为 65536，总共 4 个状态，所以 B 矩阵 4x65536。初始状态，每一个句子的开头只有 4 中状态（词性）。如果句子较长，许多个较小的数值连乘，容易造成下溢。对于这种情况，我们常常使用 log 函数解决。但是，如果有一些没有出现的词语，导致矩阵对应位置 0，那么测试的时候遇到了，连乘中有一个为 0，整体就为 0。但是 log0 是不存在的，所以我们需要给每一个 0 的位置加上一个极小值（-3.14e+100），使得其有定义。计算 A 矩阵时要注意每一行的和为 1，即从某个状态向另外 4 个状态转移概率只和为 1

实验成绩：

日期：_____年____月___日

河南理工大学教学上机实验报告

上机时间　2021 年 6 月 3 日

实验题目：

隐马尔可夫模型在分词中的应用（2）

实验目的和要求：

已知语料库中有且仅有如下 3 个句子：

商品 和 服务

商品 和服 物美价廉

服务 和 货币

应用隐马尔可夫模型和维特比算法给出‘商品和货币’这句话的分词结果。

实验过程：

1.隐马尔科夫

隐马尔可夫模型(Hidden Markov Model, HMM)是描述两个时序序列联合分布 p(x,y) 的概率模型: x 序列外界可见(外界指的是观测者)，称为观测序列(obsevation sequence); y 序列外界不可见，称为状态序列(state sequence)。比如观测 x 为单词，状态 y 为词性，我们需要根据单词序列去猜测它们的词性。隐马尔可夫模型之所以称为“隐”，是因为从外界来看，状态序列(例如词性)隐藏不可见，是待求的因变量。从这个角度来讲，人们也称状态为隐状态(hidden state),而称观测为显状态(visible state)。隐马尔可夫模型之所以称为“马尔可夫模型”,”是因为它满足马尔可夫假设。

2.维比特算法

维特比算法是一种动态规划方法。动态规划的基础是贝尔曼最优性原理（Bellman’s Principle of Optimality）：多级决策过程的最优策略（最优序列）有这样的性质：不论初始状态和初始决策如何，其余的决策对于初始决策所形成的状态来说，必定也是一个最优策略。

（在解码问题里就是说：如果最优状态序列 q1* ,q2* ,…,qT* 在时刻 t 的状态 qt* 已知为 Si，那么从 qt* 到 qT* 的局部状态序列一定是所有可能的状态序列里最优的。通俗一点说，已知最优路径 P，那么我们在路径上选择一个节点，从起点到该节点的这段局部路径 P1 一定是所有可能的局部路径里最优的；同样地，从这个节点到终点的局部路径 P2 一定是所有可能的局部路径里最优的。）

维特比算法利用以下的方法求解最优序列：从起始时刻 t=1 开始，递推地找出在时刻 t 的状态为 Si 的各个可能的状态序列中的最大概率，一直求解到时刻 t=T 的状态为 Si 的最大概率，并得到时刻 T 的状态 Sj；然后向前回溯求得其他时刻的状态。

3.代码实现

```
from pyhanlp import *
import os
```

```
import zipfile
from pyhanlp.static import download, remove_file, HANLP_DATA_PATH
CWSEvaluator = SafeJClass('com.hankcs.hanlp.seg.common.CWSEvaluator')

def test_data_path():
    """
    获取测试数据路径，位于$root/data/test，根目录由配置文件指定。
    :return:
    """
    data_path = os.path.join(HANLP_DATA_PATH, 'test')
    if not os.path.isdir(data_path):
        os.mkdir(data_path)
    return data_path

def ensure_data(data_name, data_url):
    root_path = test_data_path()
    dest_path = os.path.join(root_path, data_name)
    if os.path.exists(dest_path):
        return dest_path
    if data_url.endswith('.zip'):
        dest_path += '.zip'
    download(data_url, dest_path)
    if data_url.endswith('.zip'):
        with zipfile.ZipFile(dest_path, "r") as archive:
            archive.extractall(root_path)
        remove_file(dest_path)
        dest_path = dest_path[:-len('.zip')]
    return dest_path

sighan05 = ensure_data('icwb2-data',
'http://sighan.cs.uchicago.edu/bakeoff2005/data/icwb2-data.zip')
msr_dict = os.path.join(sighan05, 'gold', 'msr_training_words.utf8')
msr_train = os.path.join(sighan05, 'training', 'msr_training.utf8')
msr_model = os.path.join(test_data_path(), 'msr_cws')
msr_test = os.path.join(sighan05, 'testing', 'msr_test.utf8')
msr_output = os.path.join(sighan05, 'testing', 'msr_bigram_output.txt')
msr_gold = os.path.join(sighan05, 'gold', 'msr_test_gold.utf8')
FirstOrderHiddenMarkovModel =
JClass('com.hankcs.hanlp.model.hmm.FirstOrderHiddenMarkovModel')
HMMSegmenter = JClass('com.hankcs.hanlp.model.hmm.HMMSegmenter')


def train(corpus, model):
    segmenter = HMMSegmenter(model)
```

```
    segmenter.train(corpus)
    print(segmenter.segment('商品和货币'))
    return segmenter.toSegment()


def evaluate(segment):
    result = CWSEvaluator.evaluate(segment, msr_test, msr_output, msr_gold, msr_dict)
    print(result)


if __name__ == '__main__':
    segment = train(msr_train, FirstOrderHiddenMarkovModel())
    evaluate(segment)
```

实验结果：

```
(pytorch) D:\自然语言>C:/anaco3/python.exe d:/自然语言/2_隐马尔可夫分词.py
语料：86k...[商品，和，货币]
```

实验分析：

维特比算法的基础可以概括为下面三点：

1、如果概率最大的路径经过篱笆网络的某点，则从开始点到该点的子路径也一定是从开始到该点路径中概率最大的。

2、假定第 i 时刻有 k 个状态，从开始到 i 时刻的 k 个状态有 k 条最短路径，而最终的最短路径必然经过其中的一条。

3、根据上述性质，我们在计算第 i+1 状态的最短路径时，只需要考虑从开始到当前的 k 个状态值的最短路径和当前状态值到第 i+1 状态值的最短路径即可，如求 t=3 时的最短路径，等于求 t=2 时的所有状态结点 y2i 的最短路径加上 t=2 到 t=3 的各节点的最短路径。

实验成绩：

日期：_____年____月___日

河南理工大学教学上机实验报告

上机时间　　2021 年 6 月 10 日

实验题目：

基于 K-means 的文本聚类方法

实验目的和要求：

记录有某音乐网站 6 位用户点播的歌曲流派，给定用户名称和九种音乐类型的播放数量，通过 K-means 将 6 位用户分成 3 簇（把播放数量当成数组或者向量进行簇质心的迭代运算）。

实验过程：

1. K-means 算法简介

K-means 算法是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。

k 个初始类聚类中心点的选取对聚类结果具有较大的影响，因为在该算法第一步中是随机的选取任意 k 个对象作为初始聚类的中心，初始地代表一个簇。该算法在每次迭代中对数据集中剩余的每个对象，根据其与各个簇中心的距离将每个对象重新赋给最近的簇。当考察完所有数据对象后，一次迭代运算完成，新的聚类中心被计算出来。如果在一次迭代前后，J 的值没有发生变化，说明算法已经收敛。

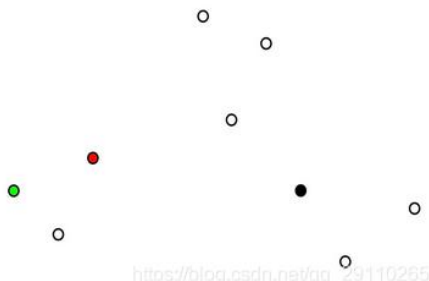
2.算法实现

1）从 N 个文档随机选取 K 个文档作为质心

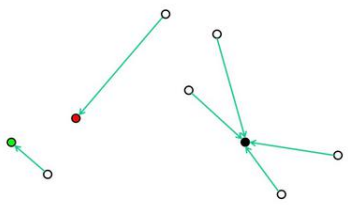
2）对剩余的每个文档测量其到每个质心的距离，并把它归到最近的质心的类

3）重新计算已经得到的各个类的质心

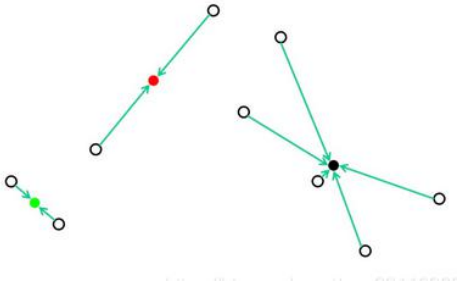
4）迭代 2～3 步直至新的质心与原质心相等或小于指定阀值，算法结束



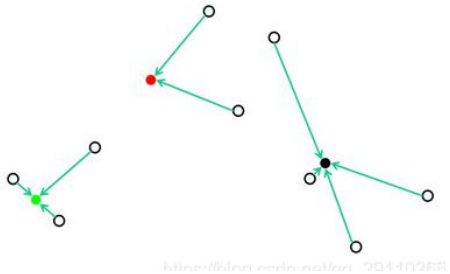
1、首先在图中随机选取 3 个点



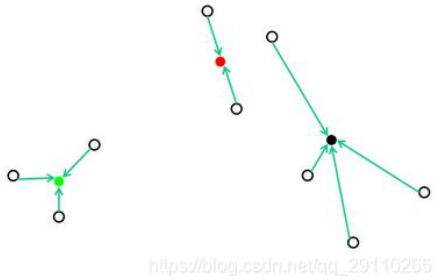
2、然后把距离这三个点最近的其他点归为一类



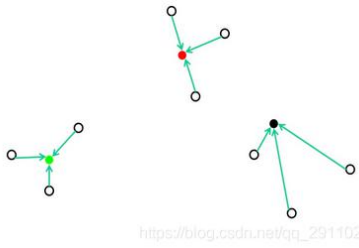
3、取当前类的所有点的均值，作为中心点



4、更新距离中心点最近的点



5、再次计算被分类点的均值作为新的中心点



6、再次更新距离中心点最近的点

通过不断重复上述步骤直至无法再进行更新为止时聚类完成。

首先从 n 个数据对象任意选择 k 个对象作为初始聚类中心；而对于所剩下其它对象，则根据它们与这些聚类中心的相似度（距离），分别将它们分配给与其最相似的（聚类中心所代表的）聚类；然 后再计算每个所获新聚类的聚类中心（该聚类中所有对象的均值）；不断重复这一过程直到标准测度函数开始收敛为止。一般都采用均方差作为标准测度函数. k 个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。

代码实现：

```
from pyhanlp import *


ClusterAnalyzer = JClass('com.hankcs.hanlp.mining.cluster.ClusterAnalyzer')
```

```
analyzer = ClusterAnalyzer()
analyzer.addDocument("赵一", "流行，流行，流行，流行，流行，流行，流行，流行，流行，流行，蓝调，蓝调，蓝调，蓝调，蓝调，蓝调，摇滚，摇滚，摇滚，摇滚")
analyzer.addDocument("钱二", "爵士，爵士，爵士，爵士，爵士，爵士，爵士，爵士，舞曲，舞曲，舞曲，舞曲，舞曲，舞曲，舞曲，舞曲")
analyzer.addDocument("张三", "古典，古典，古典，古典，民谣，民谣，民谣，民谣")
analyzer.addDocument("李四", "爵士，爵士，爵士，爵士，爵士，爵士，爵士，爵士，爵士，金属，金属，舞曲，舞曲，舞曲，舞曲，舞曲，舞曲")
analyzer.addDocument("王五", "流行，流行，流行，流行，摇滚，摇滚，摇滚，嘻哈，嘻哈，嘻哈")
analyzer.addDocument("马六", "古典，古典，古典，古典，古典，古典，古典，古典，摇滚")
print(analyzer.kmeans(3))
```

实验结果：

```
(pytorch) D:\自然语言>C:/anaco3/python.exe d:/自然语言/4_K-means文本聚类.py
[[张三，马六]，[王五，赵一]，[李四，钱二]]
```

实验分析：

k-means 的误差函数有一个很大缺陷，就是随着簇的个数增加，误差函数趋近于 0，最极端的情况是每个记录各为一个单独的簇，此时数据记录的误差为 0，但是这样聚类结果并不是我们想要的，可以引入结构风险对模型的复杂度进行惩罚：

是平衡训练误差与簇的个数的参数，但是现在的问题又变成了如何选取了，有研究指出，在数据集满足高斯分布时，其中 m 是向量的维度。

另一种方法是按递增的顺序尝试不同的 k 值，同时画出其对应的误差值，通过寻求拐点来找到一个较好的 k 值。

实验成绩：

日期：_____年____月___日