大数据大作业之分词器

电气钱82李儒欢

2184214378

目录

一、问题导入

2021年的情人节前夕，正愁不知道送出一份足够有新意有诚意的礼物，在知乎上看到了一个足够好用的API——“jieba”分词器。于是获取（pojie）自己的微信聊天记录之后，将聊天记录分词并统计词频，再使用“wordcloud”就能生成和对象的聊天记录的生动形象的词云。本次大作业基于这个引子，对中文分词这个题材进行简单讨论。

在上述任务完成的过程中，最关键的步骤在于分词，如何将聊天记录这一大列字符串组中的词语提取出来，并输入到后续进行处理，是本文主要讨论的内容。

二、算法介绍

2.1 使用DAG进行中文分词

中文分词是中文信息处理的基本技术，指将一个汉字序列切分成一个个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。

> 词是最小的能够独立活动的有意义的语言成分，英文单词之间是以空格作为自然分界符的，而汉语是以字为基本的书写单位，词语之间没有明显的区分标记。因此中文分词的难度要远大于英文分词。

中文分词并不是那么容易，它涉及许多方面的问题，主要包括：

(1) 核心词表问题

(2) 词的变形问题

(3) 词缀的问题

(4) 汉语自动分词规范须支持各种不同目标的应用，但不同目标的应用对词的要求是不同甚至是矛盾的。

最常见的分词方法是基于词典的方法：**字符串匹配，机械分词方法。**原理：按照一定策略将待分析的汉字串与一个“大机器词典”中的词条进行匹配，若在词典中找到某个字符串，则匹配成功。分词就是识别句中的词组，然后把句子拆分成尽量大的块。但由于上下文语境不同，拆分时也常常出现规则冲突，比如“研究生命的起源”，既可拆成“研究生 命 的 起源”，也可拆成“研究 生命 的 起源”。因此，需要制定一些规则处理这些冲突。

- 当我们使用DAG方法构造一个有向无环图，具体方法是以句子的每一个字为开头能组成什么词。
- 我们以：“研究生命的起源”这句话为例：

  函数返回的结果是字典DAG，其中每个元素都是位置的索引号：

```
DAG {0: [0, 1, 2], 1: [1], 2: [2, 3], 3: [3], 4: [4], 5: [5, 6], 6: [6]}
```

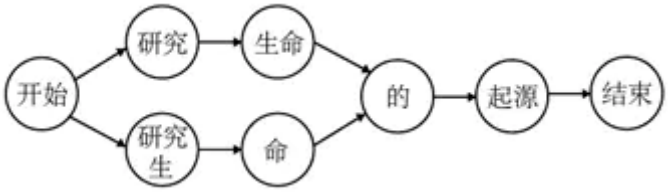
“研”字可以单独成词，也可以和“研究”，“研究生”组合，这是一个开头三种结尾的情况0:[0,1,2]；

“究”不能与后面的“生”组合成词，因此第1个开始位置只对应一个结束位置1:[1]；

“生”字可以单独成词，也可以和“生命”组合，一个开头两种结尾2:[2,3]

后面的其它字以此类推。

我们将其中每个可能出现的词（含一字词和多字词）作为单个元素，组合成图，方向按文字从左到右，既有向且无环（如果同一个词出现在句子的不同位置也认为是不同元素）。可以得到以下图。



目标是找到一条从“开始”到“结束”的路径，且整体路径的权重之和最大，每一个点的权重是该词汇出现的频率。

这是一个非常简单的有向无环图DAG应用场景。定义了入口，出口，可用节点，节点权重和节点间可达的方向和关系。通过计算权重选择最佳路径（最佳子图）。

DAG有向无环图指的是一个无回路的有向图（如果有一个非有向无环图，且A点出发向B经C可回到A，形成一个环）。DAG可看作是树结构的扩展，所有的有向树都是有向无环图。

但是这种方法对词表的依赖很大，一旦出现词表中不存在的新词，算法是无法做到正确的切分的。

因此我们需要采取一些增加对于“陌生词”的辨识能力的分词方法，引入自然语言处理的方法：隐马尔可夫模型HMM（在西瓜书的第十四章有相关内容介绍）。

2.2 隐马尔可夫模型（HMM）

隐马尔可夫模型（Hidden Markov Model；缩写：**HMM**）或称作**隐性马尔可夫模型**，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。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析，例如模式识别。（来自wiki百科）

在**正常的**马尔可夫模型中，状态对于观察者来说是直接可见的。这样状态的转换概率便是全部的参数。而在**隐**马尔可夫模型中，状态并不是直接可见的，但受状态影响的某些变量则是可见的。每一个状态在可能输出的符号上都有一概率分布。因此输出符号的序列能够透露出状态序列的一些信息。

现实生活中有这样一类随机现象，在已知现在情况的条件下，未来时刻的情况只与现在有关，而与遥远的过去并无直接关系。比如天气预测，如果我们知道“晴天，多云，雨天”之间的转换概率，那么如果今天是晴天，我们就可以推断出明天是各种天气的概率，接着后天的天气可以由明天的进行计算。这类问题可以用 Markov 模型来描述。

HMM 模型的本质是从观察的参数中获取隐含的参数信息，并且前后之间的特征会存在部分的依赖影响。在我们的中文分词中，主要负责**未收录词的划分**。

> 根据可观察状态的序列找到一个最可能的隐藏状态序列

中文分词，就是给一个汉语句子作为输入，以“BEMS”组成的序列串作为输出，然后再进行切词，进而得到输入句子的划分。其中，B代表该字是词语中的起始字，M代表是词语中的中间字，E代表是词语中的结束字，S则代表是单字成词。

例如：给个句子

> 小明硕士毕业于中国科学院计算所

得到BEMS组成的序列为

> BEBEBMEBEBMEBES

因为句尾只可能是E或者S，所以得到切词方式为

> BE/BE/BME/BE/BME/BE/S

进而得到中文句子的切词方式为

> 小明/硕士/毕业于/中国/科学院/计算/所

这是个HMM问题，因为你想要得到的是每个字的位置，但是看到的只是这些汉字，需要通过汉字来推出每个字在词语中的位置，并且每个字属于什么状态还和它之前的字有关。
此时，我们需要根据可观察状态的序列找到一个最可能的隐藏状态序列。

思维的转换很重要，下面对中文分词进行形式化描述：

> 设观察集合为$\mathbf{O}=\{o_1,o_2,\ldots,o_l\}$;状态集合为$\mathbf{S}=\{s_1,s_2,\ldots,s_k\}$
>
> 问题：已知输入的观察序列为:$\mathbf{X}=x_1x_2\ldots x_n;x_i\in\mathbf{O}$.
>
> 求对应的状态序列：$\mathbf{Y}=y_1y_2\ldots y_n;y_i\in\mathbf{S}$

可以直接进行求解

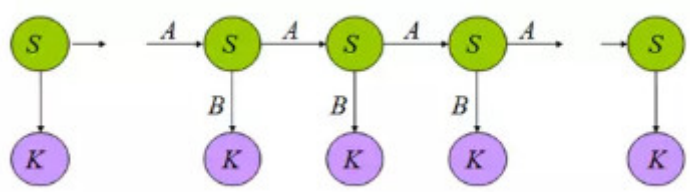
$$P(Y|X)=\frac{P(X|Y)P(Y)}{P(X)}$$

> 两个假设

利用贝叶斯公式得到：

$$P(s_1,s_2,s_3,\ldots|o_1,o_2,o_3\ldots.)=P(o_1,o_2,o_3,\ldots|s_1,s_2,s_3\ldots.)*P(s_1,s_2,s_3,\ldots)$$

这里需要用到两个假设来进一步简化上述公式



遍历状态Y的所有可能性，条件概率最大的则为对应的状态序列。

由于P(X)都一样，而我们只关心哪个最大，所以只需求$P(X|Y)P(Y)$即可。

下面需要引入马尔科夫假设了：

假设1： 齐次马尔科夫假设：状态序列只和前一个状态有关系，与其他状态及观察值无关，即：

$$P(y_i|x_1,y_1,x_2,y_2,\ldots,x_{i-1},y_{i-1})=P(y_i|y_{i-1})$$

假设2： 观察独立性假设。即假设任意时刻的观察值仅与当前的状态值有关，即：

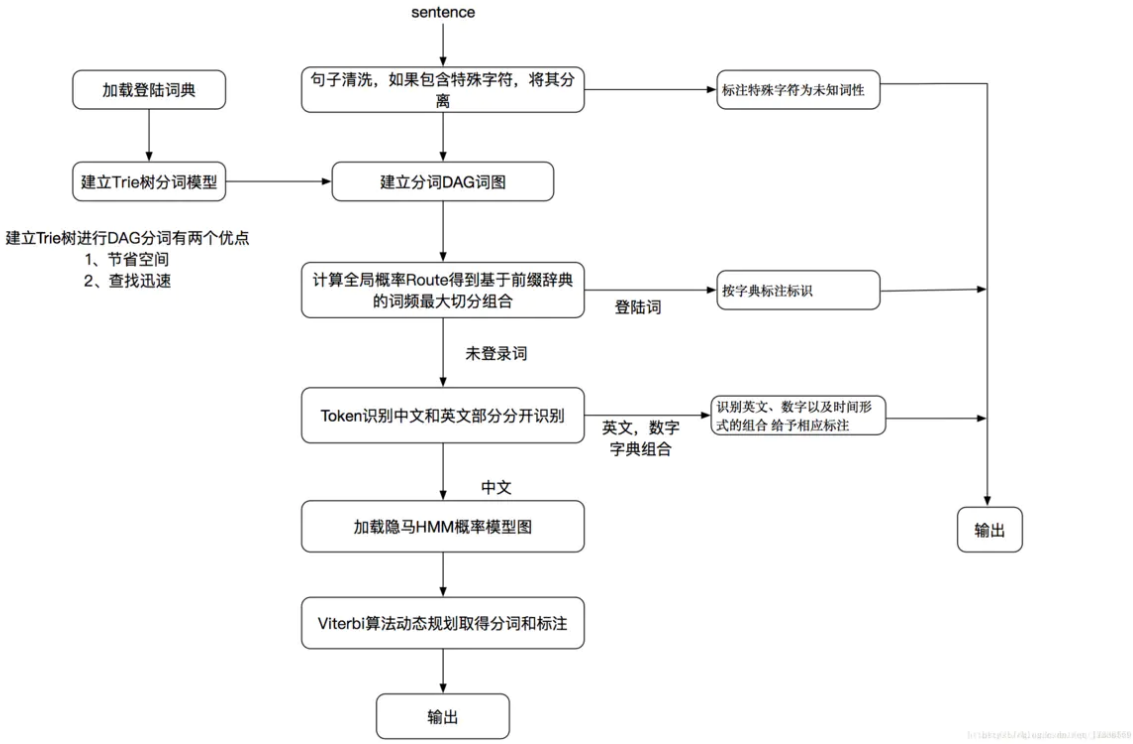
$$P(x_i|x_1,y_1,x_2,y_2,\ldots,x_{i-1},y_{i_1},y_i)=P(x_i|y_i)$$

有了以上假设就容易计算$P(X|Y)P(Y)$了

$$P(X|Y)P(Y)=\pi(y_1)\prod_{2}^{n}P(y_i|y_{i-1})P(x_i|y_i)$$

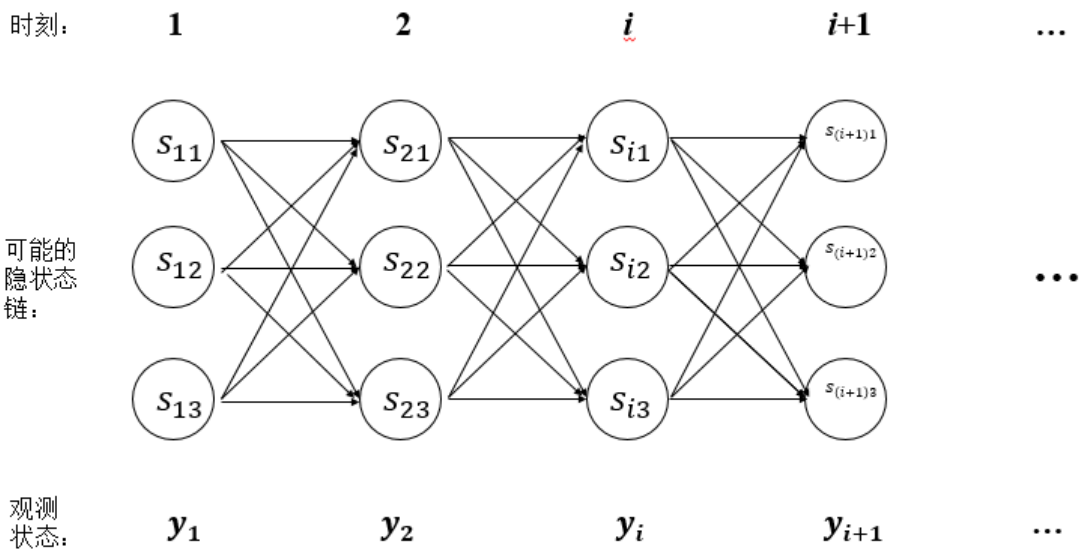
最后得出来的$P(Y|X)$就是最终结果了。

2.3 jieba分词



- **Viterbi算法**

  由于直接解上文的隐马尔可夫方程组的**计算复杂度太大**（$n*|S|^n$）了，因此我们将问题转化为：寻找**有向无环图（篱笆网络)**当中两个点之间的最短路径。我们使用一个动态规划算法——“Viterbi”算法来减少计算复杂度。



  如上图所示，每一列可以看做一簇地点，初始时你可以选择从第一列的任意一个点出发，每个点会给不同的初始金币值。然后沿着边到最后一列任何一个定点上，每经过一条边会获取相应的金币值，问最后能获取的最大金币值。

  其实这个问题和刚才求条件概率最大值是同一个问题。可以采用动态规划的方法来解决：

  设$init_gold[i]$就是第一列第i个顶点出发的初始值。设$max_gold[i][j]$表示到达第i列第j个顶点时能获得的最大金币值，$gold[i][j][k]$表示走第i列第j个定点到第$i+1$列第k个顶点的边能获得的金币值。则有如下递推公式：

  $$max\_gold[i+1][j] = max(max\_gold[i][0] + gold[i][0][j], max\_gold[i][1] + gold[i][1][j], \dots, max_gold[i][n] + gold[i][n][j])$$

  从上面的公式计算最大值，时间复杂度降为：

$$n * |S|^2$$

求得最大值后，反推即可知道每列走的是哪个顶点，也就是分词中的标注序列。

三、程序设计（包括数据集的构建，处理以及分析）

- 程序主要分为三个部分：数据处理层、分词输出层和词云生成层

3.1 数据处理层

- 主要负责提取数据库（database）中的文本，提取需要的标签里的内容，进行简单的数据清洗和数据处理，并将其转化为易处理的文本文档
- 我使用的微信聊天记录的数据集破译之后如下：

msgId	msgSvrId	type	status	isSend	isShowTime	createTime	talker	content	imgPath	reserved	lvbuffer	transConte	transBranc	talkerId	bizClient	bizChatId	bizChatUs	msgSeq	flag
2	5.482E+18	1	3	0		1.623E+12	2205430430	wxid_wt7mwk66tfk01 2: 今天必须加一个			{			70		-1		719262546	0
3	5.419E+18	1	3	0		1.623E+12	2205430430	wxid_e00mqa66tapg2 2: 老板要不要考虑做米面包或者贝果？			{			70		-1		719262553	0
4	2.324E+18	1	3	0		1.623E+12	2205430430	wxid_e00mqa66tapg2 2: 斑斓米面包yyds			{			70		-1		719262554	0
5	5.404E+18	1	3	0		1.623E+12	2205430430	wxid_wt7mwk66tfk01 2: 我们研究研究			{			70		-1		719262559	0

- 只需要根据talker选择想要提取的用户，并将相应的content从表格里面提取出来变成文本文档：

```
#!/usr/bin/python
import pandas

data0 = pandas.read_csv('chat_log.csv', usecols=[7,8])
# 读取我们上一步提取出来的csv文件，这里要改成你自己的文件名
# print(data0)
# 根据数据集中的标签提取指定列的数据（根据talker提取content）
data1 = data0.groupby(by='talker').apply(lambda x:['\n'.join(x['content'])])
print(data1)

cft = data1.at['wxid_utx0senetvbc22']
# 输出为文本文档
data=open("聊天记录_cft.txt",'w+',encoding='utf-8')
for i in cft:
    data.write(i+'\n')
    # 每行聊天记录换行
data.close()
```

- 得出结果如下：

```
应该是款的原因
ooooooooooh!
全体
起立!
<msg><img encryver="" cdnthumbheight="0" cdnthumbwidth="0" cdnmidheight="0" cdnmidwidth="0" cdnhdheight="0" cdnhdwidth="0" md5="" length="117516" cdnthumbaeskey="bc7222b8
真有点激动了
<?xml version="1.0"?>
<msg>
        <appmsg appid="wxcb8d4298c6a09bcb" sdkver="0">
                <title>【官方MV】G.E.M.邓紫棋《超能力》</title>
                <des>UP主：GEM邓紫棋工作室
```

- 利用正则化将文本中的汉字提取出来：

```
import re
import codecs
# 将聊天记录转变为纯汉字，除去微信结构中的符号
with codecs.open('聊天记录_cft.txt', 'r' ,encoding = 'utf-8') as file:
    f = file.read()
    k = re.findall(r'[\u4e00-\u9fa5]+', f)

data=open("聊天记录_cft_hanzi.txt",'w+',encoding='utf-8')
for i in k:
    data.write(i+'\n')
data.close()
```

- 得出结果如下：

```
应该是款的原因
全体
起立
真有点激动了
官方
邓紫棋
超能力
主
邓紫棋工作室
万播放
```

3.2 分词输出层

- 分词层主要将文本文档中的大片文字分割成一个一个的词语便于后续继续操作，后续可以用来统计词频、翻译等工作
  - 分词层使用了jieba进行分词，原理如上文
  - 主要包含两个函数：读取文本分词、读取自己的停词库、词库重组分词结果并统计词频

```
# coding:utf-8
import jieba
import numpy
import codecs
import pandas
import matplotlib.pyplot as plt
from scipy.misc import imread
import matplotlib.pyplot as plt

def load_file_segment():
    # 读取文本文件并分词
    f = codecs.open(u"聊天记录_cft_hanzi.txt",'r',encoding='utf-8')
    # 打开文件
    content = f.read()
    # 读取文件到content中
    f.close()
    # 关闭文件
    segment=[]
    # 保存分词结果
    segs=jieba.cut(content)
    # 对整体进行分词
    for seg in segs:
        if len(seg) != 1 and seg != '\r\n':
            # 如果说分词得到的结果非单字，且不是换行符，则加入到数组中
            segment.append(seg)
    return segment

def get_words_count_dict():
```

```
    segment = load_file_segment()
    # 获得分词结果
    df = pandas.DataFrame({'segment':segment})
    # 将分词数组转化为pandas数据结构
    stopwords =
pandas.read_csv("stopwords.txt",index_col=False,quoting=3,sep="\t",names=
['stopword'],encoding="utf-8")
    # 加载停用词
    df = df[~df.segment.isin(stopwords.stopword)]
    # 如果不是在停用词中
    words_count = df.groupby('segment').agg(
        计数=pandas.NamedAgg(column='segment',
aggfunc='size')).reset_index().sort_values(
        by='计数', ascending=False)

    # 按词分组，计算每个词的个数
    words_count = words_count.reset_index().sort_values(by="计
数",ascending=False)
    # reset_index是为了保留segment字段，排序，数字大的在前面
    return words_count
```

3.3 词云生成层

- 词云生成层主要按照分出来的词统计词频，用词语出现的频率进行词云绘制，绘制出出现频率最高的一定数量的词
    - 绘制词云使用wordcloud库，读取频率之后选取图片并显示结果

```
# coding:utf-8
import jieba
import numpy
import codecs
import pandas
import matplotlib.pyplot as plt
from scipy.misc import imread
import matplotlib.pyplot as plt
from wordcloud import WordCloud, ImageColorGenerator
from wordcloud import WordCloud
words_count = get_words_count_dict()

bimg = imread('ai.jpg')
# 读取我们想要生成词云的模板图片
wordcloud = WordCloud(background_color='white', mask=bimg,
font_path='simhei.ttf')
# 获得词云对象，设定词云背景颜色及其图片和字体

# 如果你的背景色是透明的，请用这两条语句替换上面两条
# bimg = imread('ai.png')
# wordcloud = WordCloud(background_color=None, mode='RGBA', mask=bimg,
font_path='simhei.ttf')

words = words_count.set_index("segment").to_dict()
# 将词语和频率转为字典
wordcloud = wordcloud.fit_words(words["计数"])
# 将词语及频率映射到词云对象上
bimgColors = ImageColorGenerator(bimg)
# 生成颜色
```

```
plt.axis("off")
# 关闭坐标轴
plt.imshow(wordcloud.recolor(color_func=bimgColors))
# 绘色
plt.show()
```

- 显示结果如下：

原图：

四、结果分析及拓展应用

4.1 结果分析

- 通过获取一个人的微信聊天记录并对其进行分析，我们可以获取这个人与不同对象最近的聊天热点，通过对词源的追溯，我们甚至可以找到聊的最多的话题在哪个语境下、在哪个时间点，对于以后的交流（投其所好），对以往的纪念，都有很好的意义。
- 在搜索过程中甚至找到微信转账记录是否可以充当民事诉讼的讨论，通过对于不同阶段、不同对象聊天记录的处理，甚至可以分析出参访者与不同人的关系亲疏、关系网，讨论重点，能做的切入点很多。

4.2 拓展应用之用户画像

- 可以针对微信用户对象对不同群体的聊天记录进行划分，从而得到这个用户的用户画像（本文以我本人为例）
    - 家庭（提醒我要多跟爸妈聊天了）

- 同学

我发现转账记录实在太多了（酒肉朋友）所以把转账记录剔除掉再看最后的结果

可以得出一些结论比如我玩基金（关注新能源和半导体）、喜欢打篮球和斯诺克等等

4.3 内容检索/关键词生成

(程序比较长就附在了文件夹里面)

- 当我们想要知道一本书中某些特定内容时（以三国演义为例）
  - 我们可以简单地对整体的词频进行提取并生成词云，对书本的整体情况进行把控

  - 也可以更深入地关注某些特定的分类
  - 可以通过构建自己的词典（比如三国演义中的所有人名、地名来进行筛选）
    - 文章中出现的top30人物（谁才是真正的大佬）
    - 文章中出现频次最多的几个地名（三国时期经济中心/主战场）等等很多功能

丁 仪（正礼） 丁 奉（承渊） 丁 原（建阳） 丁 谧（彦靖）
丁 廙（敬礼） 于 禁（文则） 士孙瑞（君荣） 山 涛（巨源）
卫 瓘（伯玉） 马 钧（翁叔） 马 良（季常） 马 忠（德信）
马 超（孟起） 马 谡（幼常） 马 腾（寿成） 王 允（子师）
王 双（子全） 王 平（子均） 王 匡（公节） 王 戎（睿冲）
王 观（伟台） 王 甫（国山） 王 连（文仪） 王 沈（处道）
王 肃（子雍） 王 修（叔治） 王 浑（玄冲） 王 路（文舒）
王 颀（孔硕） 王 祥（休徵） 王 朗（景兴） 王 基（伯舆）
王 谋（元泰） 王 粲（仲宣） 王 睿（士治） 韦 康（元将）
太史慈（子义） 毛 玠（孝先） 公孙度（升济） 公孙瓒（伯圭）
文 钦（仲若） 文 聘（仲业） 尹 奉（次曾） 邓 艾（士载）
邓 芝（伯苗） 邓止飏（玄茂） 孔 伷（公绪） 孔 昱（世元）

网上找到的三国演义中出现的大部分名字

（其实出场次数特别少的也没有统计的必要）

  - 最后得出来的只含人名的筛选结果

4.3 翻译

- 可以将一段中文进行中文分词之后，调用百度/谷歌的翻译API，将中文翻译成英文。这个应用就比较基础了，直接将一句话或者一段话输入即得出结果，不再进行赘述。

五、心得体会

一直以来就对机器学习十分感兴趣，但是之前一直是走“野路子”，也就是只管应用，需要什么找什么，找着什么用什么，或者哪个效果好用哪个。配环境、跑程序等等一些实操算是熟练，但是算法原理、基本框架一直没有搭建起来。

突然发现电气学院居然开了一门这样的课程，自然是见猎心喜，希望依靠这门课补上一直以来理论侏儒的阴霾，而这门课也没有让我失望。尽管课程没有讲很多具体算法具体问题（实际上这样的课也会很枯燥），但是胜在知识面非常广，从前面的基本定义到后面复杂的神经网络甚至自然语言处理，整个框架建立的非常完善，我也终于跟着课读完了我一直想读却没有读下去的西瓜书。

在完成大作业的过程中，我也体会到了知识储备健全带来的好处。在选题的时候，我第一反应就是想做完之前一直想做却没有完成的微信聊天记录词云，在查询过程中发现核心算法刚好和我学过的知识对应上了，也非常感谢老师同意我使用这个例子。

在处理微信数据的过程中，正则化的思想贯穿了整个处理过程：如何处理符号、乱码，如何提取需要的汉字、英文等内容，如何将会带来歧义的离群点剔除，正则化提取汉字等等。在自然语义分析中则运用了树的思想，有向无环图、隐马尔可夫模型的运用虽然没有学到（有点可惜），但是在我完成大作业的过程中，仔细阅读了这方面的内容，完成了最后一块拼图。在最后最短路径规划的过程中，使用了维克比动态线性规划的方法，这也与前期学习线性规划的、损失函数等等概念密切相关。

当你知道了一些原理的时候，去复现一些算法、实现一些功能、使用一些库函数是非常有成就感的一件事情，就像正规军和杂牌军的对比一样，你会感觉做的一些工作是有一个整体的指导的，他不会告诉你在每一步具体要干什么，但是会指导你下一步应该往哪个方向走，再回味以前做过的一些路径规划、人脸物品识别的算法，会生出新的体会。

再次感谢老师和助教学长！

六、参考文献

- **维基百科**

贝叶斯定理　马尔科夫链　HMM 简介

EM 最大期望算法动态规划算法　Viterbi 算法Baum welch 算法　CRF-条件随机场

- **博客**（部分）

1. https://www.jianshu.com/p/ba06032c466e
2. https://zhuanlan.zhihu.com/p/24322275
3. https://zhuanlan.zhihu.com/p/87632700
4. https://zhuanlan.zhihu.com/p/40502333
5. https://www.cnblogs.com/chenhuabin/p/13521253.html
6. https://pythondict.com/python-paintings/python-qixi-wechat-wordcloud/
7. https://blog.csdn.net/u010785550/article/details/108669652
8. https://zhidao.baidu.com/question/192918537.html

- **文献**

[1] 李航 统计学习方法 清华大学出版社

[2] 隐马尔科夫模型HMM（一/二/三/四） 刘建平Pinard