自然语言处理作业

第一次大作业 2021.10.20

07111908 班 1120193583 张昊杰

一、实验内容：

- 实现基于词典的分词方法和统计分词方法：两类方法中各实现一种即可；
- 对分词结果进行词性标注，也可以在分词的同时进行词性标注；
- 对分词及词性标注结果进行评价，包括 4 个指标：正确率、召回率、F1 值和效率。

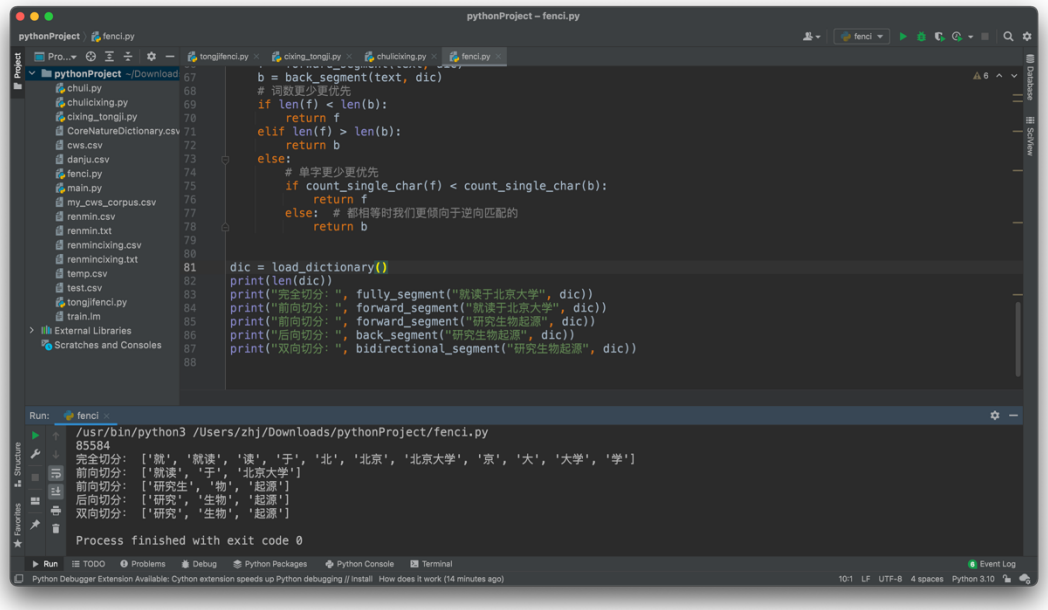
二、实现平台

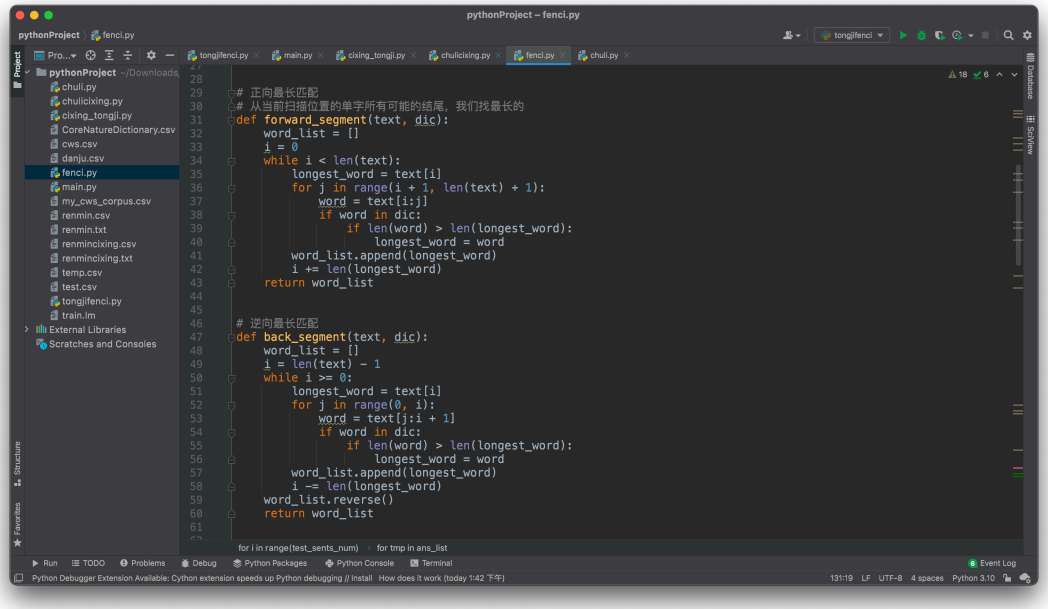
MacBook Air M1，全部使用 Python 进行实验

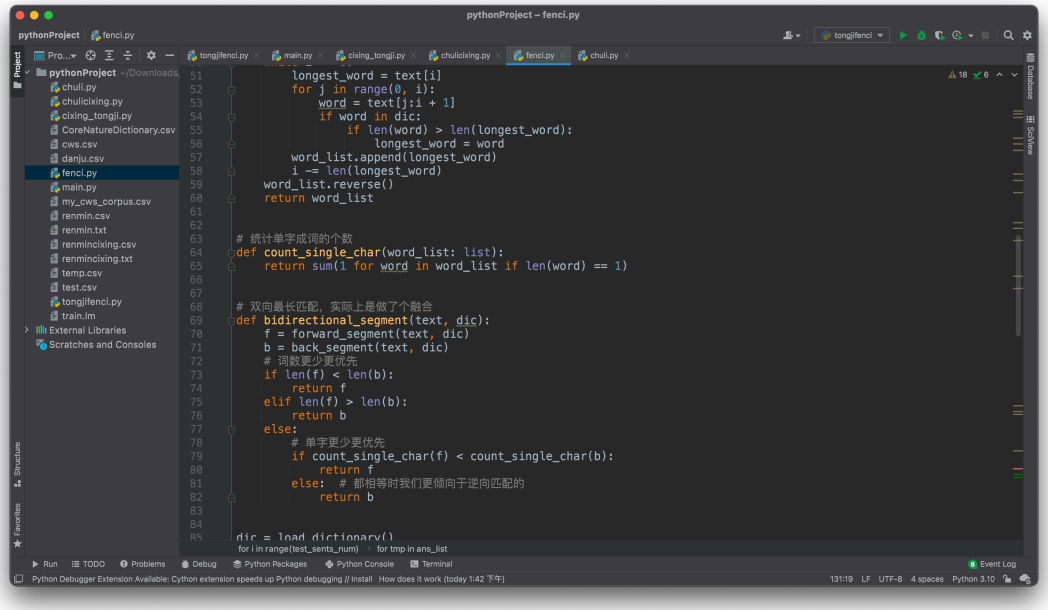
三、实验过程

1. 基于词典的分词方法中，我们使用了四种分词方法，即完全切分式，正向最长匹配，逆向最长匹配，双向最长匹配。此处代码见附录 1。

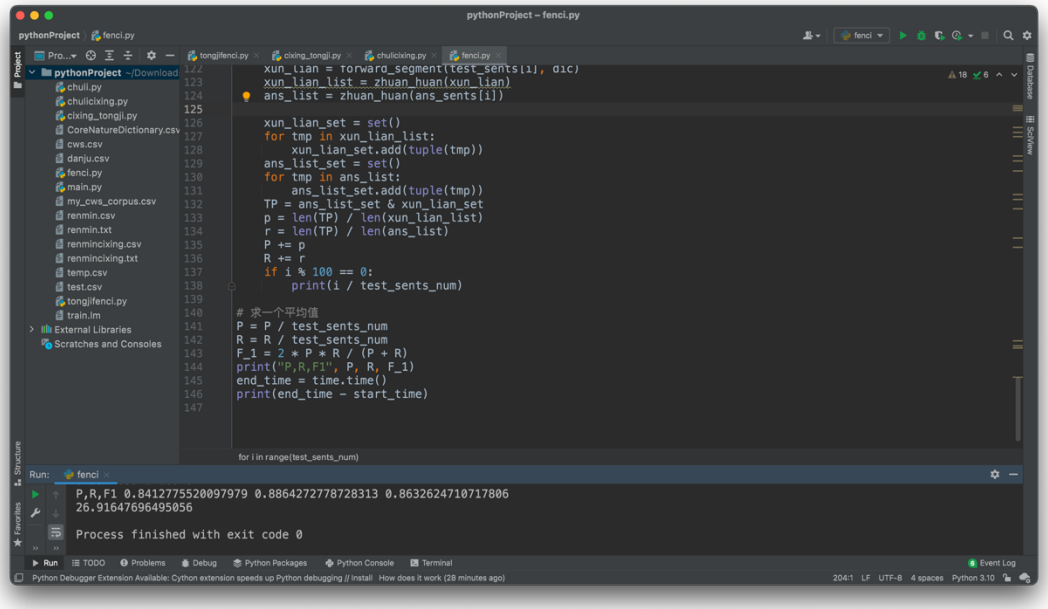
这里的词典我选择使用了北京大学统计好的词典作为词典参考来进行实验。



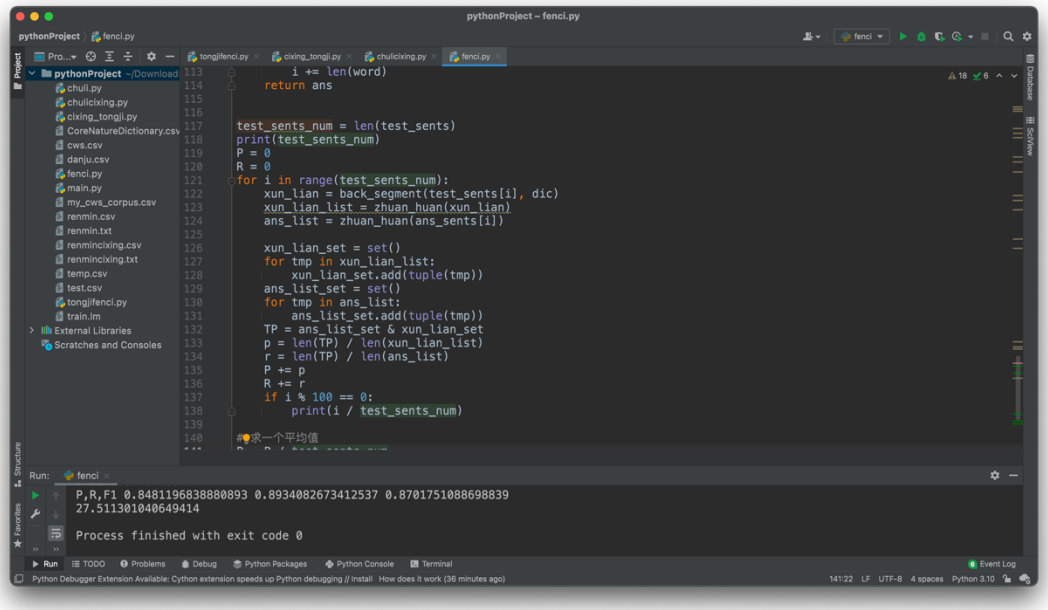




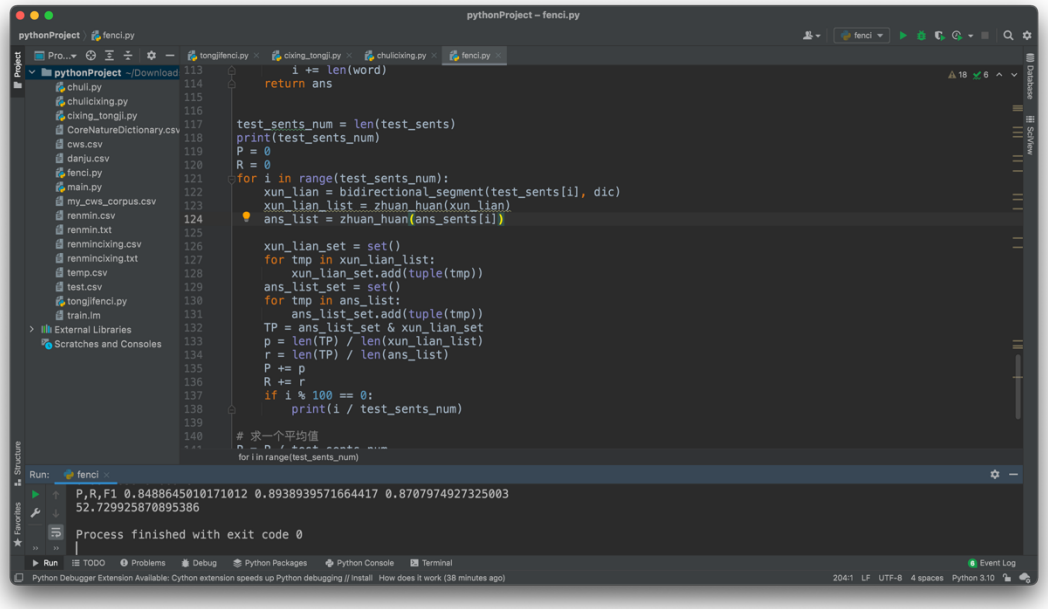
我们随意输入几个句子并输出结果，根据结果来看，各个方法分词的效果还算不错。接下来我们使用人民日报的分好的语料库进行一个全篇的预测。



首先我们使用前向切分的方法，可以看到 P，R，F1 三个值，以及用时 26.92s。



接下来使用后向切分的分法，正确率微微提升，同时用时 27.51s。



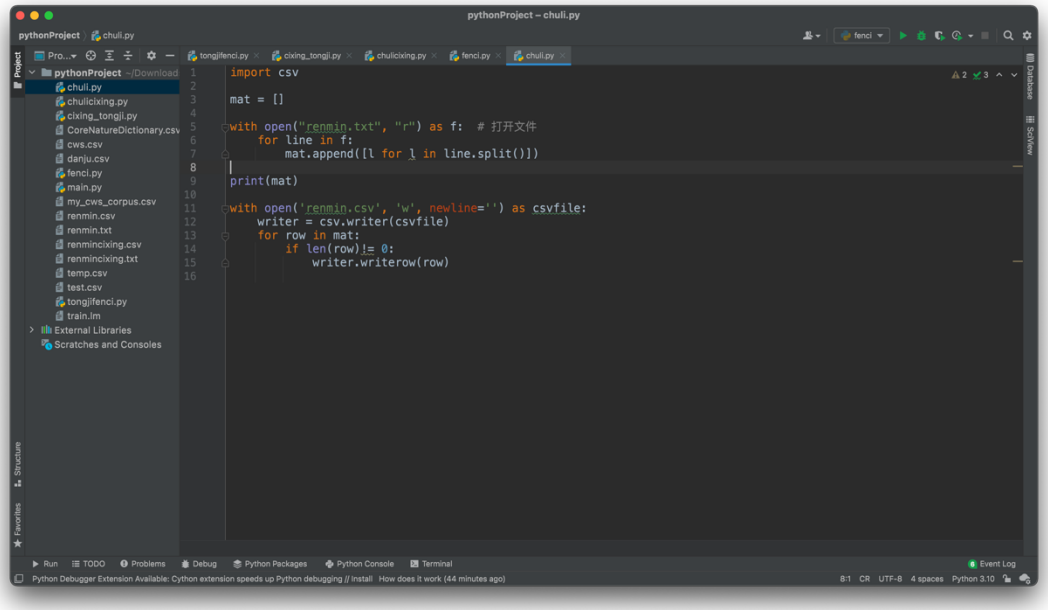
最后使用双向切分的方法，正确率几乎不变，用时 52s，几乎翻倍的时间。

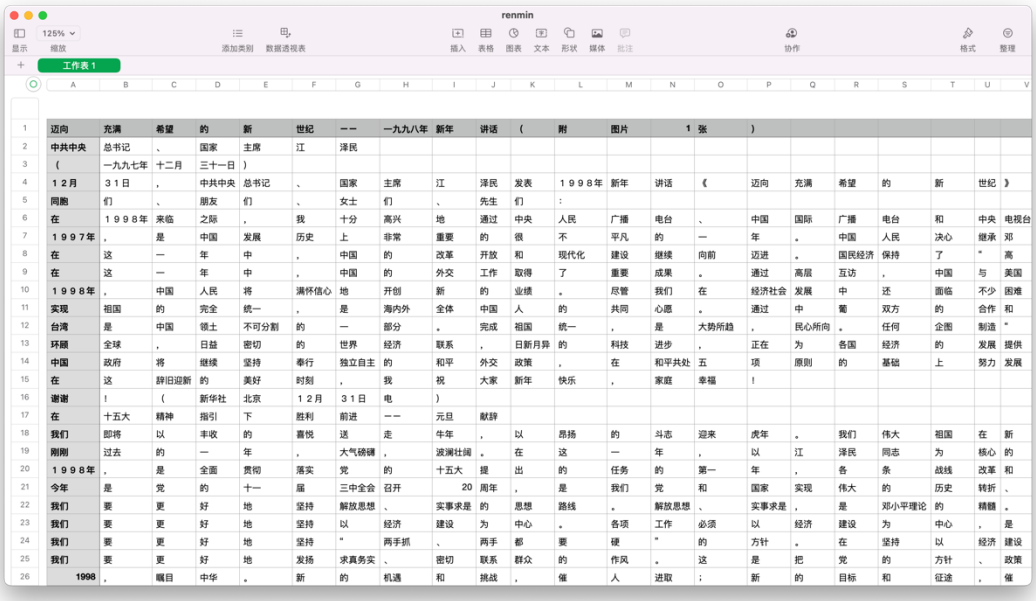
	P	R	F1	Time
前向	0.8413	0.8864	0.8864	26.91
后向	0.8482	0.8934	0.8934	27.51
双向	0.8489	0.8939	0.8708	52.73

可见，基于词典的分词方式，不论哪种方法，正确率基本稳定在这个范围上。效果还算可以。

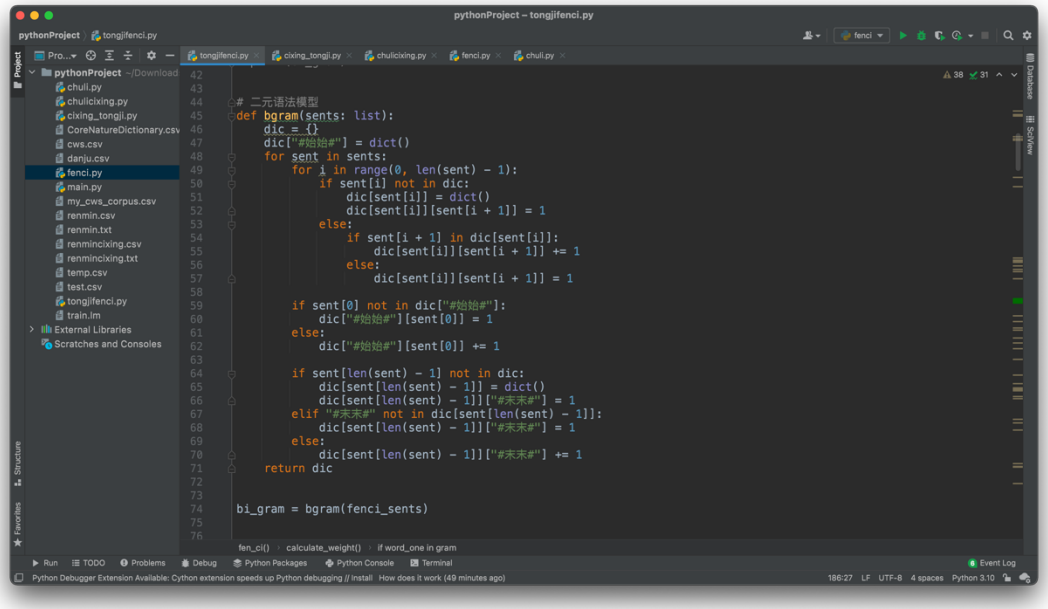
2. 基于语料的统计分词，使用二元语法模型来构建词库，然后将句子生成词网，在使用 viterbi 算法来计算最优解，在这当中使用+1 法来处理前后之间概率。代码放在附录 2

首先我们处理原始文件，将人民日报语料库的带空格的文件处理成 csv 的表格形式方便我们进行分词的整理。



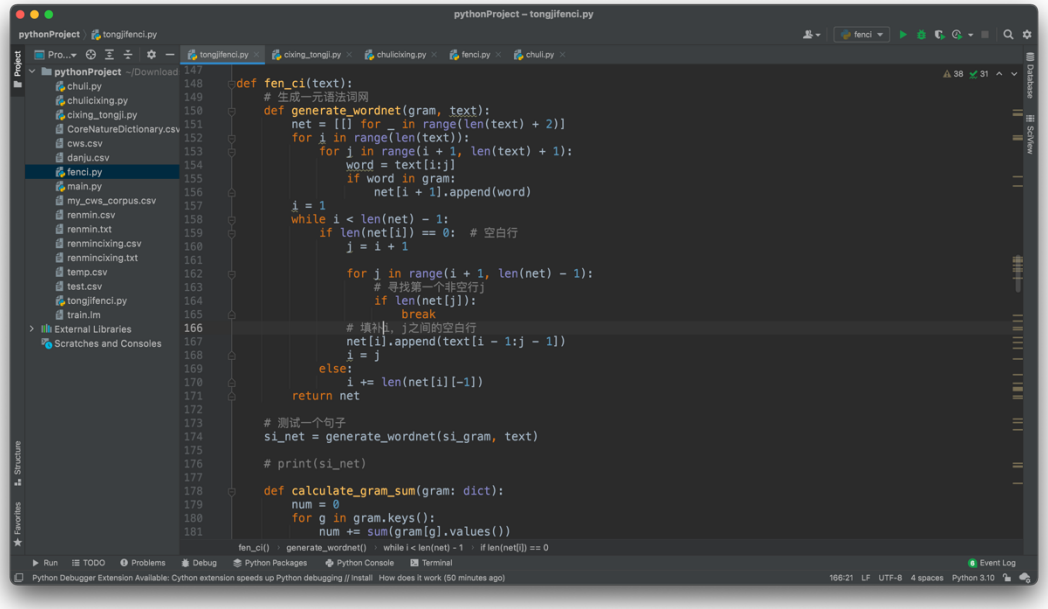


这里是二元语法模型的核心代码：



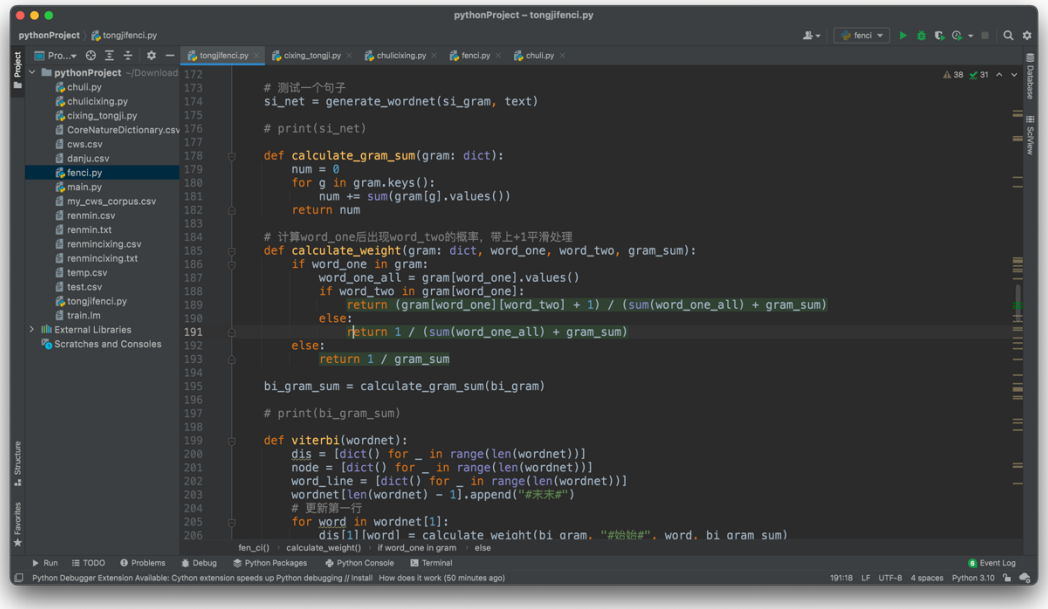
```python
# 二元语法模型
def bgram(sents: list):
    dic = {}
    dic["#始始#"] = dict()
    for sent in sents:
        for i in range(0, len(sent) - 1):
            if sent[i] not in dic:
                dic[sent[i]] = dict()
                dic[sent[i]][sent[i + 1]] = 1
            else:
                if sent[i + 1] in dic[sent[i]]:
                    dic[sent[i]][sent[i + 1]] += 1
                else:
                    dic[sent[i]][sent[i + 1]] = 1

        if sent[0] not in dic["#始始#"]:
            dic["#始始#"][sent[0]] = 1
        else:
            dic["#始始#"][sent[0]] += 1

        if sent[len(sent) - 1] not in dic:
            dic[sent[len(sent) - 1]] = dict()
            dic[sent[len(sent) - 1]]["#末末#"] = 1
        elif "#末末#" not in dic[sent[len(sent) - 1]]:
            dic[sent[len(sent) - 1]]["#末末#"] = 1
        else:
            dic[sent[len(sent) - 1]]["#末末#"] += 1
    return dic


bi_gram = bgram(fenci_sents)
```

这里是生成句子词网的核心代码：



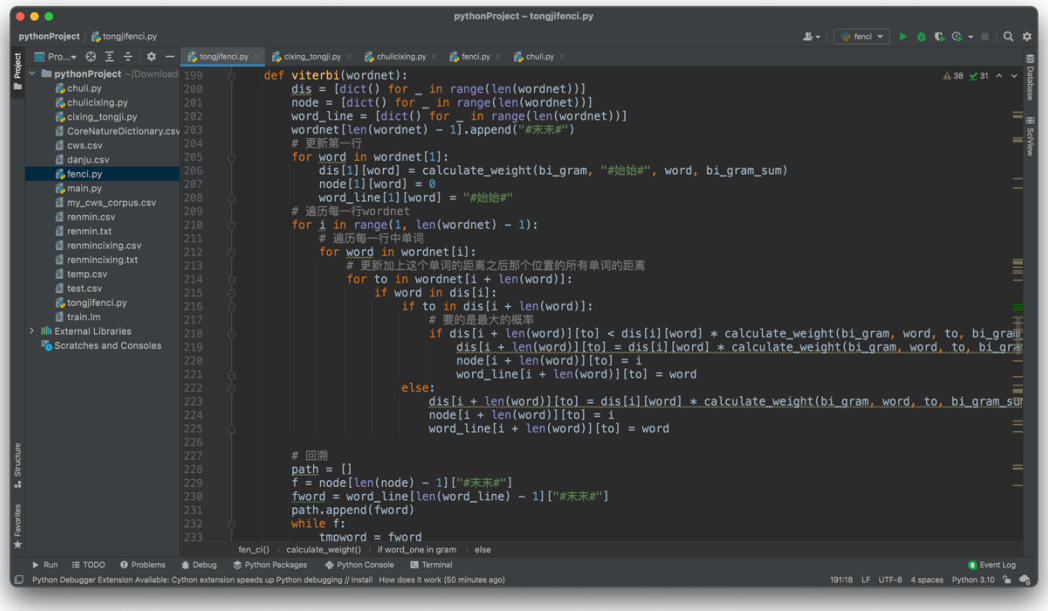
```python
def fen_ci(text):
    # 生成一元语法词网
    def generate_wordnet(gram, text):
        net = [[] for _ in range(len(text) + 2)]
        for i in range(len(text)):
            for j in range(i + 1, len(text) + 1):
                word = text[i:j]
                if word in gram:
                    net[i + 1].append(word)
        i = 1
        while i < len(net) - 1:
            if len(net[i]) == 0:  # 空白行
                j = i + 1

                for j in range(i + 1, len(net) - 1):
                    # 寻找第一个非空行j
                    if len(net[j]):
                        break
                # 填补i, j之间的空白行
                net[i].append(text[i - 1:j - 1])
                i = j
            else:
                i += len(net[i][-1])
        return net

    # 测试一个句子
    si_net = generate_wordnet(si_gram, text)

    # print(si_net)

    def calculate_gram_sum(gram: dict):
        num = 0
        for g in gram.keys():
            num += sum(gram[g].values())
```

这里是+1 平滑处理的核心代码：



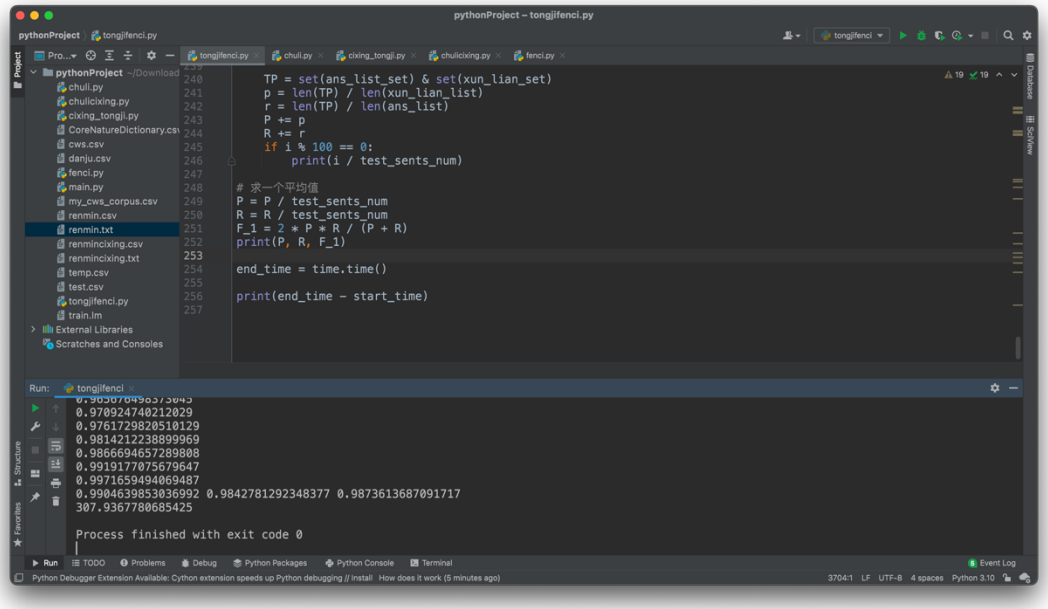
```
    # 测试一个句子
    si_net = generate_wordnet(si_gram, text)

    # print(si_net)

    def calculate_gram_sum(gram: dict):
        num = 0
        for g in gram.keys():
            num += sum(gram[g].values())
        return num

    # 计算word_one后出现word_two的概率，带上+1平滑处理
    def calculate_weight(gram: dict, word_one, word_two, gram_sum):
        if word_one in gram:
            word_one_all = gram[word_one].values()
            if word_two in gram[word_one]:
                return (gram[word_one][word_two] + 1) / (sum(word_one_all) + gram_sum)
            else:
                return 1 / (sum(word_one_all) + gram_sum)
        else:
            return 1 / gram_sum

    bi_gram_sum = calculate_gram_sum(bi_gram)

    # print(bi_gram_sum)

    def viterbi(wordnet):
        dis = [dict() for _ in range(len(wordnet))]
        node = [dict() for _ in range(len(wordnet))]
        word_line = [dict() for _ in range(len(wordnet))]
        wordnet[len(wordnet) - 1].append("#末末#")
        # 更新第一行
        for word in wordnet[1]:
            dis[1][word] = calculate_weight(bi_gram, "#始始#", word, bi_gram_sum)
```

这里是 viterbi 算法的核心代码：



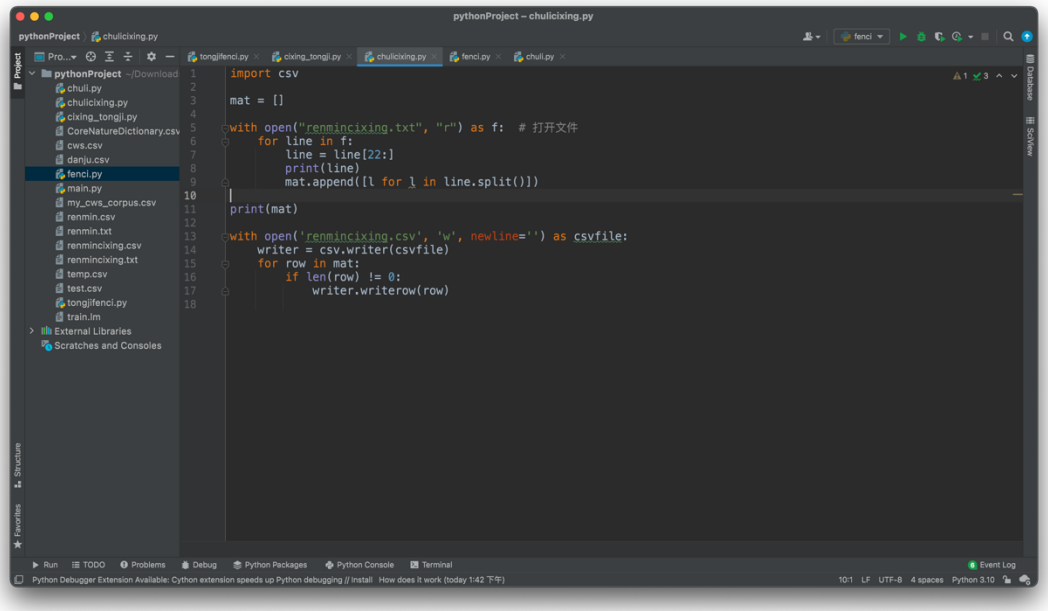
```
    def viterbi(wordnet):
        dis = [dict() for _ in range(len(wordnet))]
        node = [dict() for _ in range(len(wordnet))]
        word_line = [dict() for _ in range(len(wordnet))]
        wordnet[len(wordnet) - 1].append("#末末#")
        # 更新第一行
        for word in wordnet[1]:
            dis[1][word] = calculate_weight(bi_gram, "#始始#", word, bi_gram_sum)
            node[1][word] = 0
            word_line[1][word] = "#始始#"
        # 遍历每一行wordnet
        for i in range(1, len(wordnet) - 1):
            # 遍历每一行中单词
            for word in wordnet[i]:
                # 更新加上这个单词的距离之后那个位置的所有单词的距离
                for to in wordnet[i + len(word)]:
                    if word in dis[i]:
                        if to in dis[i + len(word)]:
                            # 要的是最大的概率
                            if dis[i + len(word)][to] < dis[i][word] * calculate_weight(bi_gram, word, to, bi_gram
                                dis[i + len(word)][to] = dis[i][word] * calculate_weight(bi_gram, word, to, bi_gra
                                node[i + len(word)][to] = i
                                word_line[i + len(word)][to] = word
                        else:
                            dis[i + len(word)][to] = dis[i][word] * calculate_weight(bi_gram, word, to, bi_gram_su
                            node[i + len(word)][to] = i
                            word_line[i + len(word)][to] = word

        # 回溯
        path = []
        f = node[len(node) - 1]["#末末#"]
        fword = word_line[len(word_line) - 1]["#末末#"]
        path.append(fword)
        while f:
            tmpword = fword
```

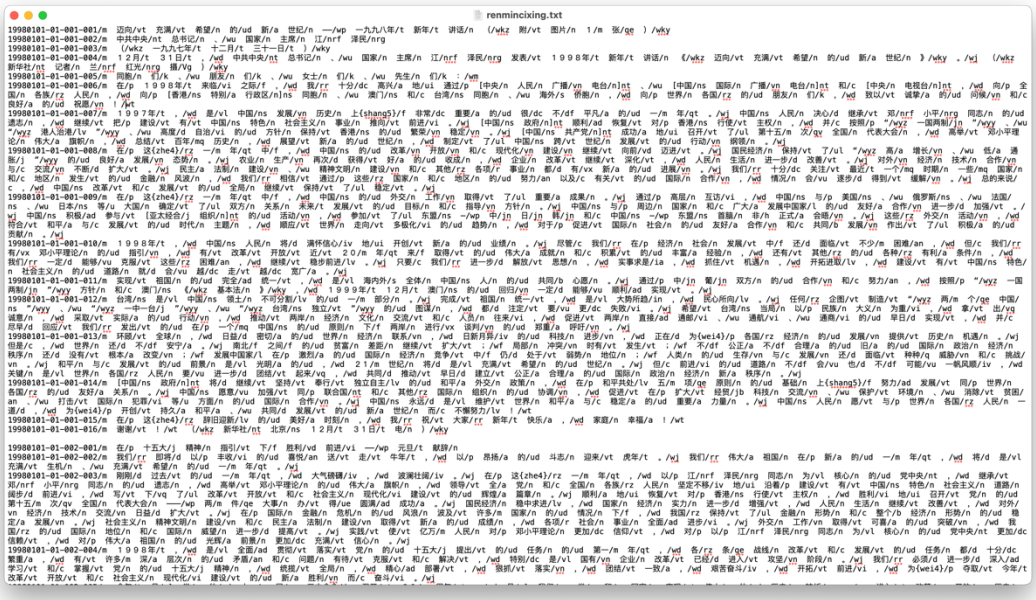
测试集和训练集均为人民日报语料库，接下来是最终训练出来的效果。
展示了 P，R，F1 的值
可以看到还是达到了一个近乎 99%的正确率，同时训练+预测共耗时 307。
P = 0.9905
R = 0.9843
F1 = 0.9873



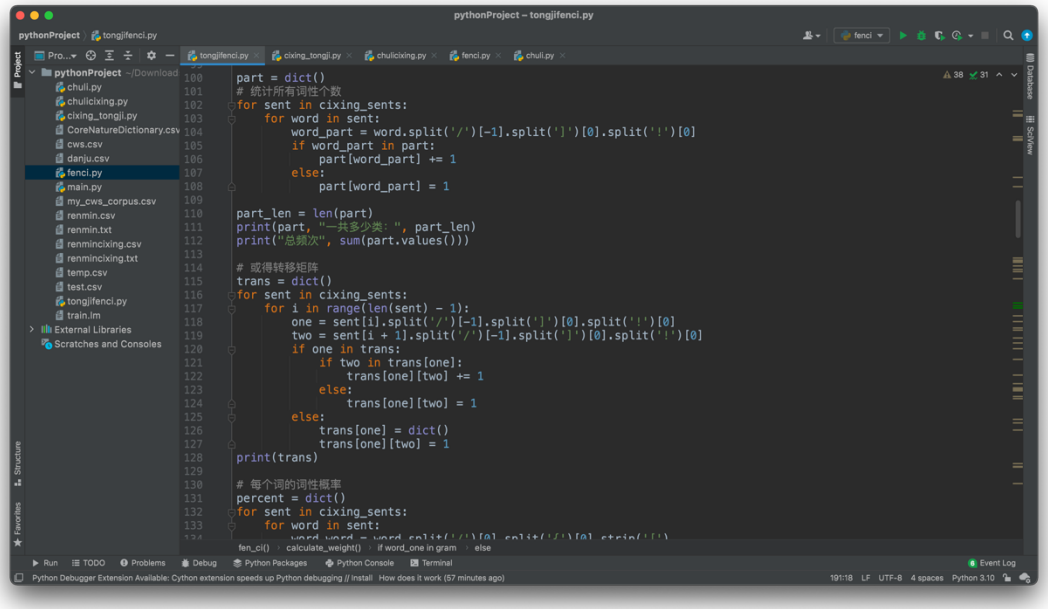
3. 对分词的结果进行词性标注。使用了统计的方法。此处代码同附录 2.

我们先对人民日报带词性标注的 txt 文件进行一个转换，转换成便于我操作的 csv 文件。



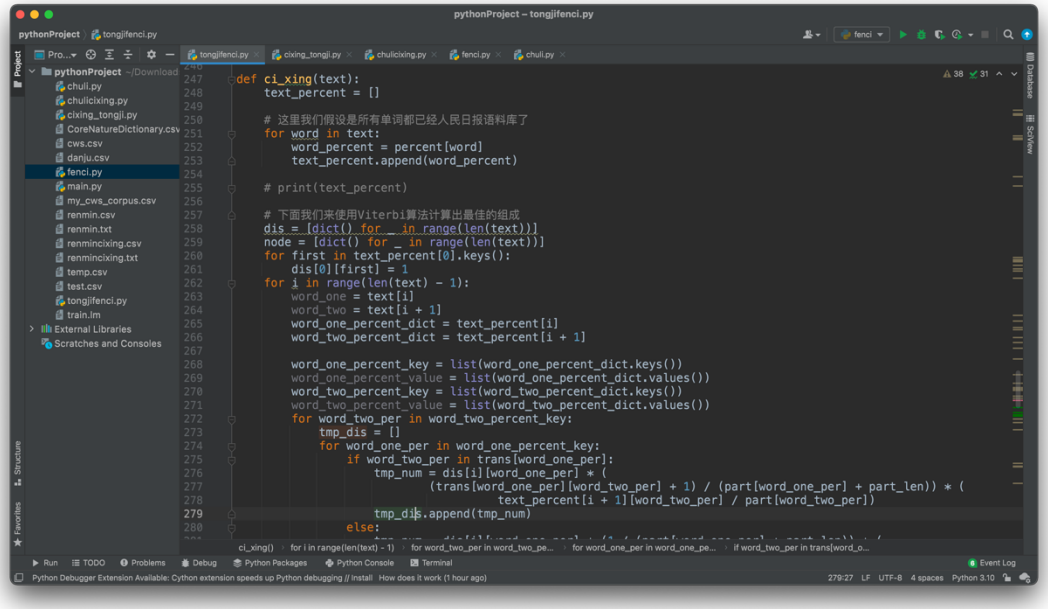


下面是我们词性标注的训练核心代码，统计所有词性和他们的个数，获得相应的词性转移矩阵，在统计每个词语的词性概率。



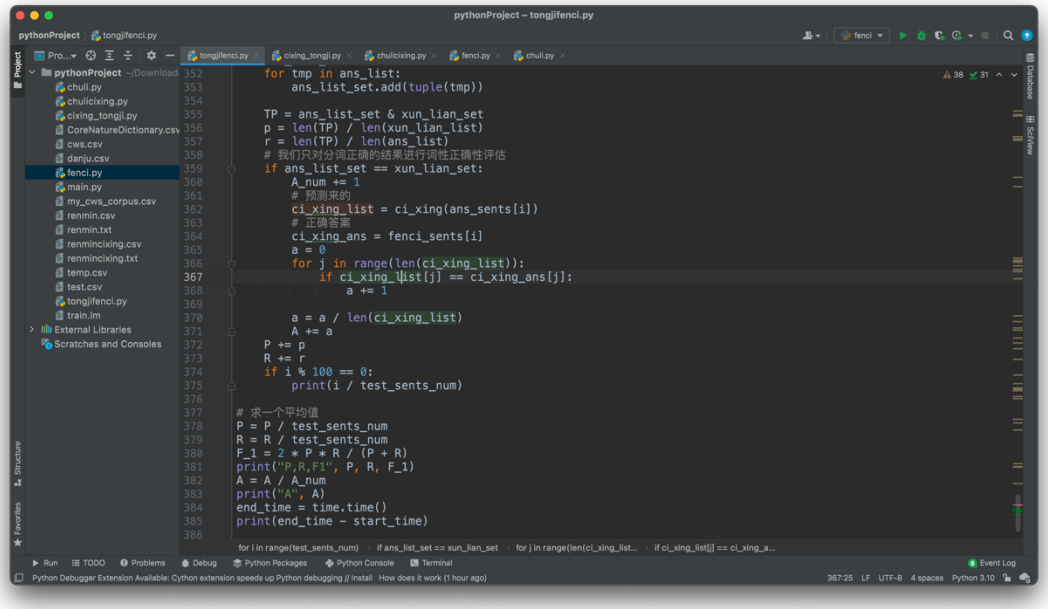
```
part = dict()
# 统计所有词性个数
for sent in cixing_sents:
    for word in sent:
        word_part = word.split('/')[-1].split(']')[0].split('!')[0]
        if word_part in part:
            part[word_part] += 1
        else:
            part[word_part] = 1

part_len = len(part)
print(part, "一共多少类：", part_len)
print("总频次", sum(part.values()))

# 或得转移矩阵
trans = dict()
for sent in cixing_sents:
    for i in range(len(sent) - 1):
        one = sent[i].split('/')[-1].split(']')[0].split('!')[0]
        two = sent[i + 1].split('/')[-1].split(']')[0].split('!')[0]
        if one in trans:
            if two in trans[one]:
                trans[one][two] += 1
            else:
                trans[one][two] = 1
        else:
            trans[one] = dict()
            trans[one][two] = 1
print(trans)

# 每个词的词性概率
percent = dict()
for sent in cixing_sents:
    for word in sent:
```

这里是推测词性的核心代码：

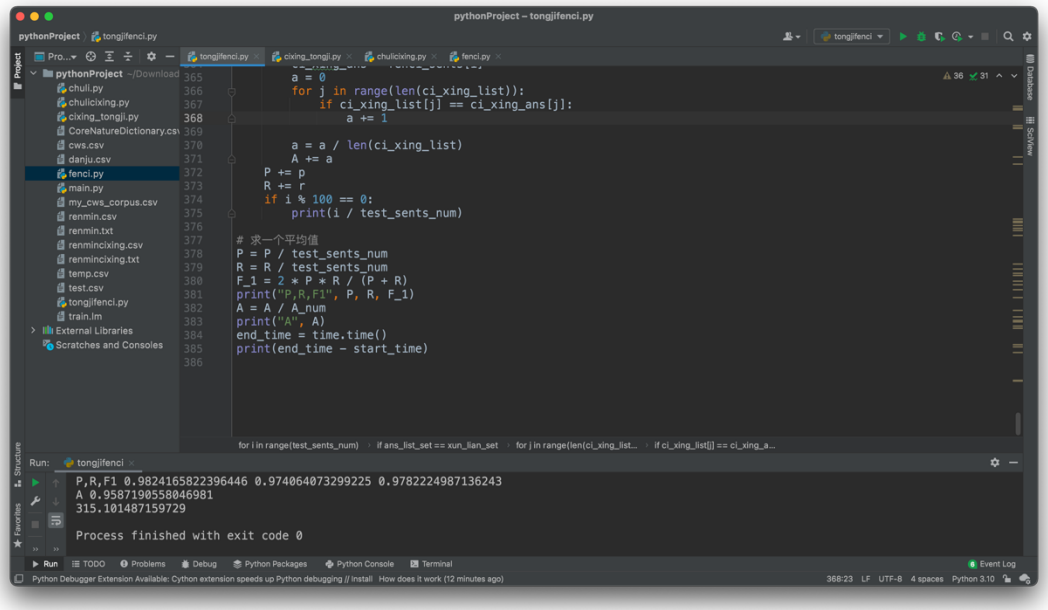


```
def ci_xing(text):
    text_percent = []

    # 这里我们假设是所有单词都已经人民日报语料库了
    for word in text:
        word_percent = percent[word]
        text_percent.append(word_percent)

    # print(text_percent)

    # 下面我们来使用Viterbi算法计算出最佳的组成
    dis = [dict() for _ in range(len(text))]
    node = [dict() for _ in range(len(text))]
    for first in text_percent[0].keys():
        dis[0][first] = 1
    for i in range(len(text) - 1):
        word_one = text[i]
        word_two = text[i + 1]
        word_one_percent_dict = text_percent[i]
        word_two_percent_dict = text_percent[i + 1]

        word_one_percent_key = list(word_one_percent_dict.keys())
        word_one_percent_value = list(word_one_percent_dict.values())
        word_two_percent_key = list(word_two_percent_dict.keys())
        word_two_percent_value = list(word_two_percent_dict.values())
        for word_two_per in word_two_percent_key:
            tmp_dis = []
            for word_one_per in word_one_percent_key:
                if word_two_per in trans[word_one_per]:
                    tmp_num = dis[i][word_one_per] * (
                            (trans[word_one_per][word_two_per] + 1) / (part[word_one_per] + part_len)) * (
                                      text_percent[i + 1][word_two_per] / part[word_two_per])
                    tmp_dis.append(tmp_num)
                else:
```

下面是对分词结果的正确性评估，这里我使用了书中使用的方法，仅计算一个 Accuracy 作为正确率评估标准，同时我只对之前分词正确的结果结果进行词性评估，这样可以避免其它的错误。



通过检测结果来看，词性标注的正确率大概在 95%左右，同时共计耗时 315s。还算一个相对不错的效果。



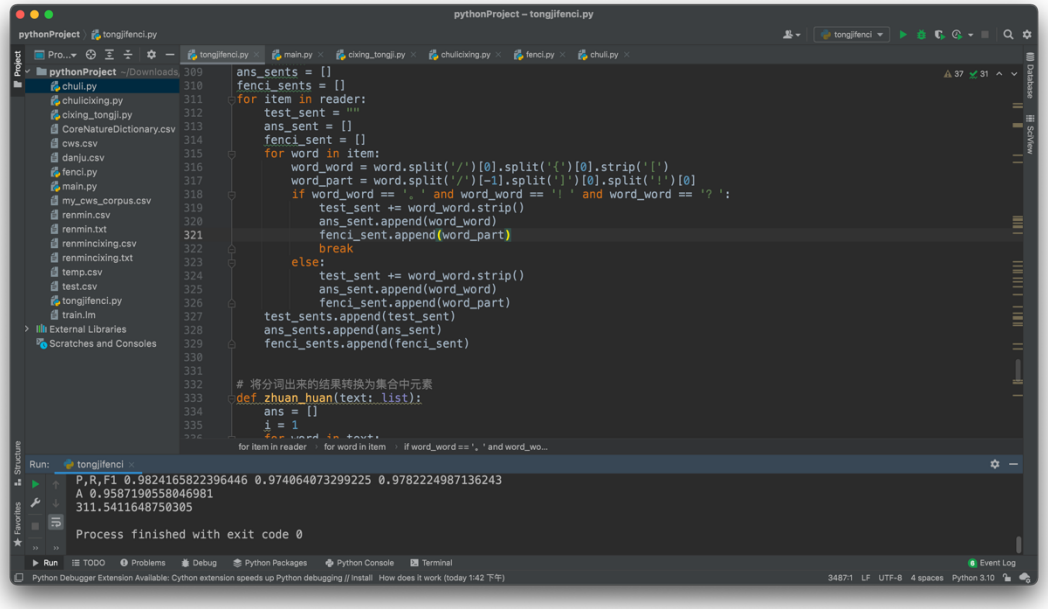
四、实验遇到的问题与麻烦

1. 首先就遇到的难题是处理人民日报那个 txt，一开始没有想到很方便的办法把它处理为 list 数据，最后通过转换为 csv 格式，在直接导入到 list 当中确实简化了不少步骤。

2. 处理二元语法模型的时候，这个“#始始#”，“#末末#”的处理确实费了不少功夫，总是在这里缺少一些项目，debug 了很长时间。

3. Viterbi 算法中存在的字典中缺少的值如何去补充，不存在的转化概率该如何去补充。

4. 这个也是尚未处理的问题，我做预测的时候仍然采用原人民日报的换行格式，即一行就为一个句子，但这个很明显并不是我们日常意义上所理解的句子，我们通常理解的句子中，是以“。”，“！”，“？”作为句子结尾的。我重新刷洗了数据，并让句子按照这三个标点符号作为结尾来预测数据。



最终得出的训练结果中，正确率不变，说明我们能很好的将标点符号分开，但是时间却缩小了一些，说明缩短句子确实有助于减少大规模词网的运算量。

五、附录

附录一：基于词典的分词方法：

```
import csv
import time

start_time = time.time()


# 读入字典
def load_dictionary():
    word_list = set()
    csvFile = open("test.csv", "r")
    reader = csv.reader(csvFile)
    for item in reader:
        word_list.add(item[0])
    return word_list


# 完全切分式中文分词
# 如果在词典中则认为是一个词
def fully_segment(text, dic):
```

```
    word_list = []
    for i in range(len(text)):
        for j in range(i + 1, len(text) + 1):
            word = text[i:j]
            if word in dic:
                word_list.append(word)
    return word_list


# 正向最长匹配
# 从当前扫描位置的单字所有可能的结尾，我们找最长的
def forward_segment(text, dic):
    word_list = []
    i = 0
    while i < len(text):
        longest_word = text[i]
        for j in range(i + 1, len(text) + 1):
            word = text[i:j]
            if word in dic:
                if len(word) > len(longest_word):
                    longest_word = word
        word_list.append(longest_word)
        i += len(longest_word)
    return word_list


# 逆向最长匹配
def back_segment(text, dic):
    word_list = []
    i = len(text) - 1
    while i >= 0:
        longest_word = text[i]
        for j in range(0, i):
            word = text[j:i + 1]
            if word in dic:
                if len(word) > len(longest_word):
                    longest_word = word
        word_list.append(longest_word)
        i -= len(longest_word)
    word_list.reverse()
    return word_list


# 统计单字成词的个数
```

```python
def count_single_char(word_list: list):
    return sum(1 for word in word_list if len(word) == 1)


# 双向最长匹配，实际上是做了个融合
def bidirectional_segment(text, dic):
    f = forward_segment(text, dic)
    b = back_segment(text, dic)
    # 词数更少更优先
    if len(f) < len(b):
        return f
    elif len(f) > len(b):
        return b
    else:
        # 单字更少更优先
        if count_single_char(f) < count_single_char(b):
            return f
        else:  # 都相等时我们更倾向于逆向匹配的
            return b


dic = load_dictionary()
print(len(dic))
print("完全切分：", fully_segment("就读于北京大学", dic))
print("前向切分：", forward_segment("就读于北京大学", dic))
print("前向切分：", forward_segment("研究生物起源", dic))
print("后向切分：", back_segment("研究生物起源", dic))
print("双向切分：", bidirectional_segment("研究生物起源", dic))

test_file = open("renmin.csv", "r")
reader = csv.reader(test_file)
test_sents = []
ans_sents = []
for item in reader:
    test_sent = ""
    ans_sent = []
    for word in item:
        test_sent += word.strip()
        ans_sent.append(word)
    test_sents.append(test_sent)
    ans_sents.append(ans_sent)


# 将分词出来的结果转换为集合中元素
```

```
def zhuan_huan(text: list):
    ans = []
    i = 1
    for word in text:
        ans.append([i, i + len(word) - 1])
        i += len(word)
    return ans


test_sents_num = len(test_sents)
print(test_sents_num)
P = 0
R = 0
for i in range(test_sents_num):
    xun_lian = bidirectional_segment(test_sents[i], dic)
    xun_lian_list = zhuan_huan(xun_lian)
    ans_list = zhuan_huan(ans_sents[i])

    xun_lian_set = set()
    for tmp in xun_lian_list:
        xun_lian_set.add(tuple(tmp))
    ans_list_set = set()
    for tmp in ans_list:
        ans_list_set.add(tuple(tmp))
    TP = ans_list_set & xun_lian_set
    p = len(TP) / len(xun_lian_list)
    r = len(TP) / len(ans_list)
    P += p
    R += r
    if i % 100 == 0:
        print(i / test_sents_num)

# 求一个平均值
P = P / test_sents_num
R = R / test_sents_num
F_1 = 2 * P * R / (P + R)
print("P,R,F1", P, R, F_1)
end_time = time.time()
print(end_time - start_time)
```

附录二：基于统计的分词和词性标注

```
import csv
import time
```

```
start_time = time.time()
fenci_file = open("renmin.csv", "r")
fenci_reader = csv.reader(fenci_file)
fenci_sents = []
for item in fenci_reader:
    sent = []
    for i in item:
        sent.append(i.strip())
    fenci_sents.append(sent)

sum_zero = 0
for sent in fenci_sents:
    if len(sent) == 0:
        sum_zero += 1
    print(sent)
print(sum_zero)


# 一元语法模型
def sgram(sents: list):
    dic = {}
    dic['#始始#'] = 0
    dic['#末末#'] = 0
    for sent in sents:
        for item in sent:
            if item in dic:
                dic[item] = dic[item] + 1
            else:
                dic[item] = 1
        dic['#始始#'] += 1
        dic['#末末#'] += 1
    return dic


si_gram = sgram(fenci_sents)


# print(si_gram)


# 二元语法模型
def bgram(sents: list):
    dic = {}
    dic["#始始#"] = dict()
```

```
    for sent in sents:
        for i in range(0, len(sent) - 1):
            if sent[i] not in dic:
                dic[sent[i]] = dict()
                dic[sent[i]][sent[i + 1]] = 1
            else:
                if sent[i + 1] in dic[sent[i]]:
                    dic[sent[i]][sent[i + 1]] += 1
                else:
                    dic[sent[i]][sent[i + 1]] = 1

        if sent[0] not in dic["#始始#"]:
            dic["#始始#"][sent[0]] = 1
        else:
            dic["#始始#"][sent[0]] += 1

        if sent[len(sent) - 1] not in dic:
            dic[sent[len(sent) - 1]] = dict()
            dic[sent[len(sent) - 1]]["#末末#"] = 1
        elif "#末末#" not in dic[sent[len(sent) - 1]]:
            dic[sent[len(sent) - 1]]["#末末#"] = 1
        else:
            dic[sent[len(sent) - 1]]["#末末#"] += 1
    return dic


bi_gram = bgram(fenci_sents)


# print(bi_gram)


# 查看一元词频
def select_si_gram(gram: dict, word):
    return gram[word]


def selecr_bi_gram(gram: dict, word_one, word_two):
    return gram[word_one][word_two]


cixing_file = open("renmincixing.csv", "r")
cixing_reader = csv.reader(cixing_file)
cixing_sents = []
```

```
for item in cixing_reader:
    sent = []
    for i in item:
        sent.append(i.strip())
    cixing_sents.append(sent)

# print(sents)

part = dict()
# 统计所有词性个数
for sent in cixing_sents:
    for word in sent:
        word_part = word.split('/')[-1].split(']')[0].split('!')[0]
        if word_part in part:
            part[word_part] += 1
        else:
            part[word_part] = 1

part_len = len(part)
print(part, "一共多少类：", part_len)
print("总频次", sum(part.values()))

# 或得转移矩阵
trans = dict()
for sent in cixing_sents:
    for i in range(len(sent) - 1):
        one = sent[i].split('/')[-1].split(']')[0].split('!')[0]
        two = sent[i + 1].split('/')[-1].split(']')[0].split('!')[0]
        if one in trans:
            if two in trans[one]:
                trans[one][two] += 1
            else:
                trans[one][two] = 1
        else:
            trans[one] = dict()
            trans[one][two] = 1
print(trans)

# 每个词的词性概率
percent = dict()
for sent in cixing_sents:
    for word in sent:
        word_word = word.split('/')[0].split('{')[0].strip('[')
        word_part = word.split('/')[-1].split(']')[0].split('!')[0]
```

```
        if word_word in percent:
            if word_part in percent[word_word]:
                percent[word_word][word_part] += 1
            else:
                percent[word_word][word_part] = 1
        else:
            percent[word_word] = dict()
            percent[word_word][word_part] = 1


# print(percent)

def fen_ci(text):
    # 生成一元语法词网
    def generate_wordnet(gram, text):
        net = [[] for _ in range(len(text) + 2)]
        for i in range(len(text)):
            for j in range(i + 1, len(text) + 1):
                word = text[i:j]
                if word in gram:
                    net[i + 1].append(word)
        i = 1
        while i < len(net) - 1:
            if len(net[i]) == 0:  # 空白行
                j = i + 1

                for j in range(i + 1, len(net) - 1):
                    # 寻找第一个非空行j
                    if len(net[j]):
                        break
                # 填补i，j之间的空白行
                net[i].append(text[i - 1:j - 1])
                i = j
            else:
                i += len(net[i][-1])
        return net

    # 测试一个句子
    si_net = generate_wordnet(si_gram, text)

    # print(si_net)

    def calculate_gram_sum(gram: dict):
        num = 0
```

```
        for g in gram.keys():
            num += sum(gram[g].values())
        return num

    # 计算word_one后出现word_two的概率，带上+1平滑处理
    def calculate_weight(gram: dict, word_one, word_two, gram_sum):
        if word_one in gram:
            word_one_all = gram[word_one].values()
            if word_two in gram[word_one]:
                return (gram[word_one][word_two] + 1) / (sum(word_one_all) +
gram_sum)
            else:
                return 1 / (sum(word_one_all) + gram_sum)
        else:
            return 1 / gram_sum

    bi_gram_sum = calculate_gram_sum(bi_gram)

    # print(bi_gram_sum)

    def viterbi(wordnet):
        dis = [dict() for _ in range(len(wordnet))]
        node = [dict() for _ in range(len(wordnet))]
        word_line = [dict() for _ in range(len(wordnet))]
        wordnet[len(wordnet) - 1].append("#末末#")
        # 更新第一行
        for word in wordnet[1]:
            dis[1][word] = calculate_weight(bi_gram, "#始始#", word,
bi_gram_sum)
            node[1][word] = 0
            word_line[1][word] = "#始始#"
        # 遍历每一行wordnet
        for i in range(1, len(wordnet) - 1):
            # 遍历每一行中单词
            for word in wordnet[i]:
                # 更新加上这个单词的距离之后那个位置的所有单词的距离
                for to in wordnet[i + len(word)]:
                    if word in dis[i]:
                        if to in dis[i + len(word)]:
                            # 要的是最大的概率
                            if dis[i + len(word)][to] < dis[i][word] *
calculate_weight(bi_gram, word, to, bi_gram_sum):
                                dis[i + len(word)][to] = dis[i][word] *
calculate_weight(bi_gram, word, to, bi_gram_sum)
```

```
                            node[i + len(word)][to] = i
                            word_line[i + len(word)][to] = word
                    else:
                        dis[i + len(word)][to] = dis[i][word] *
calculate_weight(bi_gram, word, to, bi_gram_sum)
                        node[i + len(word)][to] = i
                        word_line[i + len(word)][to] = word

        # 回溯
        path = []
        f = node[len(node) - 1]["#末末#"]
        fword = word_line[len(word_line) - 1]["#末末#"]
        path.append(fword)
        while f:
            tmpword = fword
            fword = word_line[f][tmpword]
            f = node[f][tmpword]
            path.append(fword)
        path = path[:-1]
        path.reverse()
        return dis, node, word_line, path

    (dis, _, _, path) = viterbi(si_net)
    # print(dis)
    # print(path)
    return path


def ci_xing(text):
    text_percent = []

    # 这里我们假设是所有单词都已经人民日报语料库了
    for word in text:
        word_percent = percent[word]
        text_percent.append(word_percent)

    # print(text_percent)

    # 下面我们来使用Viterbi算法计算出最佳的组成
    dis = [dict() for _ in range(len(text))]
    node = [dict() for _ in range(len(text))]
    for first in text_percent[0].keys():
        dis[0][first] = 1
    for i in range(len(text) - 1):
```

```
        word_one = text[i]
        word_two = text[i + 1]
        word_one_percent_dict = text_percent[i]
        word_two_percent_dict = text_percent[i + 1]

        word_one_percent_key = list(word_one_percent_dict.keys())
        word_one_percent_value = list(word_one_percent_dict.values())
        word_two_percent_key = list(word_two_percent_dict.keys())
        word_two_percent_value = list(word_two_percent_dict.values())
        for word_two_per in word_two_percent_key:
            tmp_dis = []
            for word_one_per in word_one_percent_key:
                if word_two_per in trans[word_one_per]:
                    tmp_num = dis[i][word_one_per] * (
                            (trans[word_one_per][word_two_per] + 1) /
(part[word_one_per] + part_len)) * (
                                    text_percent[i + 1][word_two_per] /
part[word_two_per])
                    tmp_dis.append(tmp_num)
                else:
                    tmp_num = dis[i][word_one_per] * (1 / (part[word_one_per] +
part_len)) * (
                            text_percent[i + 1][word_two_per] /
part[word_two_per])
                    tmp_dis.append(tmp_num)

            max_tmp_dis = max(tmp_dis)
            max_tmp_dis_loc = tmp_dis.index(max_tmp_dis)
            dis[i + 1][word_two_per] = max_tmp_dis
            node[i + 1][word_two_per] = word_one_percent_key[max_tmp_dis_loc]
    # print(dis, node)

    # 根据node来倒找答案
    path = []
    f_value = list(dis[len(dis) - 1].values())
    f_key = list(dis[len(dis) - 1].keys())
    f = f_key[f_value.index(max(f_value))]

    path.append(f)
    for i in range(len(dis) - 1, 0, -1):
        f = node[i][f]
        path.append(f)
    path.reverse()
    return path
```

```
# 对所有训练集进行测试
test_file = open("renmincixing.csv", "r")
reader = csv.reader(test_file)
test_sents = []
ans_sents = []
fenci_sents = []
for item in reader:
    test_sent = ""
    ans_sent = []
    fenci_sent = []
    for word in item:
        word_word = word.split('/')[0].split('{')[0].strip('[')
        word_part = word.split('/')[-1].split(']')[0].split('!')[0]
        if word_word == '。' and word_word == '！' and word_word == '？':
            test_sent += word_word.strip()
            ans_sent.append(word_word)
            fenci_sent.append(word_part)
            break
        else:
            test_sent += word_word.strip()
            ans_sent.append(word_word)
            fenci_sent.append(word_part)
    test_sents.append(test_sent)
    ans_sents.append(ans_sent)
    fenci_sents.append(fenci_sent)


# 将分词出来的结果转换为集合中元素
def zhuan_huan(text: list):
    ans = []
    i = 1
    for word in text:
        ans.append([i, i + len(word) - 1])
        i += len(word)
    return ans


test_sents_num = len(test_sents)
print(test_sents_num)
P = 0
R = 0
A = 0
```

```
A_num = 0
for i in range(test_sents_num):
    xun_lian = fen_ci(test_sents[i])
    xun_lian_list = zhuan_huan(xun_lian)
    ans_list = zhuan_huan(ans_sents[i])

    xun_lian_set = set()
    for tmp in xun_lian_list:
        xun_lian_set.add(tuple(tmp))

    ans_list_set = set()
    for tmp in ans_list:
        ans_list_set.add(tuple(tmp))

    TP = ans_list_set & xun_lian_set
    p = len(TP) / len(xun_lian_list)
    r = len(TP) / len(ans_list)
    # 我们只对分词正确的结果进行词性正确性评估
    if ans_list_set == xun_lian_set:
        A_num += 1
        # 预测来的
        ci_xing_list = ci_xing(ans_sents[i])
        # 正确答案
        ci_xing_ans = fenci_sents[i]
        a = 0
        for j in range(len(ci_xing_list)):
            if ci_xing_list[j] == ci_xing_ans[j]:
                a += 1

        a = a / len(ci_xing_list)
        A += a
    P += p
    R += r
    if i % 100 == 0:
        print(i / test_sents_num)

# 求一个平均值
P = P / test_sents_num
R = R / test_sents_num
F_1 = 2 * P * R / (P + R)
print("P,R,F1", P, R, F_1)
A = A / A_num
print("A", A)
end_time = time.time()
```

```
print(end_time - start_time)
```

附录三：处理人民日报分词数据：

```
import csv

mat = []

with open("renmin.txt", "r") as f:  # 打开文件
    for line in f:
        mat.append([l for l in line.split()])

print(mat)

with open('renmin.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in mat:
        if len(row) != 0:
            writer.writerow(row)
```

附录四：处理人民日报词性数据：

```
import csv

mat = []

with open("renmincixing.txt", "r") as f:  # 打开文件
    for line in f:
        line = line[22:]
        print(line)
        mat.append([l for l in line.split()])

print(mat)

with open('renmincixing.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in mat:
        if len(row) != 0:
            writer.writerow(row)
```