دانشگاه اصفهان
دانشکده مهندسی کامپیوتر
گروه هوش مصنوعی

تمرین دوم پردازش زبان های طبیعی

مهرداد قصابی

استاد راهنما

دکتر برادران

اردیبهشت ۱۴۰۳

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست الگوریتم‌ها

فهرست برنامه‌ها

فصل ۱

پیش پردازش

۱.۱ پرسش نخست

پرسش: هر آگاهی را به جملات زیر تجزیه کنید.

پاسخ:

با استفاده از توکنایزر جمله کتابخانه هضم طبق عکس زیر ۱.۱ هر آگهی به جملات تشکیل دهنده آن تجزیه شد.

```
[4]  def get_sentences(text):
         tokenizer = hazm.SentenceTokenizer()
         sentences = tokenizer.tokenize(text)
         return sentences

     comments_sentences = []
     for comment in comments:
         sentences = get_sentences(comment)
         comments_sentences.append(sentences)
     print(comments_sentences[17])
```

```
['تلوزیون رنگی ۲۹ اینچ توشیبا صفحه تخت کاملا سالم همراه با میز فابریک']
```

شکل ۱.۱: تجزیه هر کامنت به جملات تشکیل دهنده آن ۱.۱

پرسش: برای هر جمله علائم نگارشی، فضاهای خالی اضافه و... را حذف کنید به طوری که در انتها فقط اعداد و کلمات را داشته باشید.

پاسخ:

طبق برنامه زیر علائم نگارشی مورد نظر حذف شد.

```
[5]  def remove_punctuation(text):
         persian_punctuation = 'ء\n()/•٪,،’«»؟:.!-…*_×''+$'
         cleaned_text = ""
         for char in text:
             if char in persian_punctuation:
                cleaned_text = cleaned_text + '\u200C'
             else:
                cleaned_text = cleaned_text + char

         return cleaned_text
```

شکل ۲.۱: حذف علائم نگارشی از متن ۲.۱

پرسش: هر گونه پیش پردازش دیگری که بتواند نتایج را بهبود دهد و مطابق با مسائل مدلسازی زبانی باشد انجام داده و توضیح دهید که چرا این مراحل پیش پردازش انتخاب شده است.

پاسخ:

برای بهبود مدل زبانی دو پیش پردازش دیگر در نظر گرفته شده است که برنامه آن در عکس زیر آمده است.

```
[ ]  def normalize_comments(comments):
         normalizer = hazm.Normalizer()
         normalized_comments = []
         for comment in comments:
             comment = normalizer.normalize(comment)
             comment = remove_punctuation(comment)
             comment = comment.replace('\u200C', " ")
             comment = transliterate(comment)
             comment = replace_persian_numbers(comment)
             comment = comment.replace("$Num", "")
             comment = comment.replace("  ", " ")
             normalized_comments.append(comment)
         return normalized_comments

[ ]  normalized_comments = normalize_comments(comments)
     print(normalized_comments[0])

     کلاسیک و شیک و استثنایی چرم مالزی چوب راش فوق العاده سالم و بدون عیب و ایراد
```

شکل ۳.۱: پیش پردازش ۳.۱

- حذف اعداد: برای کاهش بعد و تعداد توکن های منحصر به فرد بهتر است تمامی اعداد را به یک توکن تبدیل کرده یا به طور کلی آنها را حذف کنیم.
- فارسی سازی حروف الفبا [۱] : ممکن است در بین نظرات فارسی برخی از واژگان مانند اسامی شرکت ها

[۱] transliterate

با حروف الفبای لاتین نوشته شده باشد تبدیل آنها علاوه بر یکدست شدن متن میتواند باعث کاهش بعد نیز بشود.

```python
[ ] def transliterate(text):
        persian_alphabet = {
            'a': 'ا',
            'b': 'ب',
            'c': 'س',
            'd': 'د',
            'e': '',
            'f': 'ف',
            'g': 'گ',
            'h': 'ه',
            'i': 'ی',
            'j': 'ج',
            'k': 'ک',
            'l': 'ل',
            'm': 'م',
            'n': 'ن',
            'o': '',
            'p': 'پ',
            'q': 'ک',
            'r': 'ر',
            's': 'س',
            't': 'ت',
            'u': 'و',
            'v': 'و',
            'w': 'و',
            'x': 'کس',
            'y': 'ی',
            'z': 'ز'
        }

        result = ''
        for char in text:
            if char.lower() in persian_alphabet:
                result += persian_alphabet[char.lower()]
            else:
                result += char

        return result
```

شکل ۴.۱: فارسی سازی حروف الفبا ۴.۱

فصل ۲

ساخت مدل زبانی

۱.۲ پرسش نخست

پرسش: برای داده های پیش پردازش شده مراحل زیر را انجام دهید.

- یک مدل زبانی n-gram را پیاده کنید که به n اجازه میدهد از یک تا سه تغییر کند. سپس هشت عدد از پر تکرار ترین unigram ، bigram و trigram ها را نمایش دهید.

  پاسخ:

  طبق برنامه موجود در عکس ۱.۲ مدل n-gram ساخته شد؛ پر تکرارترین آنها نیز در عکس ۲.۲ آمده است.

- توضیح دهید دلیل هموارسازی در محاسبه احتمالات n-gram ها چیست و سپس Laplace smoothing و Good Turing smoothing را نیز توضیح دهید.

  پاسخ:

  از آنجایی که مجموعه داده ما محدود است ممکن است برخی از ترکیب های n تایی معنا دار از واژگان در متن مجموعه داده ما وجود نداشته باشد و به همین دلیل مدل زبانی ما احتمال وقوع آنها را صفر پیش بینی کند؛ در صورتی که مطلوب این است که مدل مقدار نزدیک به صفر برای آنها پیش بینی شود چون به هر حال احتمال وقوع یک دنباله معنا دار همواره وجود دارد.

برای هموار سازی شیوه های متفاوتی وجود دارد که یکی از آنها روش لاپلاس یا add-k می باشد در این روش ما وانمود میکنیم که همه n-gram ها را یک بار بیشتر دیده ایم.

بنابراین احتمال رخداد یک ترکیب n تایی برابر است با

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

روش دیگر هموار سازی روش Good Turing است در این روش تعداد n-gram هایی که تنها یک مرتبه تکرار شده اند [1] شمرده شده و بر اساس آن یک احتمال به توکن های دیده نشده نسبت داده میشود.

$$P^*(unseen) = \frac{N_{seen-once}}{N_{total}}$$

احتمال رخ دادن n-gram های دیده شده نیز با استفاده از فرمول زیر بدست می آید.

$$P^*(seen - Ctime) = \frac{C^*}{N_{total}}, C^* = \frac{(C+1) \times N_{C+1}}{N_C}$$

- پرسش: تابعی برای محاسبه n-gram ها بنویسید.

پاسخ: مطابق عکس ۳.۲ و طبق فرمول هموار سازی لاپلاس و تورینگ که بالاتر از آنها یاد شده است انجام شد.

- پرسش: تابعی برای محاسبه perplexity بنویسید و جدول زیر را تکمیل کنید.

پاسخ: مطابق عکس ۳.۲ این تابع نوشته شده و جدول ۵.۲ نیز تکمیل گشت.

۲.۲ پرسش دوم

- پرسش: با استفاده از مدل ساخته شده جملات ناقص جدول زیر را تکمیل کنید.

پاسخ: جدول تکمیل شد ۶.۲ مدل بایگرم جملات با مفهوم تری نسبت به دو مدل دیگر تولید میکند.

[1] یک مرتبه منظور فرکانس رخداد کم است، کم بودن فرکانس رخداد میتواند به شیوه های مختلفی تعریف گردد.

```
[ ]  def build_ngram_model(comments, n):
         ngrams = defaultdict(int)
         total_ngrams = 0

         for comment in comments:
             tokenizer = hazm.WordTokenizer()
             words = tokenizer.tokenize(comment)

             for i in range(len(words) - n + 1):
                 ngram = tuple(words[i:i+n])
                 ngrams[ngram] += 1
                 total_ngrams += 1

         ngram_model = {}
         for ngram, count in ngrams.items():
             ngram_model[ngram] = count

         return ngram_model,total_ngrams
```

شکل ۱.۲: برنامه ساختن مدل n-gram ۱.۲

- پرسش: perplexity جملات ساخته شده را بیابید.

  پاسخ: طبق عکس ۷.۲ محاسبه شد.

- پرسش: توضیح دهید در استفاده از مدل زبانی ngram چه عواملی در انتخاب n موثر است.

  پاسخ:

  اندازه مجموعه داده در انتخاب n بسیار موثر است برای مجموعه داده های کوچک مانند مجموعه داده این تمرین n=2 مناسب است ولی برای مجموعه داده های بزرگتر میتوان از n=3 استفاده کرد؛ در واقع اندازه مجموعه داده ها با n رابطه توانی دارد.

  البته عوامل دیگری مانند sparse بودن مجموعه داده ها و پیچیدگی زبان نیز در انخاب n موثر هستند.

```
[ ]  unigram_model,total_unigrams = build_ngram_model(comments = normalized_comments, n = 1)
     unigram_model = dict(sorted(unigram_model.items(), key = lambda x:-x[1]))
     bigram_model,total_bigrams = build_ngram_model(comments = normalized_comments, n = 2)
     bigram_model = dict(sorted(bigram_model.items(), key = lambda x:-x[1]))
     trigram_model,total_trigrams = build_ngram_model(comments = normalized_comments, n = 3)
     trigram_model = dict(sorted(trigram_model.items(), key = lambda x:-x[1]))
```

```
[ ]  print_top_ngrams(unigram_model,1)
     print("============================================================")
     print_top_ngrams(bigram_model,2)
     print("============================================================")
     print_top_ngrams(trigram_model,3)
```

```
118  و
66  با
60  سالم
50  نو
42  به
41  تمیز
34  بدون
34  در
============================================================
17  در حد
13  بسیار تمیز
13  کاملا سالم
11  سالم و
10  یک عدد
10  می باشد
10  تمیز و
9  همراه با
============================================================
8  در حد نو
5  داده می شود
4  کاملا سالم و
4  سالم و تمیز
4  به خریدار واقعی
3  سالم و بدون
3  بسیار تمیز و
3  میل هفت نفره
```

شکل ۲.۲: پر تکرار ترین n-gram ها در مجموعه دادگان ۲.۲

```
[ ]  def ngram_probablity(ngram,word,
                          unigram_model,total_unigrams,bigram_model,trigram_model,
                          n):
         assert n>=1 and n<=3
         prob = 0
         if n == 1:
            V = len(unigram_model.keys())
            K = 0.005
            prob = (get_ngram_count(unigram_model,(word,)) + K )/(total_unigrams + K*V)
         if n == 2:
            V = len(unigram_model.keys())
            K = 0.005
            prob = (get_ngram_count(bigram_model,ngram+(word,)) + K )/(get_ngram_count(unigram_model,ngram) + K*V)
         if n == 3:
            dic = count_dic(trigram_model)
            c = get_ngram_count(trigram_model,ngram+(word,))
            c_star = 0
            if c == 0:
               c_star = N(dic,c)
            else:
               c_star = ((c+1)*N(dic,c+1))/(N(dic,c))
            bicount = get_ngram_count(bigram_model,ngram)
            if bicount == 0:
               prob = 0.05
            else:
               prob = c_star / bicount

         return prob
```

شکل ۳.۲: تابع احتمال رخداد یک n-gram ۳.۲

```
[ ]  def perplexity(sentence,n):
        assert n>=1 and n<=3
        tokenizer = hazm.WordTokenizer()
        words = tokenizer.tokenize(sentence)
        num_words = len(words)
        log_prob = 0.0
        if n == 1 :
           for i in range(0, num_words):
               word = words[i]
               ngram = None
               prob = ngram_probablity(ngram,word,
                         unigram_model_1,total_unigrams_1,None,None,
                         1)
               log_prob += math.log(prob)
        elif n == 2:
           n = n - 1
           for i in range(n-1, num_words - 1):
              ng = tuple(words[i-n+1:i+2])
              ngram = ng[0:n]
              word = ng[-1]
              prob = ngram_probablity(ngram,word,
                         unigram_model_2,total_unigrams_2,bigram_model_2,None,
                         2)
              # print(prob)
              log_prob += math.log(prob)
        elif n == 3:
           n = n - 1
           for i in range(n-1, num_words - 1):
              ng = tuple(words[i-n+1:i+2])
              ngram = ng[0:n]
              word = ng[-1]
              prob = ngram_probablity(ngram,word,
                         None,None,bigram_model_3,trigram_model_3,
                         3)
              # print(prob)
              log_prob += math.log(prob)

        perplexity = math.exp(-log_prob / num_words)
        return perplexity
```

شکل ۴.۲: تابع perplexity ۴.۲

#	عبارت	Unigram Perplexity	Bigram Perplexity	Trigram Perplexity
1	گوشی بسیار بسیار تمیز و فقط سه هفته کار کرده در حد آک	237.20	20.56	49.40
2	دو عدد پیراهن دخترونه مارک در حد نو مناسب تا یک سال و نیم	226.18	76.25	85.50
3	کفش طبی چرم مصنوعی مارک لمون طب که نو میباشد	1167.34	26.07	21.46
4	دوربین عکاسی خانگی کنان سالم در حد نو	322.87	15.05	21.72
5	گل مصنوعی کاملا سالم بدون ایراد همراه با گلدان	223.62	11.15	32.62

شکل ۵.۲: جدول perplexity ۵.۲

#	عبارت	Predicted sentence by Unigram model	Predicted sentence by Bigram model	Predicted sentence by Trigram model
1	مبل هفت نفره خود رنگ	مبل هفت نفره خود رنگ تهران هزار شرط خریدار قهوه	مبل هفت نفره خود رنگ دارد متوری تمیز وکم کار	مبل هفت نفره خود رنگ درحد درضمن دلیل ساپورت تشک
2	دستگاه تردمیل نو	دستگاه تردمیل نو تماس رو خط ویژگی نفره فقط استفاده	دستگاه تردمیل نو هستند اگر خوشتون آمد زنگ بزنید تنوع	دستگاه تردمیل نو این نرم نو فاق شده سالم کلش
3	کفش مردانه	کفش مردانه اصلی نو ایکس تیبا مونده مکانیکی ومدارک خرجی	کفش مردانه سایز نومویک عدد کیف هم داره سوالی دیگه	کفش مردانه آمریکا العاده ببین واسه طبقات همه روآ واستفاده
4	تعدادی وسایل اداری و پزشکی	تعدادی وسایل اداری و پزشکی ه چوبی هر حمل اگر	تعدادی وسایل اداری و پزشکی استفاده نمی شود تماس تلفنی	تعدادی وسایل اداری و پزشکی پژو بیمه مناسب با باشد

شکل ۶.۲: جدول جملات تولید شده ۶.۲

```
[ ]  sentence1 = 'مبل هفت نفره خود رنگ تهران هزار شرط خریدار قهوه پ'
     sentence2 = 'ش مبل هفت نفره خود رنگ دارد متوری تمیز وکم کار پ'
     sentence3 = 'ش ش مبل هفت نفره خود رنگ درحد درضمن دلیل سا پورت تشک پ'
     print('unigram perplexity: ' + str(perplexity(sentence1,1)))
     print('bigram perplexity: ' + str(perplexity(sentence2,2)))
     print('trigram perplexity: ' + str(perplexity(sentence3,3)))

     unigram perplexity: 453.3268535315307
     bigram perplexity: 15.260468565132317
     trigram perplexity: 15.608236005912978

[ ]  sentence1 = 'دستگاه تردمیل نو تماس رو خط ویژگی نفره فقط استفاده پ'
     sentence2 = 'ش دستگاه تردمیل نو هستند اگر خوشتون آمد زنگ بزنید تنوع پ'
     sentence3 = 'ش ش دستگاه تردمیل نو هستند اگر خوشتون آمد زنگ بزنید تنوع پ'
     print('unigram perplexity: ' + str(perplexity(sentence1,1)))
     print('bigram perplexity: ' + str(perplexity(sentence2,2)))
     print('trigram perplexity: ' + str(perplexity(sentence3,3)))

     unigram perplexity: 353.8009191059761
     bigram perplexity: 23.28045410542312
     trigram perplexity: 49.577941952160295

[ ]  sentence1 = 'کفش مردانه اصلی نو ایکس تیبا مونده مکانیکی ومدارک خرجی پ'
     sentence2 = 'ش کفش مردانه سایز نومویک عدد کیف هم داره سوالی دیگه پ'
     sentence3 = 'ش ش کفش مردانه آمریکا العاده ببین واسه طبقات همه روآ واستفاده پ'
     print('unigram perplexity: ' + str(perplexity(sentence1,1)))
     print('bigram perplexity: ' + str(perplexity(sentence2,2)))
     print('trigram perplexity: ' + str(perplexity(sentence3,3)))

     unigram perplexity: 890.4492006387612
     bigram perplexity: 12.560729481747448
     trigram perplexity: 21.68243562161054

[ ]  sentence1 = 'تعدادی وسایل اداری و پزشکی ه چوبی هر حمل اگر پ'
     sentence2 = 'ش تعدادی وسایل اداری و پزشکی استفاده نمی شود تماس تلفنی پ'
     sentence3 = 'ش ش تعدادی وسایل اداری و پزشکی پژو بیمه مناسب با باش پ'
     print('unigram perplexity: ' + str(perplexity(sentence1,1)))
     print('bigram perplexity: ' + str(perplexity(sentence2,2)))
     print('trigram perplexity: ' + str(perplexity(sentence3,3)))

     unigram perplexity: 831.5953654398569
     bigram perplexity: 17.737607077593907
     trigram perplexity: 21.95058428706163
```

شکل ۷.۲: perplexity جدول جملات تولید شده ۷.۲

فصل ۳

برچسب گذاری کلمات

۱.۳ قسمت اول برنامه نویسی

- پرسش: با استفاده از کتابخانه هضم عملیات POSTagging را روی مجموعه داده پیش پردازش شده اعمال کنید و تگ هر توکن را در خروجی نشان دهید.

  پاسخ:

  ابتدا مدل از پیش تمرین شده هضم را دانلود کرده و مطابق برنامه ۳.۳ برای همه کامنت ها برچسب گذاری را انجام میدهیم.

- پرسش: تعداد رخداد هر تگ pos را بیابید وگزارش دهید.

  پاسخ:

  مطابق برنامه ۳.۳ تعداد رخداد هر تگ یافته شده است.

- پرسش: اسم ها را جدا کرده و پانزده اسم پرتکرار را نمایش دهید

  پاسخ:

  مطابق برنامه ۳.۳ انجام شده است.

شکل ۱.۳: تگ همه نظرات ۱.۳

۲.۳ قسمت دوم رویکرد عملی

- الف: جملات داده شده دارای POSTagging زیر هستند:

Mark can watch : Noun Modal Verb

Will can mark watch : Noun Modal Verb Noun

can Tom watch : Modal Noun Verb

Tom will mark watch : Noun Modal Verb Noun

بر اساس شمارش، جدول احتمال هر کلمه بر اساس سه برچسب در این عکس ۴.۳ آمده است.

- ب:طبق عکس ۵.۳ انجام شد.
- پ: طبق عکس ۶.۳ انجام شد.

```
[ ]  tokenizer = hazm.WordTokenizer()
     valid_tags = ['NOUN','NOUN,EZ','VERB',
                   'CCONJ','ADV','ADJ','ADP',
                   'PRON','NUM','ADJ,EZ','ADP,EZ',
                   'SCONJ','DET','INTJ','DET,EZ','ADV,EZ','NUM,EZ']
     count = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
     for cmnt in normalized_comments:
         words = tokenizer.tokenize(cmnt)
         for tagged_sen in postagger.tag(words):
             count[get_occuring_index(valid_tags,tagged_sen[1])] += 1
```

```
[ ]  for i in range(17):
         print(str(valid_tags[i]) + ': ' + str(count[i]))
```

```
NOUN: 978
NOUN,EZ: 845
VERB: 241
CCONJ: 155
ADV: 134
ADJ: 320
ADP: 210
PRON: 32
NUM: 91
ADJ,EZ: 184
ADP,EZ: 79
SCONJ: 14
DET: 29
INTJ: 7
DET,EZ: 15
ADV,EZ: 1
NUM,EZ: 1
```

شکل ۲.۳: تعداد رخداد هر تگ ۲.۳

```
[ ]  Nouns_tag = ['NOUN','NOUN,EZ']
     repeated_words = {}
     for cmnt in normalized_comments:
         words = tokenizer.tokenize(cmnt)
         for tagged_sen in postagger.tag(words):
             if tagged_sen[1] == 'NOUN' or tagged_sen[1] == 'NOUN,EZ':
                if tagged_sen[0] not in repeated_words.keys():
                   repeated_words[tagged_sen[0]] = 1
                else:
                   repeated_words[tagged_sen[0]] += 1
     repeated_words = dict(sorted(repeated_words.items(), key = lambda x:-x[1]))

     i = 0
     for word, count in repeated_words.items():
         i += 1
         print('word: '+ str(word) + ' count: ' + str(count))
         if i == 10:
            break
```

```
word: می count: 33
word: رنگ count: 32
word: عدد count: 26
word: تخفیف count: 24
word: بیمه count: 24
word: قیمت count: 23
word: فروش count: 23
word: نفره count: 19
word: استفاده count: 19
word: لاستیک count: 18
```

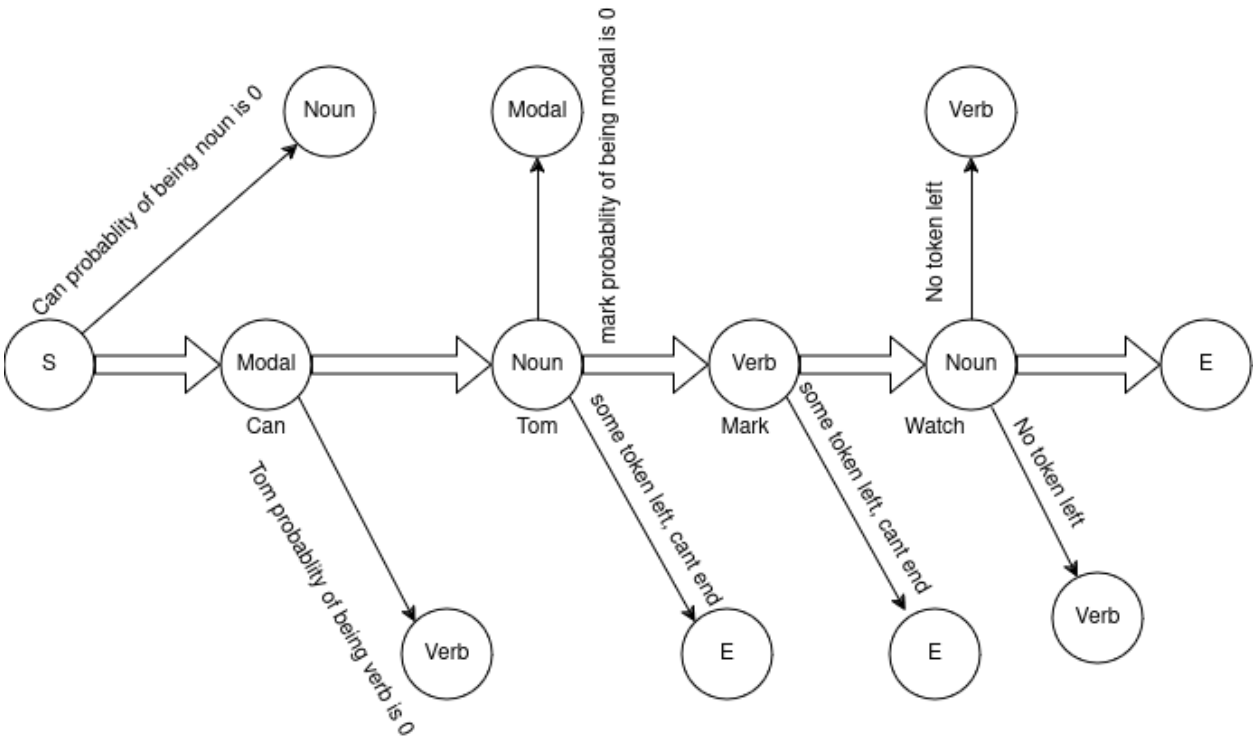
شکل ۳.۳: پانزده اسم پرتکرار ۳.۳

کلمات	Noun	modal	verb	#
Tom	2/6	0	0	1
Mark	1/6	0	2/5	2
Can	0	1/2	1/5	3
Watch	2/6	0	2/5	4
Will	1/6	1/2	0	5

شکل ۴.۳: احتمال هر کلمه بر اساس سه برچسب ۴.۳

کلمات	Noun	modal	verb	E
S	3/4	1/4	0	0
Noun	0	3/6	1/6	2/6
Modal	1/4	0	3/4	0
Verb	2/4	0	0	2/4

شکل ۵.۳: احتمال وقوع هم زمان دو بر چسب ۵.۳



شکل ۶.۳: گراف POSTagging ۶.۳