به نام خدا

گزارش تمرین اول درس پردازش زبان طبیعی

استاد درس:

جناب دکتر برادران

نام و نام خانوادگی دانشجو:

امیررضا صدیقین

شماره دانشجویی:

۹۹۳۶۱۴۰۲۴

مرحله ۱:

در این بخش متغیرهای متنی مورد استفاده در این تمرین تعریف شده اند.

مرحله ۲:

بخش a:

Chunking یک پروسه‌ی استخراج عبارات از یک متن ساختار نیافته است. و به هر عبارت یک chunk گویند. فرق chunk با توکن در آن است که chunk یه عبارت معنی دار را جدا می‌کند در حالی که token صرفا بر اساس یک سری قوانین این کار را انجام میدهد. برای مثال South Africa در پروسه‌ی chunking یک عبارت در نظر گرفته می‌شود به جای این که دو توکن South و Africa شود. همچنین noun-phrase chunking پروسه‌ی استخراج عبارات اسمی است.

بخش b:

در این بخش متن اول توکنایز شده و سپس عملیات POS tagging روی آن اعمال شده است.

```
In [4]: tokens = nltk.word_tokenize(text_a)

In [6]: nltk.download('averaged_perceptron_tagger')

        [nltk_data] Downloading package averaged_perceptron_tagger to
        [nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
        [nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.

Out[6]: True

In [7]: nltk.pos_tag(tokens)

Out[7]: [('Natural', 'JJ'),
         ('language', 'NN'),
         ('processing', 'NN'),
         ('is', 'VBZ'),
         ('fun', 'RB'),
         ('!', '.'),
         ('This', 'DT'),
         ('text', 'NN'),
         ('is', 'VBZ'),
         ('a', 'DT'),
         ('sample', 'JJ'),
         ('text', 'NN'),
         ('.', '.')]
```

بخش c:

می‌توان با استفاده از کلاس RegexpParser عملیات chunking را با استفاده از یک عبارت منظم روی یک متن انجام داد. (در ورودی کلاس RegexpParser یک گرامر گرفته می‌شود که میتوان عبارت منظم را در قالب گرامر به این کلاس داد.). همچنین با استفاده از تابع parse در این کلاس میتوان درخت آن را به دست آورد.

بخش d:

در این بخش گرامر گفته شده تعریف شد و parserی با استفاده از RegexpParser و گرامر گفته شده ساخته شد.

```
grammar = ('''
    NP: {<DT>?<JJ>*<NN>} # NP
    ''')
```

```
parser = nltk.RegexpParser(grammar)
```

```
tree = parser.parse(tags)
```

```
for subtree in tree.subtrees():
    print(subtree)
```

```
(S
  (NP Natural/JJ language/NN)
  (NP processing/NN)
  is/VBZ
  fun/RB
  !/.
  (NP This/DT text/NN)
  is/VBZ
  (NP a/DT sample/JJ text/NN)
  ./.)
(NP Natural/JJ language/NN)
(NP processing/NN)
(NP This/DT text/NN)
(NP a/DT sample/JJ text/NN)
```

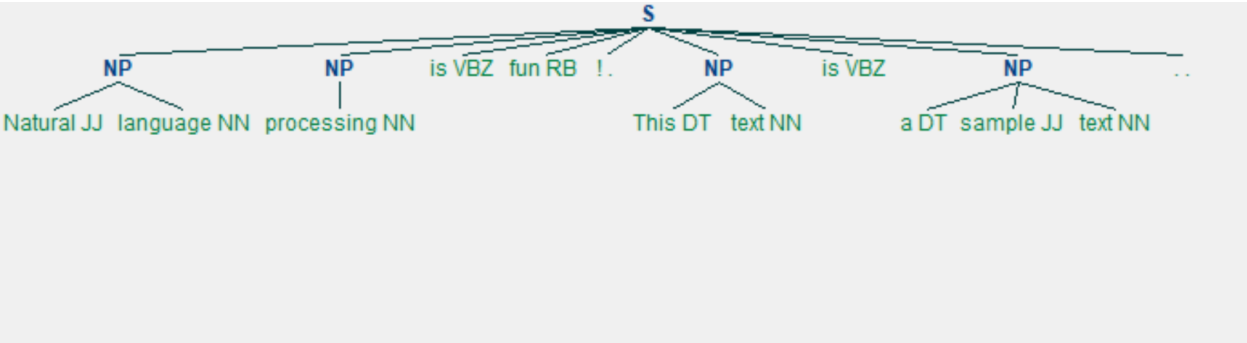
بخش e:

با توجه به گرامر گفته شده، منظور آن است که اسم‌های مفرد را به عنوان یک Noun Phrase در نظر بگیر و همچنین اگر قبل از آن‌ها نیز صفت‌هایی آمده باشد، آن صفت‌ها به همراه اسم را نیز به عنوان Noun Phrase در نظر می‌گیریم و اگر قبل از این موارد کلمه‌ی Determinerی وجود داشت نیز آن هم به عنوان عبارت اسمی در نظر میگیریم. (کلمات Determiner مثل the یا a و ... است)

بخش f:

با استفاده از متد draw روی درخت ایجاد شده، درخت زیر بدست می‌آید.

درخت آن به صورت زیر است.



بخش g:

در این بخش دو گرامر و دو متن دیگر مثال زده شده است.

مثال اول:

- متن نمونه : hello! My name is Amirreza Seddighin
- پترن : دنباله‌ی اسامی خاص پشت سرهم.

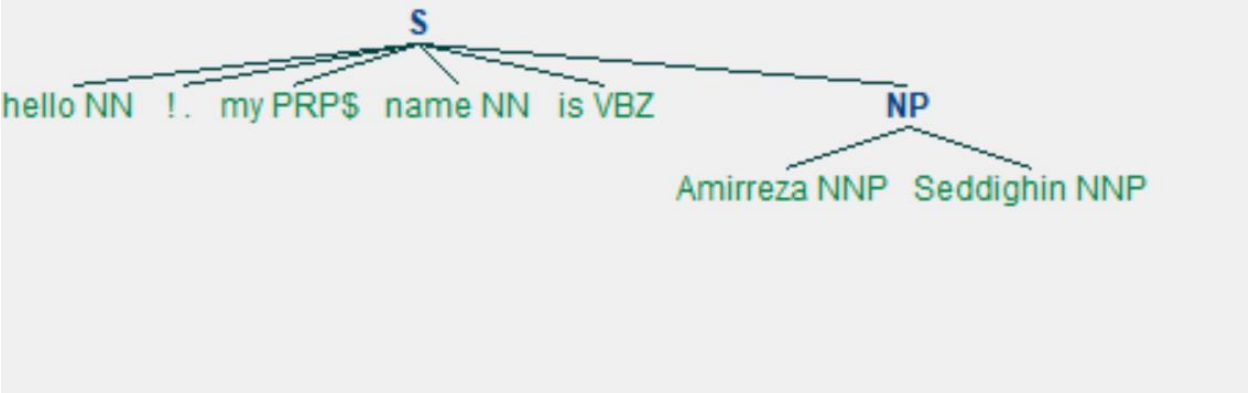
<NNP>*

تکه کد آن نیز به صورت زیر است.

sample 1 :

```
In [22]: text = "hello! my name is Amirreza Seddighin"

In [23]: tags = nltk.pos_tag(nltk.word_tokenize(text))

In [24]: tags
Out[24]: [('hello', 'NN'),
          ('!', '.'),
          ('my', 'PRP$'),
          ('name', 'NN'),
          ('is', 'VBZ'),
          ('Amirreza', 'NNP'),
          ('Seddighin', 'NNP')]

In [25]: grammar = ('''
             NP: {<NNP>*} # NP
             ''')

In [26]: parser = nltk.RegexpParser(grammar)

In [28]: tree = parser.parse(tags)

In [29]: for subtree in tree.subtrees():
             print(subtree)

(S
  hello/NN
  !/.
  my/PRP$
  name/NN
  is/VBZ
  (NP Amirreza/NNP Seddighin/NNP))
(NP Amirreza/NNP Seddighin/NNP)

In [*]: tree.draw()
```

درخت آن نیز به صورت زیر است.



مثال دوم:

- متن نمونه : My teacher is good
- پترن : ضمیر به علاوه‌ی یک اسم مفرد

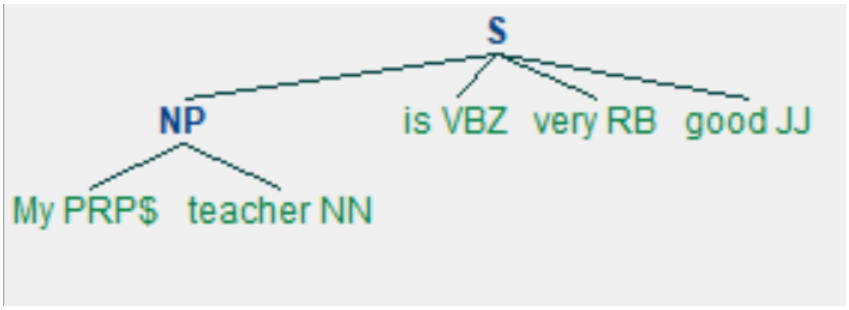
<PRP\$> <NN>

تکه کد آن نیز به صورت زیر است.

example 2 :

```
In [121]: text = "My teacher is very good"

In [122]: tags = nltk.pos_tag(nltk.word_tokenize(text))

In [123]: tags

Out[123]: [('My', 'PRP$'),
           ('teacher', 'NN'),
           ('is', 'VBZ'),
           ('very', 'RB'),
           ('good', 'JJ')]

In [130]: grammar = ('''
              NP: {<PRP\$> <NN>} # NP
              ''')

In [131]: parser = nltk.RegexpParser(grammar)

In [132]: tree = parser.parse(tags)

In [133]: for subtree in tree.subtrees():
              print(subtree)

          (S (NP My/PRP$ teacher/NN) is/VBZ very/RB good/JJ)
          (NP My/PRP$ teacher/NN)

In [134]: tree.draw()
```

درخت آن نیز به صورت زیر است.



مرحله‌ی ۳:

بخش a:

برای IOB encoding کافی است از تابع tree2conlltags استفاده کنیم و در ورودی آن درخت تجزیه‌ی خود را بدهیم.

بخش b:

در این بخش عملیات IOB encoding بر روی متن دوم اعمال شده است که به صورت زیر است.(همچنین بخش‌هایی از جواب‌ها نیز به نمایش در آمده است.)

```
In [172]: tags = nltk.pos_tag(nltk.word_tokenize(text_b))

In [173]: tags

Out[173]: [('Jensen', 'NNP'),
           ('Huang', 'NNP'),
           (',', ','),
           ('the', 'DT'),
           ('CEO', 'NNP'),
           ('of', 'IN'),
           ('Nvidia', 'NNP'),
           (',', ','),
           ('the', 'DT'),
           ('nation', 'NN'),
           ("'", 'NNP'),
           ('s', 'VBZ'),
           ('most', 'RBS'),
           ('valuable', 'JJ'),
           ('semiconductor', 'NN'),
           ('company', 'NN'),
           (',', ','),
           ('with', 'IN'),
           ('a', 'DT'),
           ('stock', 'NN')

In [174]: grammar = ('''
              NP: {<DT>?<JJ>*<NN>} # NP
              ''')

In [175]: parser = nltk.RegexpParser(grammar)

In [176]: tree = parser.parse(tags)

In [181]: tree2conlltags(tree)

Out[181]: [('Jensen', 'NNP', 'O'),
           ('Huang', 'NNP', 'O'),
           (',', ',', 'O'),
           ('the', 'DT', 'O'),
           ('CEO', 'NNP', 'O'),
           ('of', 'IN', 'O'),
           ('Nvidia', 'NNP', 'O'),
           (',', ',', 'O'),
           ('the', 'DT', 'B-NP'),
           ('nation', 'NN', 'I-NP'),
           ("'", 'NNP', 'O'),
           ('s', 'VBZ', 'O'),
           ('most', 'RBS', 'O'),
           ('valuable', 'JJ', 'B-NP'),
           ('semiconductor', 'NN', 'I-NP'),
           ('company', 'NN', 'B-NP'),
           (',', ',', 'O'),
           ('with', 'IN', 'O'),
           ('a', 'DT', 'B-NP')
```

بخش c:

در این بخش با توجه به گرامر گفته شده، عبارات به سه بخش O و B_NP و I_NP است. که منظور از O یعنی آن که خارج از محدوده است و NP نیست. منظور از B_NP یعنی Begin NP به معنای توکنی است که نشان دهنده‌ی اول عبارت اسمی متناسب با regex تعریف شده‌ی ما است و I_NP یعنی inside NP که به معنای توکنی است که به عنوان شروع کننده‌ی عبارت اسمی نیست و بعد از یک B_NP آمده است و جزوی از آن عبارت اسمی است.

معمولا B_NP یک Determiner یا در صورت نبود یک صفت است و اگر صفت هم نبود یک عبارت اسم مفرد است.

مرحله ۴ :

بخش a:

برای استفاده از مدل از پیش آموزش داده شده‌ی Stanford باید از کلاس StanfordNERTagger در nltk استفاده کنیم و در ورودی آن مسیر فایل مدل و مسیر فایل jar را بدهیم و بعد با استفاده از تابع tag. موجود در کلاس آن مجموعه توکن‌ها را تگ گذاری کنیم.

بخش b:

در این بخش عملیات NER بر روی متن b اعمال شد. قطعه کد آن و خروجی به صورت زیر است.

```
In [37]: from nltk.tag.stanford import StanfordNERTagger

In [39]: model_path ='./stanford_data/english.all.3class.distsim.crf.ser.gz'
         jar_path = './stanford_data/stanford-ner-4.2.0.jar'
         ner_tagger = StanfordNERTagger( model_path,jar_path,encoding="utf8")

In [42]: text_b_tokens = nltk.word_tokenize(text_b)

In [47]: tags = ner_tagger.tag(text_b_tokens)
         tags

Out[47]: [('Jensen', 'PERSON'),
          ('Huang', 'PERSON'),
          (',', 'O'),
          ('the', 'O'),
          ('CEO', 'O'),
          ('of', 'O'),
          ('Nvidia', 'ORGANIZATION'),
          (',', 'O'),
          ('the', 'O'),
          ('nation', 'O'),
          ("'", 'O'),
          ('s', 'O'),
          ('most', 'O'),
          ('valuable', 'O'),
          ('semiconductor', 'O'),
          ('company', 'O'),
          (',', 'O'),
          ('with', 'O'),
          ('a', 'O'),
          ('stock', 'O'),
```

بخش c:

در این بخش موجودیت‌های سازمانی و نام افراد استخراج شده از متن b به نمایش درامده است.

```
In [49]: # person tags
         [tag[0] for tag in tags if tag[1]=="PERSON"]

Out[49]: ['Jensen', 'Huang', 'Huang', 'Huang', 'Neal', 'Stephenson']

In [50]: # organization tags
         [tag[0] for tag in tags if tag[1]=="ORGANIZATION"]

Out[50]: ['Nvidia']
```

مرحله‌ی ۵:

بخش a:

برای استفاده از کتابخانه‌ی Spcay باید آن را و مدل خواسته شده را نصب کرد و مدل en_core_web_sm را لود کرد. سپس متن داده شده را به مدل لود شده داد.

بخش b:

در این بخش عملیات NER را برروی متن b انجام شد. قطعه کد و noun_chunk ها به صورت زیر آمده است.

```
In [42]: import spacy

In [44]: nlp = spacy.load('en_core_web_sm')

In [45]: doc = nlp(text_b)

In [110]: # noun chunks
          [item.text for item in doc.noun_chunks]

Out[110]: ['Jensen Huang',
           'the CEO',
           'Nvidia',
           'the nation’s most valuable semiconductor company',
           'a stock price',
           'a market cap',
           'the metaverse',
           'what',
           'Huang',
           '“a virtual world',
           'a digital twin',
           'ours',
           'Huang',
           'author Neal Stephenson’s Snow Crash',
           'collectives',
           'shared 3-D spaces',
           'virtually enhanced physical spaces',
           'extensions',
           'the Internet',
           'the metaverse',
           'the massively popular online games',
           'Fortnite',
           'Minecraft',
           'users',
           'virtual worlds']
```

بخش c:

در این بخش موجودیت‌های نام افراد، نام سازمان، پول، اعداد استخراج شده اند.

```
In [105]: [item.text for item in doc.ents]
```

```
Out[105]: ['Jensen Huang',
           'Nvidia',
           '645',
           '$400 billion',
           'Huang',
           'Huang',
           'Neal Stephenson’s',
           'Snow Crash',
           '3']
```

```
In [111]: # person entities
          [item.text for item in doc.ents if item.label_=="PERSON"]
```

```
Out[111]: ['Jensen Huang', 'Huang', 'Huang', 'Neal Stephenson’s', 'Snow Crash']
```

```
In [112]: # organization entities
          [item.text for item in doc.ents if item.label_=="ORG"]
```

```
Out[112]: ['Nvidia']
```

```
In [113]: # money entities
          [item.text for item in doc.ents if item.label_=="MONEY"]
```

```
Out[113]: ['645', '$400 billion']
```

```
In [114]: # cardinal entities
          [item.text for item in doc.ents if item.label_=="CARDINAL"]
```

```
Out[114]: ['3']
```

مرحله ۶ :

در کتابخانه‌ی Spacy توابع متعدد برای دیدن entity ها خاص در یکجا وجود دارد و همچنین به نسبت nltk دقیق تر است و قابلیت آن را دارد که موجودیت‌های اسمی جزئی تری (مثل money) را پیدا کند و همچنین در تشخیص موجودیت‌های اسمی دو بخشی نسبت به nltk قوی تر کار کرده است مثل نام و نام خانوادگی.