NLP_HM1

سوال اول

ابهام در پردازش زبان های طبیعی در 4 سطح :

1- speech recognition
2- syntactic
3- semantic
4- discourse

1- **ابهام در سطح گفتار :** معمولا زمانی رخ می دهد که سیستم قادر به تشخیص صحیح کلمه نیست. این مسئله به دلایلی چون هم آوا بودن کلمات ، کیفیت پایین صدا ، نویز پس زمینه و لهجه ها یا تلفظ های مختلف ممکن است رخ دهد.

مثال :

- تشخیص "سلام علی کم" به جای "سلام علیکم"
- تشخیص "خون" به جای "خونه" (خانه)
- "وقت" و "بخت"

2- **ابهام نحوی :** زمانی رخ می‌دهد که یک جمله به دلیل ساختار خود چند تفسیر مختلف دارد و می‌تواند به شکل‌های مختلفی معنی شود.

مثال: جمله‌ی **"پسر دختر من را دید"** می‌تواند به دو شکل تفسیر شود:

- تفسیر اول : پسر، دختر مرا دید
- تفسیر دوم : پسرِ دخترِ من کسی را دید

مثال: جمله‌ی **"دختر عموی سارا که پزشک است، به مسافرت رفت"** می‌تواند به دو شکل تفسیر شود:

- تفسیر ۱ : دختر عموی سارا، که خودش پزشک است، به مسافرت رفت.
- تفسیر ۲ : دختر عموی سارا که به مسافرت رفته، پزشک است.

3- **ابهام در سطح معنا :** این نوع ابهام زمانی ایجاد می‌شود که یک جمله یا عبارت با اینکه از نظر ساختاری واضح است، می‌تواند چندین معنی داشته باشد.
مثال: جمله‌ی **"علی یک عکس از دوستش به او داد"** می‌تواند دو تفسیر داشته باشد:

- تفسیر اول: علی عکسی از دوستش به او داد
- تفسیر دوم: علی عکسی به دوستش داد

4- ابهام در گفتمان: معمولا زمانی رخ می‌دهد که معنای دقیق یا هدف اصلی یک بخش متن با سایر قسمت‌ها مبهم باشد. این نوع ابهام زمانی به وجود می‌آید که جملات گفته شده طولانی باشند و به علت نبود اطلاعات لازم، قابل‌درک نباشند

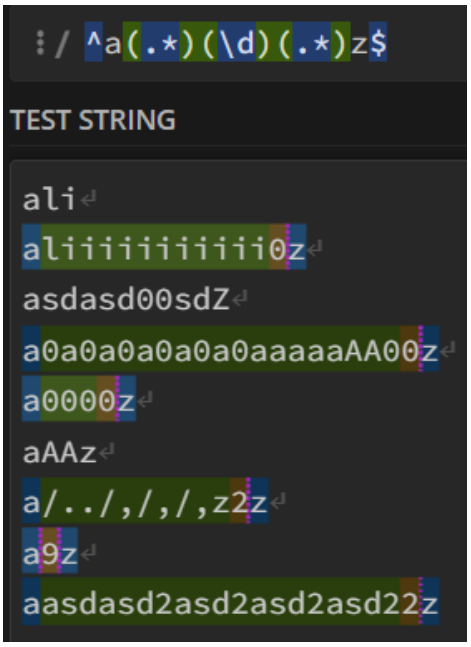
مثال: **"علی و سعید دیروز بحث کردند. او از این اتفاق ناراحت شد"**
در این جمله معلوم نیست که چه کسی ناراحت شده است؟ علی یا سعید؟

مثال: **"لیلا به علی گفت که دیر خواهد رسید"**
این جمله می‌تواند دو معنا داشته باشد: اول اینکه لیلا به علی گفته که خودش (لیلا) دیر خواهد رسید.
دوم اینکه لیلا به علی گفته که شخص دیگری دیر خواهد رسید.

سوال دوم

جواب بخش اول :

^a(.*)(\d)(.*)z$



بخش دوم:

```
^([^0-9]*\d){3}[^0-9]*$
```

توضیح کاراکتر اول (^) : نشان‌دهنده شروع رشته است.

`[^0-9]*` : 0 یا بیشتر کاراکترهایی که عدد نیستند.

/d : آمدن یک عدد = [0-9]

{3} : به این معنا است که این گروه قبلی باید 3 مرتبه تکرار شود.

$: نشان‌دهنده انتهای رشته است.

خلاصه الگو

این الگو به دنبال رشته‌ای است که دارای **دقیقاً سه عدد** باشد و شروع رشته با کاراکتر های غیر عددی باشد.

رشته هایی که این الگو می پذیرد :

- abc1ali2ghi3
- a1b2c3
- xyz3#@!5*9
- #1$2%3&

رشته هایی که این الگو نمی پذیرد :

- 1a2b3c4
- Abc
- 4abc2def3

بررسی رشته **'jhg4jhbhj291'** :

رشته با کاراکتر غیر عددی شروع شده است.

اولین عدد 4 است

بعد از عدد 4 این عبارت آمده است : **jhbhj**

دومین عدد 2 است

بعد از عدد 2 سومین عدد مشاهده می‌شود که 9 است

رشته باعدد 1 پایان می‌یابد، که عدد چهارم در این رشته است

از آنجایی که الگوی regex دقیقاً سه عدد را می‌پذیرد این رشته پذیرفته نمی‌شود.

سوال سوم

برای محاسبه احتمال S با استفاده از Chain Rule باید از فرمول زیر استفاده کنیم:

$P(S) = P(He) \times P(eats \mid He) \times P(an \mid He, eats) \times P(apple \mid He, eats, an)$

$P(S) = 0.1 * 0.4 * 0.3 * 0.5 = 0.006$

سوال چهارم

مفهوم tokenization : فرایندی که در یک رشته متنی استفاده میشود تا از آن رشته متنی واحد های کوچکتری را استخراج کند که به آن توکن می‌گویند.

این توکن‌ها می‌توانند کلمات، زیرکلمات یا حتی حروف مجزا باشند.

برای مثال در جمله "من به باشگاه میروم" دارای توکن های زیر است :

- من
- به
- باشگاه
- میروم

اما tokenization در زبان چینی !

در زبان چینی و برخی زبان‌های دیگر مانند ژاپنی جمله ها بدون استفاده از فاصله‌ها نوشته می‌شود

در این نوع زبان ها به دلیل مشخص نبود مرزهای کلمه، این فرایند به مسئله‌ای پیچیده‌تر تبدیل می‌شود که معمولاً نیاز به استفاده از ابزارهای قطعه‌بندی یا روش‌های زیرکلمه‌ای دارد که در ادامه روش های موجود رو بررسی می‌کنیم.

1. توکن‌سازی مبتنی بر کاراکتر :

در این روش هر کاراکتر چینی به عنوان یک توکن در نظر گرفته می‌شود که این ساده‌ترین روش است، اما ممکن است از نظر معنایی دقیق نباشد، زیرا کاراکترهای منفرد معمولاً به تنهایی یک کلمه کامل را نشان نمی‌دهند.

2.توکن‌سازی مبتنی بر کلمه (قطعه‌بندی) :

این روش برای تقسیم متن به کلمات معنادار استفاده میشود، مشابه با روشی که فاصله‌ها در زبان انگلیسی مرز کلمات را مشخص می‌کنند.

الگوریتم‌های مشهور قطعه‌بندی شامل: **Jieba** و **ICTCLAS**

3. توکن‌سازی زیرکلمه‌ای (WordPiece , Byte-Pair Encoding) :

این روش به‌طور معمول در مدل‌هایی مانند BERT یا GPT برای توکن‌سازی چینی استفاده می‌شود. در این روش، متن به واحدهای زیرکلمه‌ای تقسیم می‌شود که می‌توانند به کوچکی کاراکترهای منفرد یا ترکیب‌های چندکاراکتری باشند.

سوال پنجم

Lemmatization : فرآیندی است که در آن کلمات به "ریشه‌ی لغوی" برگردانده می‌شوند. این ریشه‌ی لغوی، شکل پایه‌ای یک کلمه است که در فرهنگ لغت وجود دارد.

Lemmatization : تلاش می‌کند کلمه را به شکل پایه و معنی‌دارش برگرداند. برای این کار، معمولاً به نوع کلمه (مثل اسم، فعل، صفت و غیره) توجه می‌شود.

نحوه کار این عمل به این صورت است که اول نوع کلمه را مشخص می‌کند. نوع کلمه می‌تواند اسم، فعل، صفت، یا قید باشد.

Final Result:

"Running" → "Run"

"runners" → "runner"

"ran" → "run"

"goals" → "goal"

"Cats" → "Cat"

"paws" → "paw"

"are" → "be"

"cleaner" → "clean"

"dogs" → "dog"

Case Folding : یک تکنیک پیش‌پردازش متن است که در آن تمام حروف یک متن به حروف کوچک تبدیل می‌شوند. این کار به منظور یکسان‌سازی داده‌ها و کاهش حساسیت به اختلافات حروف بزرگ و کوچک انجام می‌شود.

جمله بعد از این تکنیک

run runner run quickly to achieve their goal. cat paw be clean than dog

Normalization : به مجموعه‌ای از تکنیک‌ها اشاره دارد که هدف آنها یکسان‌سازی متن و کاهش تنوع داده‌ها است تا پردازش آن آسان‌تر شود. این تکنیک‌ها شامل مراحل مختلفی هستند که می‌توانند در کنار هم استفاده شوند که عبارت است از :

- تبدیل به حروف کوچک
- حذف نشانه‌گذاری
- حذف کلمات توقف
- تبدیل اختصارات و اشکال مختلف به یک شکل واحد
- حذف فضای خالی اضافی

جمله بعد از این تکنیک:

run runner run quickly achieve goal cat paw clean dog

Stemming یکی از تکنیک‌های پیش‌پردازش متن است که به معنای کاهش کلمات به ریشه‌ی خود (stem) است. برخلاف Lemmatization که به شکل دقیق‌تری کلمات را به فرم پایه برمی‌گرداند،Stemming معمولاً با حذف پسوندها یا پیشوندها انجام می‌شود و ممکن است به کلمات غیر واقعی یا نادرست منجر شود.

runner → run

quickly → quick

achieve → achiev

run → run (بدون تغییر)

goal → goal (بدون تغییر)

cat → cat (بدون تغییر)

paw → paw (بدون تغییر)

clean → clean (بدون تغییر)

dog → dog (بدون تغییر)

سوال ششم

تمرین اول

```
#1.1
text = "I love teaching. If you do not love teaching what else can you love.
text = re.sub(r"\.","",text)
listOfWords = re.split(" ",text)
for i in range(len(listOfWords)):
    listOfWords[i] = listOfWords[i].lower()
listOfWords.sort()
new_list = {}
counter = 1
for j in range(len(listOfWords)):
    if(j!=len(listOfWords)-1):
        if(listOfWords[j]==listOfWords[j+1]):
            counter+=1
        else:
            new_list[listOfWords[j]] = counter
            counter = 1
    else:
            new_list[listOfWords[j]] = counter
            counter = 1
sorted_new_list = sorted(new_list.items(), key=lambda x:x[1],reverse=True)
print(sorted_new_list)
```

در این سوال با استفاده از کتابخانه re

نقطه های متن داده شده رو از متن حذف کردیم

```
text = re.sub(r"\.","",text)
```

```
listOfWords = re.split(" ",text)
```

بعد کلمات درون متن را با کاراکتر فاصله اسپلیت می‌کنیم

برای شمارش فراوانی تعداد کلمه من از یک الگوریتم استفاده کردم که اول کلمات داخل متن رو سورت می کنه و کلمه j و j+1 رو با هم مقایسه می کنه

اگر برابر باشه تعداد کلمه j یکی اضافه می شود

و اما اگر برابر نباشد به این معنی است که به سراغ کلمه جدید رفته ایم

حال باید کلمه j رو درون دیکشنری اد کنیم و باید counter را یک کنیم

```
if(j!=len(listOfWords)-1):
```

این if اول برای این است که اگر کلمه آخر در حال بررسی باشد j+1 اندیس تعریف نشده است و ارور می دهد.

```
sorted_new_list = sorted(new_list.items(), key=lambda x:x[1],reverse=True)
```

این بخش کد برای سورت کردن نهایی دیکشنری بر اساس مقدار (شمار فراوانی) استفاده می شود.

تمرین دوم

```
pattern = "^[a-zA-Z_][0-9A-Za-z_]*[0-9a-zA-Z_]*$"
txt = "_Ali236_H_"
result = re.search(pattern,txt)
if(result):
    print("valid")
else:
    print("not valid")
```

این جا یک الگو تعریف کردیم که رشته مورد نظر با حروف انگلیسی کوچک و بزرگ و یا آندراسکور می تواند شروع شود و در ادامه میتواند هر رشته از حروف کوچک و بزرگ ، اعداد و آندراسکور به تعداد صفر یا بیشتر میتواند تکرار شود.

در ادامه چک می کنیم که رشته مورد نظر شامل این الگو هست یا خیر ؟

تمرین سوم

– بخش اول

```
# print 'clean text'
import re
def clean_text(text):
    pattern = "[^A-Za-z0-9 ]"
    cleanText = re.sub(pattern,"",text)
    return cleanText
text = '''%I $am@% a %tea@cher%, &and& I lo
cleanText = clean_text(text)
print(cleanText)
```

برای این بخش یک تابع تعریف کرده ایم

الگوی مورد نظر ما این است که رشته هایی که شامل حروف انگلیسی کوچک و بزرگ و اعداد و فاصله را **نباشد** را بر میگرداند.

حال با استفاده از تابع sub رشته هایی که شامل این الگو هستند را با هیچی جایگزین می کنیم.

- بخش دوم

```
# print 'most frequent words'
cleanText = re.split(" ",cleanText)
for i in range(len(cleanText)):
    cleanText[i] = cleanText[i].lower()
cleanText.sort()
new_list = {}
counter = 1
for j in range(len(cleanText)):
    if(j!=len(cleanText)-1):
        if(cleanText[j]==cleanText[j+1]):
            counter+=1
        else:
            new_list[cleanText[j]] = counter
            counter = 1
    else:
            new_list[cleanText[j]] = counter
            counter = 1
sorted_new_list = sorted(new_list.items(), key=lambda x:x[1],reverse=True)
print(sorted_new_list)
```

کد این بخش همانند کد سوال اول می باشد.

محمد حقیقت - 403722042