تشخیص اعداد و یکاهای اندازه‌گیری در متن

آرین احدی نیا، دانا افاضلی، محمد ابول‌نژادیان

بازیابی پیشرفته اطلاعات

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

مقدمه

در این نوتبوک قصد داریم مسئله‌ی تشخیص واحدهای اندازه گیری را و همچنین مسئله تبدیل واحد را حل کنیم.

ساختار پروژه به شکل زیر است:

در کنار این نوتبوک ۲ پوشه قرار دارند که در پوشه `resources` فایل‌هایی که به عنوان دیتا یا مدل از قبل آماده شده هستند قرار دارند. مانند مدل `POSTagger` برای هضم و همچنین دیتای گرفته شده از سایت باحساب برای برخی از واحدها.

همچنین در پوشه `src` ، تمامی کدهای زده شده قرار دارد که به دلیل حجم زیاد و اینکه عملا در لایه نهایی استفاده نمیشوند از نوتبوک حذف شده اند. در ادامه و در قسمت مربوطه به توضیح آنها پرداخته میشود.

در انتهای هر بخش تعدادی تست کیس نمونه قرار داده شده تا فانکشنالیتی آن قسمت برسی شود.

در بخش اول قسمت اصلی یعنی پیدا کردن واحدها برسی میشود و در قسمت بعد تبدیل واحد برسی میشود.

وابستگی‌ها

```
# nothing fancy
%pip install hazm
%pip install pandas
```

```
from src.pipelines import *
from src.base import SpeechTagsPipeline
from src.utils import filter_non_units, convert, match_patterns
```

تعریف مساله

تعریف مسئله به این شکل است: یک استرینگ به عنوان ورودی به تابع `run(org_str)` داده میشود و خروجی آن مواردی از واحدها، عددهای مربوطه، مرجع عدد و واحد پیدا شده (در صورت وجود) و همچنین بازه این موارد در استرینگ میباشد.

روند کلی

پردازش متن برای بدست آمدن نتیجه مورد نظر از دو فاز تشکیل شده است.

1. یک خط لوله برای برچسب گذاری اعداد، واحدها، اسم‌ها و صفت‌ها.
2. تطبیق برچسب‌های بدست آمده با الگوهای زبانی عدد و یکا.

فاز اول: برچسب‌گذاری

با توجه به نیاز مساله، برچسب هایی از قبیل عدد، اسم، صفت ، کمیت، واحد و تعدادی برچسب کمکی تعریف می‌کنیم و در یک خط لوله تلاش می‌کنیم که این برچسب‌ها را با یک ترتیب منطقی پیدا کنیم. توجه بفرمایید که این برچسب‌ها هر یک در بازه‌ای قرار می‌گیرند بنابرین در نهایت با تجمیع این بازه‌های می‌توانیم بازه نهایی مورد نظر را بدست آوریم.

خط لوله برچسب گذاری

در یک خط لوله، به ترتیب منطقی ابتدا برچسب‌ها را با توجه به برچسب‌های قبلی بدست آمده، بدست می‌آوریم و در نهایت، تمام برچسب‌های سخن را برای فاز دوم برمی‌گردانیم.

ترتیب پیدا کردن برچسب‌ها به شکل زیر میباشد: ابتدا اعداد را پیدا میکنیم، سپس مراحل نسبتا زیادی را صرف پیدا کردن و تجزیه و تحلیل واحدها و کمیت‌های داخل متن میکنیم و در نهایت برچسب های مربوط به اسامی و صفت‌ها را پیدا میکنیم.

این قسمت به شکل یک پایپلاین پیاده سازی شده که در هر مرحله، یک پایپ استرینگ اولیه را به همراه تمامی برچسب‌های پیدا شده تحت یک لیست دریافت میکند. سپس با پردازش استرینگ ورودی، به لیست برچسب‌های پیدا شده، برچسب‌های خود را اضافه میکند. شمای یک برچسب در حالت کلی به شکل زیر میباشد:

src.base.SpeechTag:

```
class SpeechTag(object):

    def __init__(self, tag: Tag, span: Span, value: Union[int, float, str, None]):
        self.tag = tag
        self.span = span
        self.value = value

    @staticmethod
    def to_pattern_str(st_list):
        return ''.join(map(lambda st: st.tag.value, st_list))
```

همچنین شمای یک پایپ به صورت کلی به شکل زیر میباشد:

src.pipelines.get_xxx_tags_pipe:

```
 # Note the general prototpye for the pipe
def get_number_tags_pipe(org_str: str, speech_tags: List[SpeechTag]) -> List[SpeechTag]:
    # Some processing...
    scientific_nums = number_extractor(org_str)
    speech_tags = speech_tags + scientific_nums

    # Create newly found speech tags and add them to the main list upon validation
    for num in default_number_extractor.run(org_str):
        num_span = Span(num['span'][0], num['span'][1])
        if not any([x.span.is_overlapped(num_span) for x in scientific_nums]):
            speech_tags.append(NumberTag(num_span, num['value']))
```

```
    # Finally return the cumulated list
    return speech_tags
```

اعداد

برای پیدا کردن اعداد از دو روش همزمان استفاده میشود. نخست با استفاده از رجکس اعدادی که فرمت عددی دارند پیدا شده، و سپس با استفاده ابزار parsi.io به استخراج اعداد با فرمت متنی میپردازیم. در ادامه تست این تابع را مشاهده میکنید:

```
# Just to Show
[
    get_number_tags_pipe('دو کیلو و نیم', []),
    get_number_tags_pipe('دو کیلو و صد گرم', []),
    get_number_tags_pipe('یک متر و بیست سانت', []),
    get_number_tags_pipe('هشتاد و پنج صدم وات', []),
    get_number_tags_pipe('.21x10**50', []),
    get_number_tags_pipe('120.1', []),
    get_number_tags_pipe('120.', []),
    get_number_tags_pipe('1.022*10^23', []),
    get_number_tags_pipe('1.022', []),
    get_number_tags_pipe('1e3', []),
    get_number_tags_pipe('-18732.787e-100', [])
]
```

یکاها

واحدها به طرق مختلف با یک دیگر ترکیب می‌شوند و واحدهای جدید را بوجود می‌آورند. به عنوان مثال اضافه شدن یک پیشوند به یک واحد ساده، یک واحد پیشوندی بوجود می‌آورد (به عنوان مثال: کیلومتر = کیلو + متر) یا اضافه شدن یک مرتبه‌ساز (مربع، مکعب، مجذور) با یک واحد ساده یا یک واحد پیشوندی، یک واحد مرتبه‌دار بوجود می‌آورد (به عنوان مثال: کیلومتر مربع: کیلومتر + مربع).

عبارت‌های پایه‌ای سازنده واحدها عبارتند از:

- واحدهای ساده (Su) مانند گرم، نیوتن، پاسکال، متر و ...
- پیشوندها (Sip) مانند کیلو، میلی، مگا و ...
- مرکب‌سازها (Cmp) مانند "بر"، "در" و ...

حال میتوانیم علاوه بر واحدهای ساده، انواعی از واحدها را داشته باشیم:

- واحدهای پیشوندی (Pu)
- واحدهای مرتبه‌دار (Ou)
- واحدهای مرکب (U)

با توجه به اینکه نحو ترکیب واحدها کمابیش مشخص است، یک دستور نحوی میتوانیم برای آن تعریف کنیم. چنین تعریفی به طریق زیر خواهد بود.

$$U ::= Ou|UOu|UCmpOu$$

$$Ou ::= Pu|OrdmPu|PuOrdm$$

$$Pu ::= Su|SipSu$$

Su is defined by a set of units which we get in dataset.

Su: Simple Unit, Sip: SI Prefix, Pu: Prefixed Unit, $Ordm$: Order Modifier, Cmp: Compounder, U: Unit

با توجه به دستور فوق، میتوان درخت نحوی یک واحد بسیار پیچیده را ایجاد کرد. درخت زیر را در نظر بگیرید که واحد "کیلوگرم میلی‌متر مربع بر مجذور نانوثانیه" را توصیف میکند.

در این پیاده‌سازی، ما درخت نحوی واحدها را می‌سازیم. همچنین در داده اولیه در کنار پیشوندها ضریب آنها را و در کنار واحدها نسبت آنها به واحد SI را ذخیره می‌کنیم. همچنین رفتار مرتبه‌سازها و مرکب‌سازها را که با واحدشان دارند را در قالب یک تابع در کنار هر یک ذخیره می‌کنیم. به عنوان مثال مرکب‌ساز "بر" نسبت‌ها را بر هم تقسیم می‌کند و مرتبه‌ساز مربع نسبت مورد نظر را به توان دو می‌رساند. به این طریق با محاسبه بازگشتی، میتوانیم نسبت اندازه هر واحد مرکب به واحد SI مربوطه را بدست آوریم.

همچنین برای بدست آمدن کمیت مربوط به هر واحد، آن واحد را تحلیل ابعادی میکنیم. به عنوان مثال واحدهای نیرو در ابعاد قرار دارند. برای این منظور ابعاد کمیت‌های مختلف را در داده اولیه قرار می‌دهیم و با توجه به آن مشابه ضریب تبدیل به SI، ابعاد واحدهای مرکب را محاسبه می‌کنیم و آن را با کمیت‌های مختلف تطبیق می‌دهیم.

```
test_unit_pipeline = SpeechTagsPipeline([
    get_number_tags_pipe,
    get_quantity_name_tags_pipe,
    get_simple_unit_tags_on_names_pipe,
    get_simple_unit_tags_on_symbols_pipe,
    get_si_prefix_tags_on_names_pipe,
    get_si_prefix_tags_on_symbols_pipe,
    get_order_modifier_tags_pipe,
    get_unit_compounder_preposition_tags_pipe,
    speech_tags_sorter_pipe,
    get_prefixed_units_pipe,
    get_ordered_units_pipe,
    get_units_pipe
])
```

```
# Just to Show
[
    test_unit_pipeline.run('دو نیوتن بر متر مربع و نیم'),
    test_unit_pipeline.run('دو کیلومتر و صد گرم'),
    test_unit_pipeline.run('یک متر و بیست سانت'),
    test_unit_pipeline.run('هشتاد و پنج صدم وات'),
]
```

در هر مثال میتوانید واحدهایی که پیدا شده اند را به حالت درختی آنها، یعنی پیشوند، واحد، مرکب ساز و ... مشاهده کنید.

نقش‌های زبانی

در نهایت بعد از پردازش واحدها و یکاها، اسامی و صفت‌ها را برسی میکنیم. برای این منظور از یک مدل `POSTagger` استفاده میکنیم که در مدل آنرا از سایت این درس برداشته‌ایم. در این قسمت در واقع تمام مواردی که میتوانند `item` باشند را به عنوان اسم گرفته ایم. همچنین برای پیدا کردن توصیفات کمیتی (مانند سرعت زیاد که باید در جواب نهایی در نظر گرفته شود) صفت‌ها را نیز پیدا کرده‌ایم.

```
# Just to Show
[
    [(st.tag, st.span.to_tuple(), st.value) for st in get_postags_pipe('این یک تست عجیب و زیبا میباشد', [])],
    [(st.tag, st.span.to_tuple(), st.value) for st in get_postags_pipe('هیچگاه انقدر خفن خوابم نمیامده', [])],
    [(st.tag, st.span.to_tuple(), st.value) for st in get_postags_pipe('و چه کارهای خطیری که بخاطر فرار نمیکنیم', [])],
    [(st.tag, st.span.to_tuple(), st.value) for st in get_postags_pipe('و قسم به ددلاین', [])],
]
```

همانطور که میبینید مدل تگر میتوانست بهتر باشد...

فاز دوم: تطبیق با الگوها

در این قسمت، توابع جواب قسمت قبل را گرفته و با استفاده از پترن‌های هارد کد شده سعی میکنند که خواسته‌های مسئله را استخراج کنند. پترن ها به طور کلی به شکل زیر هستند:

```
patterns = [
    'D+\s*U+\s*N+',  # 1 علی ۳۰۰ کیلو شیشه خرید (also handles 200kg)
    'D+\s*U+',  # 2 اتوبان ۲۰۰ دقیقه تا فلان فاصله دارد
    'Q+\s*N+\s*J+\s*D+\s*U+',  # 3 وزن علی کوچولو ۲۰۰ کیلو است
    'Q+\s*N+\s*D+\s*U+',  # 4 وزن علی ۲۰۰ کیلو است
    'N+\s*J+\s*D+\s*U+',  # 5 علی کوچولو ۲۰۰ کیلو است
    'Q+\s*J+',  # 7 سرعت زیاد
    'N+\s*D+\s*U+\s*Q+',  # 8 شیشه ۲۰۰ کیلو وزن دارد
    'Q+\s*D+\s*U+',  # 9 به وزن ۲۰۰ کیلو
    'D+\s*U+\s*.\s*D+\s*U+',
    'D+\s*U+\s*.\s*D+\s*U+\s*N+',  # ۲ 11 متر و ۳۰ سانت پارچه خرید
]
```

```
# complete pipeline for final test

extractor_pipeline = SpeechTagsPipeline([
    get_number_tags_pipe,
    get_quantity_name_tags_pipe,
    get_simple_unit_tags_on_names_pipe,
    get_simple_unit_tags_on_symbols_pipe,
    get_si_prefix_tags_on_names_pipe,
    get_si_prefix_tags_on_symbols_pipe,
    get_order_modifier_tags_pipe,
    get_unit_compounder_preposition_tags_pipe,
    speech_tags_sorter_pipe,
    get_prefixed_units_pipe,
    get_ordered_units_pipe,
    get_units_pipe,
    filter_pipe,
    get_postags_pipe,
])
```

در نهایت به تابع نهایی میرسیم. که همانطور که خواسته شده بود تنها یک استرینگ به عنوان ورودی دریافت میکند. اول از همه برچست های مربوط به استرینگ را پیدا میکند و سپس با استفاده از توابع نوشته شده، پترن‌ها را استخراج و مچ میکند. در نهایت تمامی آیتم‌های ساخته شده را خروجی میدهد.

```
def run(org_str):
    speech_tags = extractor_pipeline.run(org_str)
    # print(speech_tags)
    created_objects_output = match_patterns(org_str, speech_tags)
    return created_objects_output
```

```
# Just to Show
test_list = [
    [print(o) for o in run('علی ۴۰ کیلو وزن دارد و بار به این سنگین است.')],
    [print(o) for o in run('تندی زیاد نیست')],
    [print(o) for o in run('شیشه خرید kg علی ۲.')]
]
```

تست روی داده ها و دیدن نتایج

در این قسمت تعدادی تست کیس آماده شده که تست و نمایش داده میشود. این تست کیس‌ها را میتوانید در `resources/unit_tests.txt` مشاهده کنید

```
tests_file = open("resources/unit_tests.txt", "rt")
tests = tests_file.read().split("\n")
tests_file.close()
for test in tests:
    created_objects = run(test)
    for unit_object in created_objects:
        print(unit_object)
```

بخش امتیازی: تبدیل واحد

با توجه به درختی که در قسمت قبل ساخته شده، میتوان به راحتی تبدیل واحد را نیز انجام داد. در این قسمت این قسمت را پیاده سازی و تست میکنیم. تابع نهایی به صورت `run_conver(value, source, destination)` است که آرگمان‌ها به ترتیب بیانگر مقدار برای تبدیل، واحد مبدا و واحد مقصد میباشند که دو ورودی آخر از جنس استرینگ میباشند. فرض شده که دو واحد ورودی به هم قابل تبدیل هستند.

این تابع نیز از دو بخش تشکیل شده. نخست واحدها تجزیه و تحلیل میشوند، سپس با تبدیل آنها به واحد SI تبدیل آنها صورت میگیرد.

```
# pipeline for analyzing units
unit_determiner_pipeline = SpeechTagsPipeline([
    get_number_tags_pipe,
    get_quantity_name_tags_pipe,
    get_simple_unit_tags_on_names_pipe,
    get_simple_unit_tags_on_symbols_pipe,
    get_si_prefix_tags_on_names_pipe,
    get_si_prefix_tags_on_symbols_pipe,
    get_order_modifier_tags_pipe,
    get_unit_compounder_preposition_tags_pipe,
    speech_tags_sorter_pipe,
    get_prefixed_units_pipe,
    get_ordered_units_pipe,
    get_units_pipe,
    filter_non_units,
])
```

```
def run_convert(val, source, destination):
    src_unit = unit_determiner_pipeline.run(source)[0]
    dst_unit = unit_determiner_pipeline.run(destination)[0]
    return convert(val, src_unit, dst_unit)
```

```
# Just to Show
[
    run_convert(1, 'کیلوگرم',
                'پیکوگرم'),
    run_convert(1, 'متر مربع',
                'هکتار'),
    run_convert(1, 'اتمسفر',
                'پاسکال'),
    run_convert(1, 'سال نوری',
                'متر'),
]
```