Computer Engineering Department

Natural Language Processing

Assignment 3

Ali Sedaghi
97521378

Table of contents

Theoretical questions

Q1- POS tagging via HMM[1]

Tags:

- Noun
- Modal
- Verb

Sentences:

- Mark(N) can(M) watch(V).
- Will(N) can(M) mark(V) watch(N).
- Can(M) Tom(N) watch(V)?
- Tom(N) will(M) mark(V) watch(N).

Step 1: Emission probability

Word	Noun	Modal	Verb
Tom	2/6	0/4	0/4
Mark	1/6	0/4	2/4
Watch	2/6	0/4	2/4
Will	1/6	1/4	0/4
Can	0/6	3/4	0/4

Step 2: Transition probability

- <S> Mark(N) can(M) watch(V). <E>
- <S> Will(N) can(M) mark(V) watch(N). <E>
- <S> Can(M) Tom(N) watch(V)? <E>
- <S> Tom(N) will(M) mark(V) watch(N). <E>

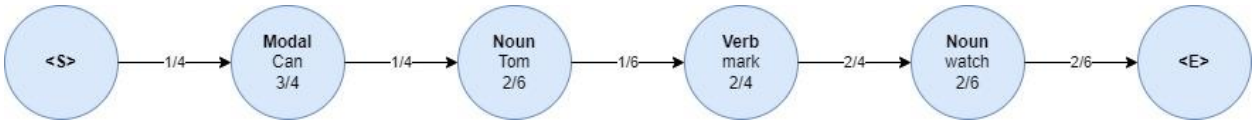
In the 3rd sentence I added <E> by myself.
In the below table rows are sources and columns are destinations.

[1] Hidden Markov Model

	Noun	Modal	Verb	<E>
<S>	3/4	1/4	0/4	0/4
Noun	0/4	3/6	1/6	2/6
Modal	1/4	0/4	3/4	0/4
Verb	2/4	0/4	0/4	2/4

Step 3: Graph

Sentence: Can Tom mark watch?



All other paths will be pruned because of zero weights.

$P = 1/4 * 3/4 * 1/4 * 2/6 * 1/6 * 2/4 * 2/4 * 2/6 * 2/6 = 0.00007233796$

Q2- CYK algorithm

John eats pie with cream

John	eats	pie	with	cream	
Noun(0.2) NP(0.04)	S(0.00384)	S(0.0000576)	-	S(0.000000414)	John
	Verb(0.3) VP(0.12)	VP(0.0018)	-	VP(0.00001296) VP(0.00001296)	eats
		Noun(0.1) NP(0.02)	-	NP(0.000144)	pie
			P(0.6)	PP(0.036)	with
				Noun(0.3) NP(0.06)	cream

ابتدا قطر اصلی را پر می‌کنیم:

Noun => John (0.2) NP => Noun (0.2) 0.2 * 0.2 = 0.04

Verb => eats (0.3) VP => Verb (0.4) 0.3 * 0.4 = 0.12

Noun => pie (0.1) NP => Noun (0.2) 0.1 * 0.2 = 0.02

P => with (0.6)

Noun => cream (0.3) NP => Noun (0.2) 0.3 * 0.2 = 0.06

سپس خانه‌های بالایی قطر اصلی:

John, eats: Noun Verb | Noun VP | NP Verb | NP VP = S (0.04 * 0.12 * 0.8 = 0.00384)

eats, pie: Verb Noun | Verb NP = VP (0.3 * 0.02 * 0.3 = 0.0018) | VP Noun | VP NP

pie, with: Noun P | NP P

with, cream: P Noun | P NP = PP (0.6 * 0.06 * 1.0 = 0.036)

خانه‌های بالاتر:

John eats pie:

John, eats pie: Noun VP | NP VP = S (0.04 * 0.0018 * 0.8 = 0.0000576)

John eats, pie: S Noun | S NP

eats pie with:

eats, pie with: -

eats pie, with: VP P

pie with cream:

pie, with cream: Noun PP | NP PP = NP (0.02 * 0.036 * 0.2 = 0.000144)

pie with, cream: -

خانه‌های بالاتر:

John eats pie with:

John, eats pie with: -

John eats, pie with: -

John eats pie, with: S P

eats pie with cream:

eats, pie with cream: Verb NP = VP (0.3 * 0.000144 * 0.3 = 0.00001296) | VP NP

eats pie, with cream: VP PP = VP (0.0018 * 0.036 * 0.2 = 0.00001296)

eats pie with, cream: -

سلول آخر:

John eats pie with cream:

John, eats pie with cream: Noun VP | NP VP = S (0.04 * 0.00001296 * 0.8 = 0.00000041472)

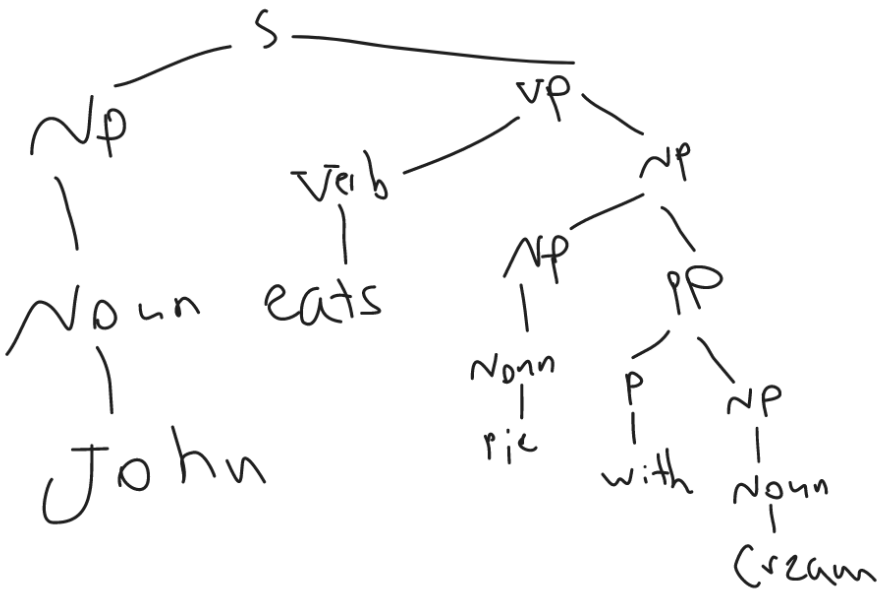
John eats, pie with cream: S NP

John eats pie, with cream: S PP

John eats pie with, cream: -

درخت تجزیه:

با توجه به تساوی سلول سبز رنگ در جدول، دو درخت تجزیه می‌توان رسم کرد.


S
NP
VP
Noun
Verb
NP
eats
NP
PP
John
Noun
P
NP
pie
with
Noun
Cream

Q3- PCFGs limitations

این روش مبتنی بر یک فرض بزرگ یعنی فرض استقلال است که باعث می‌شود احتمالات را به سادگی در هم ضرب کنیم. اما می‌دانیم این فرض درست نیست و در جا رخ دادن قوانین نیز مهم است (عدم استقلال)

همچنین در این روش بایستی به صورت Bottom to Up یا بالعکس پیش برویم که باعث می‌شود نیازمند احتمالات فرزندان یا پدران باشیم.

همچنین در این روش از اطلاعاتی که توسط تحلیلگر لغوی به دست می‌آید استفاده نمی‌شود و این امر باعث می‌شود نسبت به مدل زبانی n-grams نتیجه خوبی نداشته باشیم.

این روش دارای یک Bias مشخص نیز می‌باشد: احتمال یک درخت کوچکتر از احتمال یک درخت بزرگتر بیشتر است.

همچنین اگر دو تحلیل متفاوت را با قوانین یکسان پیش ببریم احتمالات برابر خواهد بود که یک ضعف است.

Practical questions

Q1- POS tagger with decision tree[2]

از Treebank به عنوان دیتاست استفاده می‌کنیم. این پیکره دارای 3914 جمله و 100676 کلمه است که هر کلمه درون جمله دارای تگ POS می‌باشد. جمله‌های این پیکره به صورت زیر تگ شده اند:

```
[('Pierre', 'NNP'),
 ('Vinken', 'NNP'),
 (',', ','),
 ('61', 'CD'),
 ('years', 'NNS'),
 ('old', 'JJ'),
 (',', ','),
 ('will', 'MD'),
 ('join', 'VB'),
 ('the', 'DT'),
 ('board', 'NN'),
 ('as', 'IN'),
 ('a', 'DT'),
 ('nonexecutive', 'JJ'),
 ('director', 'NN'),
 ('Nov.', 'NNP'),
 ('29', 'CD'),
 ('.', '.')]
```

همچنین برای تگ کردن جملات خودمان از Tagger درون NLTK یعنی pos_tag استفاده می‌کنیم. این Tagger توسط Averaged Perceptron Tagger آموزش داده شده است. نمونه خروجی این تگ کننده به صورت زیر است:

```
[('This', 'DT'),
 ('is', 'VBZ'),
 ('my', 'PRP$'),
 ('friend', 'NN'),
 (',', ','),
 ('John', 'NNP'),
 ('.', '.')]
```

تابع feature_extractor با ورودی گرفتن یک جمله و ایندکس یه کلمه از آن جمله ویژگی‌های آن کلمه در آن متن را استخراج می‌کند و در قالب یک دیکشنری خروجی می‌دهد. این ویژگی‌ها شامل موارد زیر است:

[2] https://medium.datadriveninvestor.com/ai-pos-parts-of-speech-tagger-part-1-a3d6bd77ce5e

- خود کلمه
- آیا کلمه در اول جمله آمده است؟
- آیا کلمه در آخر جمله آمده است؟
- آیا کلمه با حرف بزرگ شروع شده است؟
- آیا تمامی حروف کلمه حرف بزرگ است؟
- آیا تمامی حروف کلمه حرف کوچک است؟
- اولین حرف کلمه (پیشوند به طول 1) چیست؟ (همچنین پسوند)
- دو حرف اول کلمه (پیشوند به طول 2) چیست؟ (همچنین پسوند)
- سه حرف اول کلمه (پیشوند به طول 3) چیست؟ (همچنین پسوند)
- کلمه قبلی چیست؟
- کلمه بعدی چیست؟
- آیا داخل کلمه - وجود دارد؟
- آیا کلمه یک رقم است؟
- آیا حرف بزرگی درون کلمه وجود دارد؟

```
1 pprint.pprint(feature_extractor(['This', 'is', 'a', 'sentence'], 2))
```

```
{'capitals_inside': False,
 'has_hyphen': False,
 'is_all_caps': False,
 'is_all_lower': True,
 'is_capitalized': False,
 'is_first': False,
 'is_last': False,
 'is_numeric': False,
 'next_word': 'sentence',
 'prefix-1': 'a',
 'prefix-2': 'a',
 'prefix-3': 'a',
 'prev_word': 'is',
 'suffix-1': 'a',
 'suffix-2': 'a',
 'suffix-3': 'a',
 'word': 'a'}
```

از تابع transform_to_dataset برای تبدیل دیتاست Treebank به زوج‌های X و y استفاده می‌کنیم که X بیانگر Feature های هر کلمه درون یک جمله است و y بیانگر تگ POS آن کلمه در آن جمله است.

80 درصد جملات این دیتاست را برای آموزش (3131 جمله - 80637 کلمه) و 20 درصد را برای تست (783 جمله - 20039 کلمه) نگه می‌داریم.

دیتاست که به صورت دیکشنری است را با تابع Dict Vectorizer به صورت بردار در می‌آوریم تا آموزش سریع تر صورت بگیرد.

با استفاده از Pipeline یک مدل شامل Dict Vectorizer و Decision Tree ایجاد می‌کنیم.

به دلیل محدودیت حافظه مدل را تنها روی 10000 داده اول آموزش می‌دهیم و ارزیابی می‌کنیم.

نتایج به صورت زیر می‌باشد:

```
Evaluation

[18]  1 train_accuracy = classifier.score(X_train, y_train)
      2 print(f"Train Accuracy: {train_accuracy}")

      Train Accuracy: 0.894688542480499

[19]  1 test_accuracy = classifier.score(X_test, y_test)
      2 print(f"Test Accuracy: {test_accuracy}")

      Test Accuracy: 0.8929088277858177
```

دقت روی کل داده آموزش: 89.46 درصد

دقت روی کل داده تست: 89.29 درصد

همچنین خروجی مدل ما روی یک جمله جدید به صورت زیر است:

```
1 my_sentence = 'This is my friend, John.'
2 my_tagged_sentence = my_pos_tag(word_tokenize(my_sentence), classifier)

[22]  1 pprint.pprint(my_tagged_sentence)

[('This', 'DT'),
 ('is', 'VBZ'),
 ('my', 'NN'),
 ('friend', 'NN'),
 (',', ','),
 ('John', 'NNP'),
 ('.', '.')]
```