به نام خدا

امیرمحمد کمیجانی ۹۹۵۲۲۰۳۲

گزارش مربوط به تحلیل کد و قسمت تئوری تمرین سری سوم

سوالات تئوری :

۱)

الف) چالش های NER : پاسخ این سوال از صفحه ۱۶۸ کتاب اصلی درس است.

- **Ambiguity of segmentation** : این چالش زمانی ایجاد میشود که ما باید تصمیم بگیریم چه چیزی یک entity هست یا خیر و محدودیت ها و مرزها و شرایط برای در نظر گرفتن entity لازم است.
- **Type Ambiguity** : ممکن است یک کلمه هم اسم یک شخص باشد هم اسم یک مکان باشد و به طور کلی در قسمت های مختلف به کار آید که باعث میشود تعیین اینکه آن کلمه دقیقا از چه نوعی است میتواند برای ما چالش برانگیز باشد.

Unlike part-of-speech tagging, where there is no segmentation problem since each word gets one tag, the task of named entity recognition is to find and label *spans* of text, and is difficult partly because of the ambiguity of segmentation; we

[1] In English, on the WSJ corpus, tested on sections 22-24.

68 CHAPTER 8 • SEQUENCE LABELING FOR PARTS OF SPEECH AND NAMED ENTITIES

need to decide what's an entity and what isn't, and where the boundaries are. Indeed, most words in a text will not be named entities. Another difficulty is caused by type ambiguity. The mention JFK can refer to a person, the airport in New York, or any number of schools, bridges, and streets around the United States. Some examples of this kind of cross-type confusion are given in Figure 8.6.

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

Figure 8.6 Examples of type ambiguities in the use of the name *Washington*.

ب) متن تاثیر بسزایی در NER دارد کما اینکه قبل از این مبحث در کتاب در مبحث POS بیان شد که یک کلمه ممکن است در متون مختلف دارای معانی متفاوتی باشد اینجا هم به این صورت است که یک کلمه ممکن است در یک متن معمولی دارای یک معنی عام باشد اما در یک متن تخصصی و علمی معانی دیگری داشته باشد.
یا دارای چند معنی باشد که هر دو معنی عام باشند ولی باید در متن متوجه شویم که هر کلمه چه معنی ای دارد.

ج)

طبق فرمولی که در HMM داریم میدانیم هر استیت فقط استیت قبلی خودش را در نظر میگیرد که در بعضی موارد میتواند ناکافی باشد و خطا را افزایش دهد.

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

در روش CRF اما فقط استیت قبلی را در نظر نمیگیریم و چند استیت قبل را در نظر میگیریم و به نوعی روش global را در پیش میگیریم و میتواند خطا را کاهش دهد و باعث افزایش دقت شود.

قسمتی از متن کتاب:

linear chain CRF

Each of these local features f_k in a linear-chain CRF is allowed to make use of the current output token y_i, the previous output token y_{i-1}, the entire input string X (or any subpart of it), and the current position i. This constraint to only depend on the current and previous output tokens y_i and y_{i-1} are what characterizes a **linear chain CRF**. As we will see, this limitation makes it possible to use versions of the efficient Viterbi and Forward-Backwards algorithms from the HMM. A general CRF, by contrast, allows a feature to make use of any output token, and are thus necessary for tasks in which the decision depend on distant output tokens, like y_{i-4}. General CRFs require more complex inference, and are less commonly used for language processing.

از دیگر تفاوت های این دو روش در این است که HMM یک مدل generative است و CRF یک مدل discriminative میباشد.این مورد زمانی اهمیت میابد که در سناریو های واقعی ما کلمات ناشناخته ممکن است زیاد ببینیم و هندل کردن این مورد در روش HMM باید با اضافه کردن فیچر ها و قوانین و قواعد جدید باشد تا هندل کنیم که در مدل های generative این کار دشوار میباشد.

متن این مورد در عکس زیر مشخص است :صفحه ۱۷۶

While the HMM is a useful and powerful model, it turns out that HMMs need a number of augmentations to achieve high accuracy. For example, in POS tagging as in other tasks, we often run into **unknown words**: proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. It would be great to have ways to add arbitrary features to help with this, perhaps based on capitalization or morphology (words starting with capital letters are likely to be proper nouns, words ending with *-ed* tend to be past tense (VBD or VBN), etc.) Or knowing the previous or following words might be a useful feature (if the previous word is *the*, the current tag is unlikely to be a verb).

Although we could try to hack the HMM to find ways to incorporate some of these, in general it's hard for generative models like HMMs to add arbitrary features directly into the model in a clean way. We've already seen a model for combining arbitrary features in a principled way: log-linear models like the logistic regression model of Chapter 5! But logistic regression isn't a sequence model; it assigns a class to a single observation.

Luckily, there is a discriminative sequence model based on log-linear models: the **conditional random field** (**CRF**). We'll describe here the **linear chain CRF**, the version of the CRF most commonly used for language processing, and the one whose conditioning closely matches the HMM.

Assuming we have a sequence of input words $X = x_1 ... x_n$ and want to compute a sequence of output tags $Y = y_1 ... y_n$. In an HMM to compute the best tag sequence that maximizes $P(Y|X)$ we rely on Bayes' rule and the likelihood $P(X|Y)$:

د) بر اساس این جدول در کتاب اصلی درس به این سوال پاسخ میدهیم:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	infinitive to	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+, %, &*	WRB	wh-adverb	*how, where*

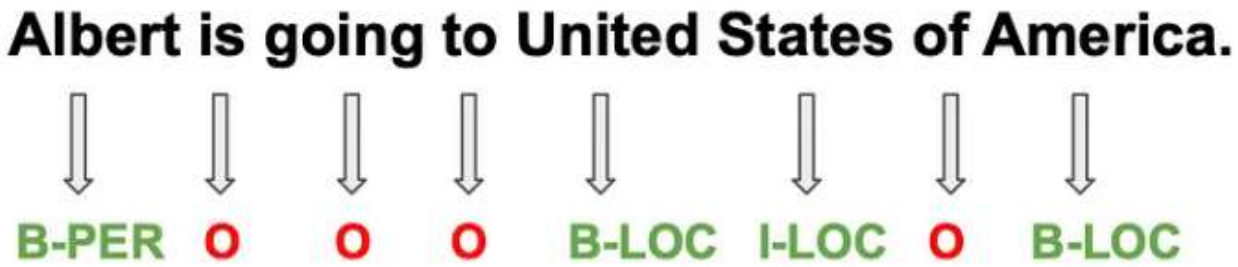
Figure 8.2 Penn Treebank part-of-speech tags.

- در جمله اول ، کلمه Atalanta یک اسم خاص مفرد میباشد در نتیجه tag مربوط به آن باید NNP باشد.
- در جمله دوم ، کلمه dinner یک اسم مفرد میباشد که برچسب فعلی آن نشان دهنده جمع بودن آن است که اشتباه است و باید به صورت NN باشد.
- در جمله سوم ، به صورت تقریبی همه چیز درست است اما میتوانیم برای Have تگ VBP را در نظر بگیریم که درست تر میباشد.
- در جمله چهارم can یک modal میباشد و باید tag آن اصلاح شود.

۵)

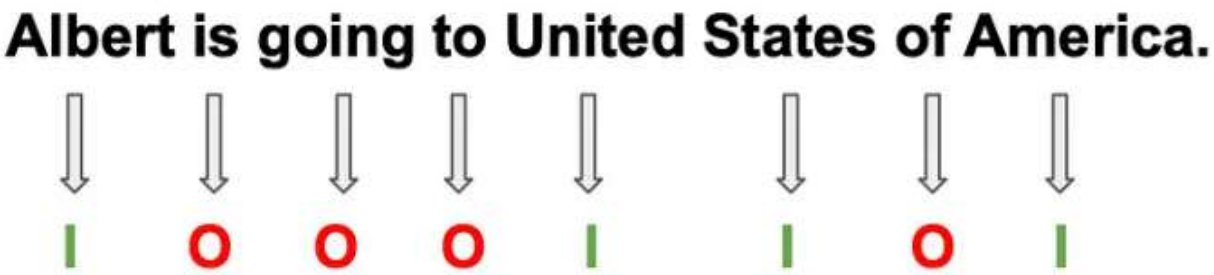
- روش BIO به این صورت عمل میکند که ، برای name entity ها سه نوع خاص را در نظر میگیریم و برای اسم های خاصی که دارای دو قسمت هستند مثل New York یا United States به کار میروند. به این صورت که به قسمت اول لیبل B (beginning) را میدهند و به کلمه دوم لیبل I (inside) را میدهیم و به کلماتی که اصلا name entity نیستند لیبل O را میدهیم.

مثال زیر برای این روش tagging است :



- روش IO : این روش ساده ترین روش tagging میباشد و فقط بررسی میکند که یک کلمه یا یک توکن یک name entity هست یا خیر اگر بود لیبل I و اگر نبود لیبل O را قرار میدهیم. تفاوتش با حالت قبل در این است که اسامی خاصی که دارای دو بخش بودند را به صورت جدا در نظر میگیرد و هر کدام را لیبلI میدهد.

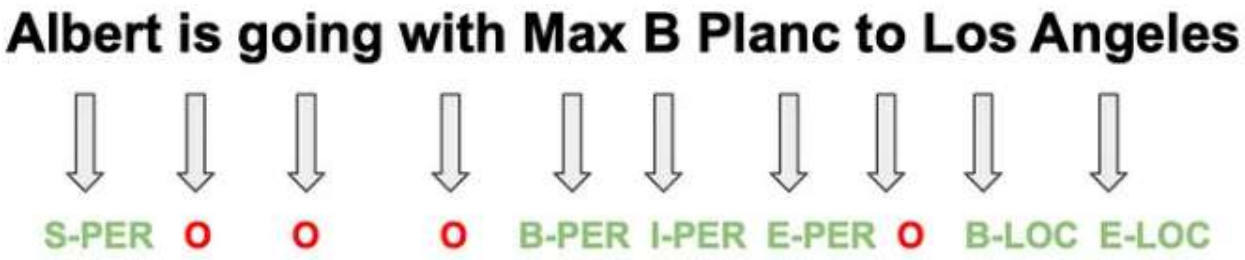
مثال زیر برای این روش است :



- روش BIOES :

این روش نسبت به حالت BIO اطلاعات بیشتری به ما میدهد و دارای طیف لیبلی بیشتری میباشد. و به این صورت است که ابتدا و انتهای اسامی خاص چند کلمه ای را نشان میدهد.همچنین توکن هایی که به صورت تکی هستند را با لیبل دیگری جدا میکند.

مثال زیر برای این روش مناسب است :



سوالات عملی :

۱) ابتدا دیکشنری خود را میسازیم. با ایتریشن بر روی samples که لیستی از تاپل ها است.یک دیکشنری میسازیم که کلمات و تگ مربوط به آنها را میسازد و به تعداد آنها در هر بار برخورد، یک مقدار اضافه میکند و اگر تگی وجود نداشت در نهایت تعداد آنرا ۱ میگذارد.

در قسمت بعد برای پیشبینی با ایتریشن بر روی مجموعه تست در هر مرحله بر اساس دیکشنری تگ هایی که داریم ابتدا خودمان پیشبینی میکنیم و در نهایت اگر پیشبینی ما با مقدار درست در مجموعه تست یکسان بود مقدار Accuracy را یکی افزایش میدهیم. و در نهایت مقدار Accuracy را بر روی کل طول مجموعه تست تقسیم میکنیم تا میزان دقت را به دست آوریم.

```
prediction = max(_tag_dict[word], key=lambda x: x['count'])['tag'] ### result = 788 more words got correctly classified.
# prediction = _tag_dict[word][0]['tag'] ### result = 1110 more words got correctly classified.
```

نکته ای که در پیاده سازی وجود دارد این قسمت از کد میباشد که هر دو درست هستند؛ فقط در اولی ، تگ هایی با بیشترین تکرار را انتخاب میکنیم در کل تاثیری در تحلیل و نتیجه نهایی به صورت زیادی نخواهد داشت صرفا تعداد کلماتی که بیشتر طبقه بندی میکنیم بیشتر میشود.(به نوعی محدودیت اضافی است و اهمیت چندانی ندارد)

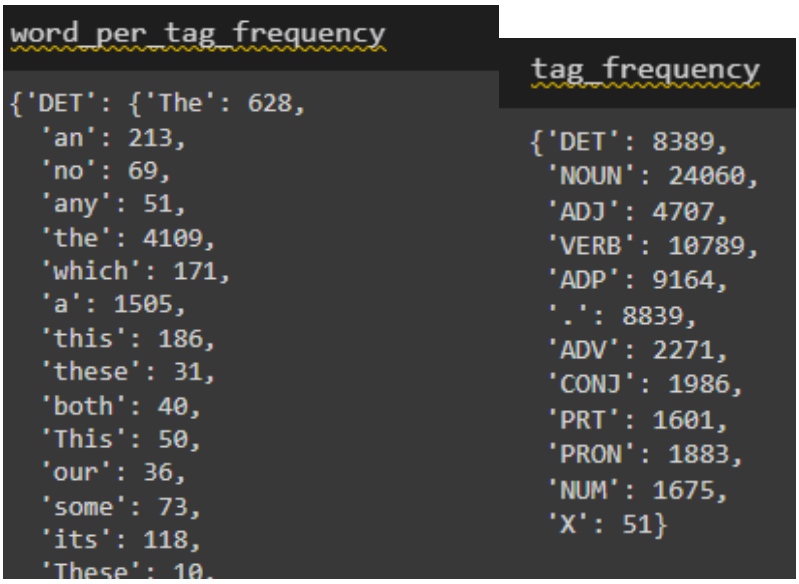
در قسمت بعدی rule هایی را اضافه میکنیم که باعث میشود دقت در عملیات تگ کردن برای کلمات ناشناخته بیشتر شود.

تحلیل : تحلیلی که داریم با اعمال یکسری قوانین و مرزبندی ها و فیچر گذاری ها میتوانیم عملیات تگ کردن برای کلمات ناشناخته را بهتر انجام دهیم.

که مشخص است بدون قوانین دارای دقت ۸۳ درصدی و با قوانین دارای دقت ۸۶ درصدی هستیم.

۲)

در تابع collect probabilities میخواهیم فرکانس و احتمال کلمات و تگ ها را بررسی کنیم.یا کلماتی که به عنوان هر تگ به کار رفته اند.



که مثلا به این صورت میشود.

به طور کلی در کد یک حلقه for در هر سمپل که به صورت یک تاپل است اعمال میکنیم و بررسی میکنیم که آیا یک تگ وجود دارد یا خیر و در صورت وجود مقدار آنرا یک مقدار افزایش میدهیم.همچنین ایندکس ها را نیز چک میکنیم که از مرز خود رد نشوند با توجه به اینکه ایندکس i+1 را در نظر داریم این مقدار را نیز کنترل میکنیم.

تابع confusion matrix ۴ حالت را بررسی میکند:

- TP : به درستی برگردانده شده
- TN : به درستی برگردانده نشده
- FP : به اشتباه برگردانده شده
- FN : به اشتباه برگردانده نشده

به این صورت است که مقدار پیشبینی شده با مقدار Actual بررسی میشود و اگر برابر بود یکی از حالات TP,TN میشود و در غیر اینصورت دو حالت دیگر.

در تابع Viterbi ، که الگوریتم با همین نام را پیاده سازی کردیم جملاتی را در مجموعه تست داریم توسط نقطه به انتها رسیده اند.اساس کار آن به این صورت است که برای بررسی پرفورمنس مدل از confusion matrix استفاده میکند و آنرا در هر مرحله پر

میکند.در این الگوریتم به دنبال پیدا کردن محتمل ترین دنباله از تگ ها برای کلمات در جمله هستیم. پیش بینی مدل از تگ ها را با مقدار actual مقایسه میکنیم و با توجه به ماتریسconfusion که پیشتر توضیح دادیم مقادیر درون این ماتریس را پر میکنیم. در نهایت با مقادیری که درست پیشبینی کردیم را با کل مقادیری که بررسی کردیم تقسیم کردیم تا میزان دقت مدل را ارزیابی کنیم. که در انتهای خروجی سِل مشخص است که به دقت ۹۱ درصد رسیده ایم.

۳)

الف) مواردی که میتوانند مشکل ساز شوند :

- **مقادیر تکراری :** در این دیتاست مقادیر تکراری برای اسم افراد وجود دارد مثل اسم Yilmaz Erdogan ، که ممکن است اسم دو شخص متفاوت باشد ولی راهی برای فهمیدن این تفاوت وجود ندارد و ممکن است مشکل ساز شود
- **نبودن مقدار در بعضی ستون ها :** در بعضی ستون ها مثلا در ستون certificate در فیلم my sassy girl خالی میباشد که میتواند ما را دچار مشکل کند. یا مقدار سود بعضی از فیلم ها دارای مقداری نیست.
- **صحیح نبودن فرم بعضی مقادیر:** به طور مثال در بسیاری از فیلم ها سال ساخت آنها به صورت عدد منفی ذخیره شده. حتی اگر این مورد هم بپذیریم بعضی از فیلم ها هستند که به صورت (year) نوشته شده اند که در آنها سال مثبت است در نتیجه این تضاد میتواند در parse کردن دیتا ما را دچار مشکل کند.
- **اشتباه وارد شدن اطلاعات بعضی ستون ها :** در بسیاری از فیلم ها که چند سری از آنها ساخته شده است شماره های هر سری به صورت I , II,III است ولی به جای اینکه در اسم مربوط به فیلم باشد در ستون سال ساخت وارد شده است مثلا فیلم Rush I

- **استفاده کردن از کلماتی که در زبان انگلیسی support نمیشوند:** اگر در قسمتی از اطلاعات از کلماتی خارج از زبان انگلیسی استفاده کنیم مثلا در فیلم های خارجی(غیر انگلیسی) از کلمات به همان زبان استفاده کنیم و مدل را بر اساس زبان انگلیسی آموزش داده باشیم ممکن است دچار مشکل شود.

ب)

اساس کار BIO را در سوالات تئوری توضیح داده ایم. به طور کلی فایل csv را خواندیم و سپس تایتل فیلم ها را استخراج میکنیم و از ست برای اینکه مقادیر تکراری کنار بگذاریم استفاده میکنیم. در مرحله بعد بر اساس منطق BIO کد را پیاده سازی میکنیم که قبلا توضیح داده ایم و در انتها خروجی میگیریم.