به نام خدا

هوم ورک سوم درس NLP

مهدی فقهی

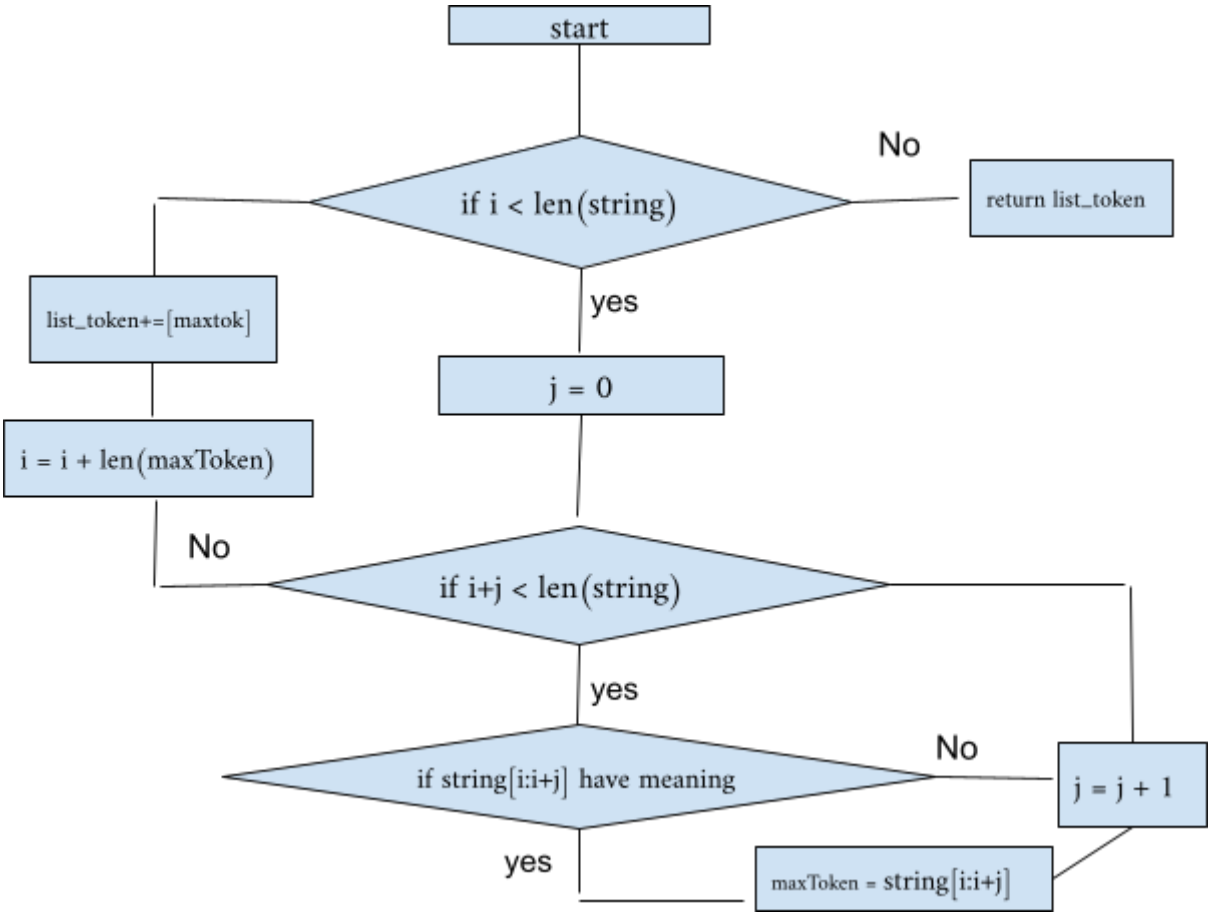
۴۰۱۷۲۲۱۳۶

Tokenization

A)

قسمت اول درباره این توضیح می‌دهیم که این الگوریتم greedy چطور کار میکند :

برای اینکار ابتدا از ابتدا شروع می‌کنیم به حرکت بر روی string داده شده تا بتوانیم یک ترکیب معنادار از کاراکترها را پیدا کنیم وقتی همچین ترکیبی را پیدا کردیم آن را به عنوان بزرگترین token در حال حاضر در نظر می‌گیریم و سپس هنوز نیز بر روی string داده شده حرکت می‌کنیم تا اگر ترکیب بزرگتری هنوز قابل پیدا کردن است آن را پیدا کنیم تا اینکه به پایان string می‌رسیم و بزرگترین token که در string توانسته‌ایم پیدا کنیم را به عنوان عضو جدید اضافه می‌کنیم . سپس به اندازه تعداد کاراکترهای بزرگترین token به جلو حرکت می‌کنیم و از آنجا به بعد دوباره همین روند را تکرار می‌کنیم و بزرگترین token با معنی را پیدا می‌کنیم تا به انتهای متن برسیم و در انتها لیست از token ها را گزارش می‌نماییم .



ما چرا این الگوریتم در زبان چینی خوب کار می‌کند اما در زبان فارسی و انگلیسی درست کار نمی‌کند :

مثلا مثال زیر را در انلگیسی در نظر بگیرید :

Thetabledownthere

اگر از الگوریتم برای اینکار استفاده کنیم حاصل :

چیزی که خواهیم داشت theta bled own there خواهد بود حال آنکه نوشته اصلی می‌تواند مورد زیر باشد : theta bled own there

یا در زبان فارسی می‌توانیم مورد زیر را داشته باشیم .

درخورپرازماهی‌است .

شکل اصلی :

در خور پر از ماهی است . (خور : شاخه‌ای (محدوده نیمه بسته آب) از دریا گفته می‌شود.)

شکل تشخیص داده شده :

درخور پر از ماهی است . (درخور : درخورنده)

زبان‌های فارسی و انگلیسی به دلیل اینکه گاهی بعضی از کلمات از ترکیب چند کلمه کوچک دیگر تشکیل شده‌اند که می‌توانند معنی داشته باشند یا کنارقرار گرفتن چند کلمه درکنار هم می‌تواند به چندین شکل که هرکدام معنای مناسبی داشته باشند جدا شود و این نوع ابهام‌ها بدون فاصله بدون توجه به معنای کلی جمله زیاد است و چون در این نوع روش به معنی کلی کلمات باهمدیگر توجه نمی‌شود این روش مانند مثل های بالا درست عمل نمی‌کند .

(B

1) Algorithm Byte Pair Encoding (BPE)

بعد از tokenization تعداد تکرار هر کلمه که رویداده است در training data مشخص می‌شود و بعد در قدم اول BPE یک base vocabulary شامل تمامی symbol ها تشکیل دهنده کلمات را

می‌سازد سپس در مرحله بعد، BPE یک واژگان پایه متشکل از تمام نمادهایی که در مجموعه کلمات منحصر به فرد وجود دارد ایجاد می کند و قوانین ادغام را می آموزد تا یک نماد جدید از دو نماد از واژگان پایه تشکیل دهد.
این کار را تا زمانی انجام می دهد که واژگان به اندازه واژگان مورد نظر برسد. توجه داشته باشید که اندازه واژگان مورد نظر یک فرد پارامتر است که قبل از آموزش tokenization باید تعریف شود.

فرض کنید ما یک متن را به شکل زیر tokanization کرده ایم .

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

براساس این vocabulary ما برابر است با :["b", "g", "h", "n", "p", "s", "u"]

تمامی کلمات را به symbolهای تشکیل دهنده آنان از این vocabulary ابتدا تقسیم می کنیم .

(h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5")

در قدم بعد دوتایی را پیدا می کنیم در از کلمات که بیشترین تکرار را در بین کلمات داشته باشد .
مثلا دوتایی که ابتدا "h" بیاید و سپس بعد از آن "u" بیاد در این document به تعداد ده بار در hug و پنج بار در hugs آمده که برابر با ۱۵ عدد می‌باشد در مقابل دوتایی "ug" به تعداد 10 بار در hug و پنج بار در pug و پنج بار در hugs آمده که برابر با ۲۰ بار تکرار است پس این ترکیب جدید به vocabulary اضافه می کنیم پس خواهیم داشت

("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)

در ادامه به دنبال بیشترین تکرار دیگر می گردیم که در این بین "un" باز با ۱۶ بار تکرار برنده می‌شود پس از این مورد "h" هنگامی که پس از آن "ug" بیاید اضافه می‌شود پس تا حالا vocabulary شامل موارد زیر است :

["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]

و لیست ما به شکل زیر در آمده :

("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)

اگر توی این مرحله Byte-Pair Encoding را تمام کنیم از این پس قوانین ادغام آموخته شده برای کلمات جدید اعمال می شود . البته باید در نظر داشته باشیم که کلمات جدید داده شده دارای symbol هایی باشند که در دایره کلمات ما در vocabulary قرار بگیرند . براساس این نوع Tokenization ما کلمه bug به شکل ["b" , "ug"] درمی‌آید و همچنین کلمه mug به این دلیل که m در vocalury ما نیست به شکل ["<unk>", "ug"] نوشته می‌شود که unk به معنا این است که کلمه اول ناشناخته است .

https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7

2) WordPiece Algorithm

این الگوریتم هم مثل الگوریتم بالا یک الگوریتم greedy است .

خوب از آنجا که این دو مدل بسیار شبیه هم هستند پس بهتر است ابتدا با این پرسش شروع کنیم که مشکل که BPE دارد چیست ؟

می تواند مواردی داشته باشد که در آن بیش از یک راه برای رمزگذاری یک کلمه خاص وجود دارد. سپس انتخاب subword token زیرکلمه برای الگوریتم دشوار می‌شود، زیرا هیچ راهی برای اولویت‌بندی اینکه کدام یک اول استفاده شود وجود ندارد. از این رو، ورودی یکسان را می توان با رمزگذاری های مختلف نشان داد که بر دقت نمایش های آموخته شده تاثیر می گذارد.

برای مثال :

Number	Token	Frequency
1	li	3
2	l	5
3	ea	2
4	eb	4
5	near	6
6	bra	2
7	al	4
8	n	5
9	r	1
10	ge	11
11	g	7
12	e	8

فرض کنید می‌خواهیم "linear algerbra " را می‌خواهیم tokenize در نتیجه خواهیم داشت :

linear = **li** + **near** *or* **li** + **n** + **ea** + **r**

algebra = **al** + **ge** + **bra** *or* **al** + **g** + **e** + **bra**

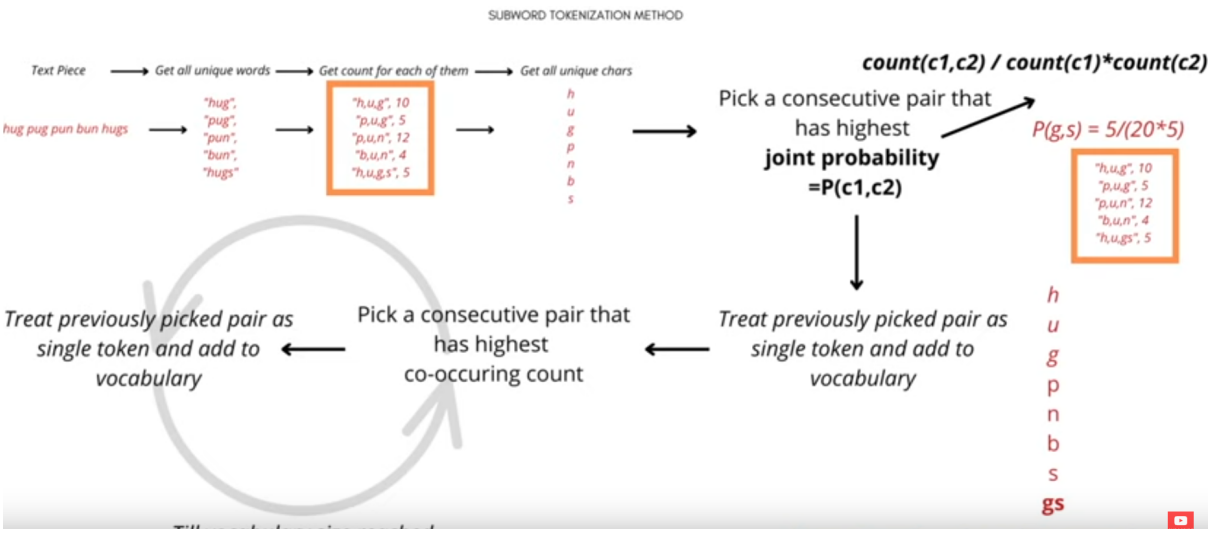
می بینیم که دو روش مختلف برای توکن کردن هر کلمه در عبارت داده شده وجود دارد که در مجموع چهار روش برای نشانه گذاری این عبارت ارائه می شود. بنابراین، همان متن ورودی را می توان به چهار روش رمزگذاری کرد و این در واقع یک مشکل است.

در WordPiece، ما در واقع همین کار را انجام می دهیم. تنها تفاوت WordPiece و BPE در نحوه افزودن جفت نمادها به واژگان است. در هر مرحله تکراری، WordPiece یک جفت نماد را انتخاب می کند که منجر به بیشترین افزایش احتمال در هنگام ادغام می شود. به حداکثر رساندن احتمال داده های آموزشی معادل یافتن جفت نمادی است که احتمال تقسیم آن بر احتمال اولین و به دنبال آن احتمال نماد دوم در جفت بیشتر از هر جفت نماد دیگری است.

به عنوان مثال، الگوریتم بررسی می کند که آیا احتمال وقوع "es" بیشتر از احتمال وقوع "e" و به دنبال آن "s" است یا خیر. ادغام فقط در صورتی اتفاق می افتد که احتمال "es" تقسیم بر "s" ،"e" از هر جفت نماد دیگری بیشتر باشد.

بنابراین، می‌توان گفت که WordPiece آنچه را که با ادغام دو نماد از دست می‌دهد ارزیابی می‌کند تا مطمئن شود قدمی که برمی‌دارد واقعاً ارزش آن را دارد یا نه.

با یک مثال این تفاوت را بهتر می‌بینیم .



به این شکل که به جای انتخاب بیشترین تکرار به سراغ بیشترین احتمال می‌رود یعنی در اینجا احتمال joint probability اینکه ابتدا حرف "g" بیاید و سپس به دنبال آن "s" بیاید برابر میشود برابر با :

(حالت‌هایی که g است * حالت‌هایی که S موجود است)/تعداد جایی که این اشتراک وجود دارد

و برای هر جفت هرکدام که این حاصل برایشان بیشتر باشد به در هر مرحله به subword tokens اضافه می‌شود .

پس الگوریتم WordPiece یک مدل زبان را بر اساس واژگان پایه آموزش می‌دهد، جفتی را انتخاب می‌کند که بیشترین احتمال را دارد، این جفت را به واژگان اضافه می‌کند، مدل زبان را روی واژگان جدید آموزش می‌دهد و مراحل تکرار شده را تکرار می‌کند تا به اندازه واژگان یا آستانه احتمال مورد نظر برسد. .

https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7

https://www.youtube.com/watch?v=mzJK2kJX3Bo

(C

در هر زبانی، کلمات با توجه به نقشی که در جملات ایفا می کنند، به شکل‌های ظاهری متفاوتی خواهند بود. اما با توجه به این که تمامی آن‌ها از یک ریشه (بُن) ساخته می شوند، از نظر معنا و مفهوم در گام های بعدی تشخیصی به ما کمک شایانی خواهند نمود.

ازهمین رو در بسیاری از روش های مبتنی بر NLP، ابتدا می بایست ریشه کلمات را پیدا کنیم. برای ریشه‌یابی کلمات معمولاً از دو روش **ریشه یابی**(**Stemming**) و **بُن واژه سازی** (**Lemmatization**) استفاده می‌شود که هر دو روش در نهایت ریشه‌ی یک کلمه را به دست می آورند.

عمل ریشه‌یابی این امکان را می‌دهند که فُرم‌های مختلفِ یک کلمه را به یک فُرمِ واحد تبدیل کنیم. با این کار تعداد ویژگی‌ها کمتر می‌شود و همچنین شکل‌های مختلفِ یک کلمه حذف شده و کامپیوتر می‌تواند شکل‌های مختلفِ یک کلمه را یکی در نظر بگیرد.

ریشه یابی (Stemming):

فرآیندی که طی آن ریشه یک واژه به دست می‌آید. مثل (کتاب‌ها به کتاب) یا "می‌خوردن" که دارای ریشه "خورد#خور" است. در فرآیند Stemming، برای بدست آوردن ریشه، یکسری گام‌ها به ترتیب انجام می شوند و در اصطلاح برای آن الگوریتم تعریف می‌شود برای مثال ابتدا پیشوند حذف شود. در گام بعدی به سراغ پسوندها می‌روند و گام‌های بعدی به همین صورت. ضعف جدی این روش گاها تولید ریشه بی معنی است مثلا ریشه "بشارت دادم" که ممکن است "ب" به عنوان پیشوند در نظر گرفته و نتیجه غلط شود. برای زبان انگلیسی معمولا از الگوریتم **Porter** استفاده می کنند .

بُن واژه سازی (Lemmatization)

هر دوی Stemming و Lemmatization یک کار یکسان ولی به روش مختلف انجام می دهند. Lemmatization شبیه به این است که با دیدن یک کلمه، به سراغ دایره المعارف رفته و بررسی های لازم را انجام داده و یک کلمه را به عنوان ریشه بر می گردانیم. مزیت این روش تولید نشدن کلمات نامربوط به عنوان ریشه است. البته سرعت این روش نسبت به Stemming پایینتر است چون باید جستجو انجام شود.

مثلا (کتاب‌ها به کتاب) برمی گردد البته توانایی تشخیص این مورد در stemming هم هست یا رفتیم و برویم که حالت گذشته و حال یکدیگر هستند با اینکه بن‌های یکسانی ندارند ولی از نظر معنای در کنار هم قرار می گیرند.

است ، بود ، گشت و گردید هم می‌توانیم همه را به یک شکل نشان دهیم .

https://virgool.io/@mohammad_d/%D9%BE%DB%8C%D8%B4-%D9%BE%D8%B1%D8%AF%D8%A7%D8%B2%D8%B4-%D9%85%D8%AA%D9%88%D9%86-%D9%81%D8%A7%D8%B1%D8%B3%DB%8C-%D8%A8%D8%A7-%D8%A7%D8%B3%D8%AA%D9%81%D8%A7%D8%AF%D9%87-parsivar-cmug6xkqbl6d

https://dataio.ir/%D9%BE%D8%B1%D8%AF%D8%A7%D8%B2%D8%B4-%D8%B2%D8%A8%D8%A7%D9%86-%D9%87%D8%A7%DB%8C-%D8%B7%D8%A8%DB%8C%D8%B9%DB%8C-nlp-2100-r8pw3ahib6xs

https://chistio.ir/%DB%8C%D8%A7%D9%81%D8%AA%D9%86-%D8%B1%DB%8C%D8%B4%D9%87-%DA%A9%D9%84%D9%85%D8%A7%D8%AA-stemming-lemmatization/

Language Models (Theory)

(a

مدل زبان در NLP یک مدل آماری احتمالی است که احتمال وقوع یک دنباله معین از کلمات را در یک جمله بر اساس کلمات قبلی تعیین می کند. به پیش بینی اینکه کدام کلمه بیشتر در جمله ظاهر می شود کمک می کند. ورودی یک مدل زبان معمولاً مجموعه ای آموزشی از جملات مثال است. خروجی یک توزیع احتمال بر روی دنباله ای از کلمات است. می‌توانیم از آخرین کلمه (unigram)، دو کلمه آخر (bigram)، سه کلمه آخر (trigram) یا n کلمه آخر (n-gram) برای پیش‌بینی کلمه بعدی بر اساس نیاز خود استفاده کنیم.

Unigram :

در اینجا احتمال وقوع هر کلمه وابسته به دیگر کلمات نیست و مستقل از بقیه برحسب رخداد خود کلمه در کل داده‌های تست انجام می‌شود و برای پیدا کردن احتمال رخداد زنجیره‌ای کلمات W1 ,W2,W3,...,Wn حاصل برابر است با ضرب احتمال تک تک رخدادها :

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

bigram :

دراینجا احتمال وقوع را برحسب احتمال رخدا برحسب کلمه قبل از کلمه‌ای که احتمال رخداد آن را می‌خواهیم حساب کنیم بررسی می کنیم . برای پیدا کردن احتمال رخداد زنجیره‌ای کلمات s ,W1,W2,W3,...,Wn,/s حاصل برابر است ضرب احتمال رخداد کلمه اول به شرط اینکه کلمه اول باشد در رخداد کلمه دوم به شرط آنکه کلمه اول قبل از آن باشد تا زمانی که به رخداد کلمه آخر برسیم به شرط آخرین کلمه بودن .

که به صورت فرمول ریاضی برابر است با :

$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

trigram:

همانند bigram است با این تفاوت که به جای یک قبل به دو کلمه قبل از کلمه احتمال رخداد آن را می‌خواهیم بررسی کنیم توجه می کنیم .

ngrams:

در اینجا برحسب n-1 کلمه قبل احتمال رخداد کلمه بعد را می‌سنجیم و برای پیدا کردن احتمال رخداد کل جمله این احتمالات را در هم ضرب می‌نماییم .

(b

for all m < V - M + 1, the smoothed probability is greater than the unsmoothed probability

(C

ابتدا لیست vocabulary را پیدا می کنیم که در ابتدا به شکل :

{ (ایران ، ۳) (سرای ، ۲) (است ، ۳) (من ، ۱) (امید ، ۱) (بوستان ، ۱) (آبی ، ۱) (بانوان ، ۱) (آسمان ، ۱) (دخترانه ، ۱) (علی ، ۱) (دایی ، ۱) (اسطوره ، ۱) }

سپس سعی می کنیم از Lemmatization استفاده کنیم در نتیجه خواهیم داشت :

{ (ایران ، ۳) (سرای ، ۲) (است ، ۳) (من ، ۱) (امید ، ۱) (بوستان ، ۱) (آب ، ۱) (مونث ، ۲) (آسمان ، ۱) (علی ، ۱) (دایی ، ۱) (اسطوره ، ۱) }

همان طور که می‌بینید کلمه بانوان و دخترانه به جنسیت کلی خود یعنی جنس مونث Lemmatiz می‌شوند وهمچنین کلمه آبی منسوب به کلمه آب می‌شود .
پس داده‌های ورودی تست را نیز همانند train هنگام ورود Lemmatization می کنیم .

<s> آسمان آب است </s>

<s> مونث دایی امید ایران است </s>

</s> ایران مونث <unk>سرای بوستان است <s>

<s> سرای من ایران است </s>

برای پیدا کردن احتمال به کمک unigram جدولی به شکل زیر می‌سازیم .
تعداد vocabulary را برای اینکار برابر با ۱۳ عدد میگیریم که پنج خط داریم .

ایران	سرای	است	من	امید	بوستان	آب	مونث	آسمان	علی	دایی	اسطوره	UNk
۳+۱	۲+۱	۳+۱	۱+۱	۱+۱	۱+۱	۱+۱	۲+۱	۱+۱	۱+۱	۱+۱	۱+۱	0+۱
۴	۳	۴	۲	۲	۲	۲	۳	۲	۲	۲	۲	۱
۴/ ۱۳+ ۵	۳/ ۱۳+ ۵	۴/ ۱۳+ ۵	۲/ ۱۳+ ۵	۲/ ۱۳+ ۵	۲/ ۱۳+ ۵	۲/ ۱۳+ ۵	۳/ ۱۳+ ۵	۲/ ۱۳+ ۵	۲/ ۱۳+ ۵	۲/ ۱۳+ ۵	۲/ ۱۳+ ۵	۱/ ۱۳+ ۵

در نتیجه براساس جدول بالا احتمال جمله های test براساس unigram prediction برابر است :

جمله اول :

۰.۰۰۲۷۴۳۴۸۴=۴/۱۸*۲/۱۸*۲/۱۸

جمله دوم :

۰.۰۰۰۱۰۱۶۱۱ = ۴/۱۸ * ۴/۱۸ * ۲/۱۸ * ۲/۱۸ * ۳/۱۸

جمله سوم :

۲/۱۸ * ۲/۱۸ * ۱/۱۸ * ۳/۱۸ * ۲/۱۸ * ۴/۱۸ = 0.000002823

جمله چهارم :

۳/۱۸ * ۲/۱۸ * ۴/۱۸ * ۴/۱۸ = 0.000914495

برای ساخت جدول برای bigram داریم .

	ایران	سرای	است	من	امید	بوستان	آب	مونت	آسمان	علی	دایی	اسطوره	UNK
ایران	0	2	1	0	0	0	0	0	0	0	0	0	0
سرای	0	0	0	1	1	0	0	0	0	0	0	0	0
است	0	0	0	0	0	0	0	0	0	0	0	0	0
من	0	0	1	0	0	0	0	0	0	0	0	0	0
امید	0	0	0	0	0	0	0	0	0	0	0	0	0
بوستان	0	0	0	0	0	0	0	1	0	0	0	0	0
آب	0	0	0	0	0	0	0	0	1	0	0	0	0
مونث	0	0	1	0	0	0	0	0	0	0	0	0	0
آسمان	0	0	0	0	0	0	0	1	0	0	0	0	0
علی	0	0	0	0	0	0	0	0	0	0	1	0	0
دایی	0	0	0	0	0	0	0	0	0	0	0	1	0
اسطوره	۱	0	0	0	0	0	0	0	0	0	0	0	0
UNK	0	0	0	0	0	0	0	0	0	0	0	0	0

برای اینکه Laplace Smoothing اعمال بشه داریم .

	ایران	سرای	است	من	امید	بوستان	آب	مونت	آسمان	علی	دایی	اسطوره	UNK
ایران	0+1	2+1	1+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
سرای	0+1	1+1	0+1	1+1	1+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
است	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
من	0+1	0+1	1+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
امید	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
بوستان	0+1	0+1	0+1	0+1	0+1	0+1	0+1	1+1	0+1	0+1	0+1	0+1	0+1
آب	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
مونث	0+1	0+1	1+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
آسمان	0+1	0+1	0+1	0+1	0+1	0+1	0+1	1+1	0+1	0+1	0+1	0+1	0+1
علی	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	1+1	0+1	0+1
دایی	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	1+1	0+1
اسطوره	1+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1
UNK	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1	0+1

برای محاسبه احتمال جمله اول :

<s> آسمان آب است </s>

احتمال اینکه P(آسمان|<s>)

برابر است با :1/(5+13)

احتمال اینکه P (آسمان | آب)

برابراست با : 1/ (5+13)

احتمال اینکه P (آب | است)

برابراست با : 1 / (5+13)

احتمال اینکه(است | <s>)

برابراست با : 3 / (5 + 13)

در نتیجه احتمال جمله برابراست با :

0.000028578 = 3/18*1/18*1/18*1/18

برای محاسبه احتمال جمله دوم :

<s> مونث دایی امید ایران است </s>

احتمال اینکه P(مونث|<s>)

برابر است با :1/(5+13)

احتمال اینکه P (مونث | دایی)

برابر است با :1/(5+13)

احتمال اینکه P (دایی |امید)

برابر است با :1/(5+13)

احتمال اینکه P (امید |ایران)

برابر است با :1/(5+13)

احتمال اینکه P (ایران | است)

برابر است با :2/(5+13)

احتمال اینکه (است | </s>)

برابراست با : 3/(13+5)

احتمال کل برابراست با :

0.000000176 = 3/18*2/18*1/18*1/18*1/18*1/18

<s> ایران مونث <unk>سرای بوستان است </s>

احتمال اینکه P(ایران|<s>)

برابر است با :2/(5+13)

احتمال اینکه (ایران | مونث) P

برابراست با : 1/(13+5)

احتمال اینکه (مونث | Unk)

برابراست با : 1/(13+5)

احتمال اینکه (Unk | سرای)

برابراست با : 1/(13+5)

احتمال اینکه (سرای | بوستان)

برابراست با : 1/(13+5)

احتمال اینکه (بوستان |است)

برابراست با : 1/(13+5)

احتمال اینکه (است | </s>)

برابراست با : 3/(13+5)

احتمال این جمله برابراست با :

0.000000001 = 3/18*1/18*1/18*1/18*1/18*1/18*1/18*2/18

<s> سرای من ایران است </s>

احتمال اینکه P(سرای|<s>)

برابراست با : 1/(13+5)

احتمال اینکه (سرای |من)

برابراست با : 2/(13+5)

احتمال اینکه (من |ایران)

برابراست با : 1/(13+5)

احتمال اینکه (ایران | است)

برابراست با : 2/(13+5)

احتمال اینکه (است | </s>)

برابراست با : 3/(13+5)

0.000006351 = 1/18*2/18*1/18*2/18*3/18

Language Models(Code):

a :

همانند توضیحات در notebook ابتدا punctuation را با فاصله punctuation فاصله تعویض می کنیم .

سپس از استرینگ ورودی کمک split نسبت به فاصله و نیم فاصله توکن‌ها را بدست می‌آوریم .

```
def tokenize(text):

    for punc in string.punctuation:
        text = text.replace(punc,' '+punc+' ')

    text_without_space = re.split(" |\u200c",text)

    return text_without_space
```

b :

در قسمت بعد سعی می کنیم n-grams یک لیست از tokenها را برحسب اینکه مقدار n چی‌باشد بدست می‌آوریم .
به انتها یک توکن <End> اضافه می کنیم و به تعداد n-1 عدد توکن <START> اضافه می کنیم .
سپس به ازای هر کلمه در لیست tokenها از<start> token به بعد تعداد n-1 توکن قبل را به عنوان tuple که context شناخته می‌شود در نظر می گیریم .

```
def ngrams(n, tokens):
    tokens.append('<END>')
    for i in range(n-1):
        tokens = ['<START>']+ tokens
    # print(tokens)
    list_ngram = []
    for i in range(n-1,len(tokens)):
        # print(i)
        item_before = []
        for j in range(n-1):
            # print(j)
            item_before = [tokens[i-j-1]]+ item_before
        list_ngram.append((tuple(item_before),tokens[i]))

    return list_ngram
```

در ادامه کلاس NgramModel را داریم .
در قسمت update جمله‌ها را یکی یکی از ورودی می گیریم و از آنها یک tokens لیست می‌سازیم و به مدلمان زبانیمان اضافه می کنیم .
در قسمت prob هم براساس فرمول احتمال شرطی مقدار احتمال رخداد هر token به شرط context مدنظر را می‌سنجیم .

```python
    def update(self, sentence):
        self.res_str += ngrams(self.n,tokenize(sentence))

    '''
        for calculate prob of each token with its context .
        at first must calculate number of context in the all res_str as denominator.
        then calculate number of tokens come after this context as numerator.
        at end we return division of numerator and denominator for probability .
    '''

    def prob(self, context, token):

        denominator = 0
        numerator = 0
        # print(self.res_str)
        for item in self.res_str:
            if item[0] == context:
                denominator += 1
                if item[1] == token:
                    numerator += 1

        return numerator/denominator
```

سپس قسمت random token را داریم که در آن به کمک یک مقدار r براساس فرمول احتمال ذکر شده در زیر یک token را برمی گردانیم .

$$\sum_{j=1}^{i-1} P(t_j \mid \text{context}) \leq r < \sum_{j=1}^{i} P(t_j \mid \text{context}).$$

```
    Find all token that come with context
    I sort it by alphabet .
    then I find random.number
    then I find word that sum of the prob of word before it with this token upper
    then I find prob of each item .

'''

def random_token(self, context):

    r = random.random()
    map_to_probs = {}
    token_of_interest = [item[1] for item in self.res_str if item[0]==context]
    for token in token_of_interest:
        map_to_probs[token] = self.prob(context, token)

    summ = 0
    for token in sorted(map_to_probs):
        summ += map_to_probs[token]
        if summ > r:
            return token
    return "<END>"
```

سپس به کمک random text از تابع random token کمک می گیریم و به تعداد کلمه‌های داده شده یک text برمی گردانیم . در اینجا برای آنکه جمله ما به واقعیت نزدیک‌تر شود مقدار context به ازای هر کلمه جدیدی که وارد جمله می‌شود تغییر می‌دهیم .

```
I generate a text with number token function give with random_token.
for better result I make context with use each word I generate with random_token.
"""
def random_text(self, token_count):

    list_item = []
    "add n-1 <start> for context for first context"
    new_context = ["<START>" for i in range(self.n-1)]

    # new_context = tuple(new_context)
    for count in range(token_count):

        find_item = self.random_token(tuple(new_context))
        list_item.append(find_item)
        if self.n > 1 :
            if find_item == '<END>':
                "if find <end> as item I want to make new sentence because of that change the new context like first one ."
                new_context = ["<START>" for i in range(self.n-1)]
            else:
                "update new context with new word we find "
                new_context.pop(0)
                new_context.append(find_item)

    return " ".join(list_item)
```

قسمت آخر نیز مربوط به حساب کردم perplexity هست که برای محاسبه آن ابتدا یک text را گرفته سپس ngram آن را حساب می‌کنیم و سپس براساس ngram موجود احتمال رخداد هر وقوع را برحسب تابع prob که داریم حساب می‌کنیم و در هم ضرب می‌کنیم سپس از وارون آن رادیکال به فرجه تعداد token های text داده شده می‌گیریم و حاصل را برمی‌گردانیم .

```
"""
calculate perplexity like formulation . at first I find ngram list then I calculate prob of each token and then mul it with res mul of befor
 token in sentence.
"""
def perplexity(self, sentence):
    ngram_list = ngrams(self.n,tokenize(sentence))
    res_mul = 1
    for item in ngram_list:
        res_mul = res_mul * self.prob(item[0],item[1])

    return (1/res_mul)**(1/len(ngram_list))
```

در قسمت نهایی یک تابع نوشته شده است که یک متن می‌گیرید و سپس یکی یکی جمله را به Ngram model می‌دهیم و آن را update می‌کنیم البته \n را از خط به خط داده‌هایی که به آن می‌دهیم حذف می‌کنیم .

```
With this function I read all sentence of text file and update of ngram model that make with n.

def create_ngram_model(n, path):
    model = NgramModel(n)
    file1 = open(path, 'r')
    lines = file1.readlines()

    # x = 0
    # list_add = []
    for line in lines:
        model.update(line.strip())
    return model
```

در ادامه می‌توانید نتایج را در notebook مشاهده کنید و با نمونه‌های تست مقایسه کنید و ببینند که همان‌ها است .