به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

**درس پردازش زبان طبیعی**

تمرین 2 بخش دوم

نام و نام خانودگی: علی عطاءاللهی

شماره دانشجویی: 810199461

**فروردین ماه 1403**

فهرست

[پاسخ سوال دوم 3](#_Toc163187046)

[پاسخ بخش اول – پیش پردازش مجموعه داده 3](#_Toc163187047)

[پاسخ بخش دوم – بارگذاری Glove 4](#_Toc163187048)

[پاسخ بخش سوم – آموزش مدل 4](#_Toc163187049)

[پاسخ سوال سوم 5](#_Toc163187050)

[پاسخ بخش اول 5](#_Toc163187051)

[پاسخ بخش دوم 13](#_Toc163187052)

[پاسخ بخش سوم 14](#_Toc163187053)

# پاسخ سوال دوم

## پاسخ بخش اول – پیش پردازش مجموعه داده

ابتدا داده را با كمك كتابخانه pandas مي‌خوانيم.سپس با استفاده از كتابخانه seaborn نمودار لیبل‌ها را رسم مي كنيم.

A graph with a bar and a number of text

Description automatically generated with medium confidence

در تابع clean\_headline\_text، متن را به حالت استاندار و تمیز تبدیل می‌کنیم. این کار شامل حذف علائم غیرالفبا، تبدیل کاراکترها به شکل کوچک، جدا کردن کلمات، حذف کلمات stopword و متصل کردن کلمات باقیمانده به هم می‌شود.A screen shot of a computer code

Description automatically generated

## پاسخ بخش دوم – بارگذاری Glove

Glove : از یک روش تعبیه سازی کلمات استفاده می‌کنیم که آن را از قبل دانلود کرده و کده را از حالت zip خارج کرده و از glove.6B.100d.txt به عنوان منبع استفاده می‌کنیم. سپس یک دیکشنری embedding از کلمات ایجاد می‌کنیم. حالا جملات را ابتدا توکنایز می‌کنیم و سپس برای تمام جملات دیتاست، بردار glove مربوط به کلمات آن‌ها را بدست می‌آوریم. برای این کار کافی است کلمات جمله را در دیکشنری embedding بخوانیم و مقدار آن‌ها را ذخیره کنیم. بعد از اینکه بردار glove را برای همه کلمات جمله بدست آوردیم، از آن‌ها میانگین می‌گیریم و آن را برمی‌گردانیم تا به عنوان ویژگی برای classification استفاده کنیم.

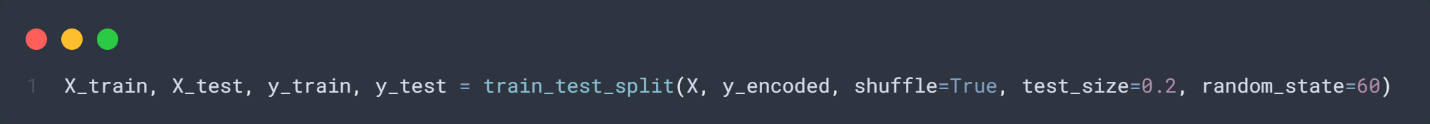
A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

## پاسخ بخش سوم – آموزش مدل

در مرحله بعد، داده‌ها را با استفاده از ماژول train\_test\_split از کتابخانه Sklearn به دو بخش train و test تقسیم می‌کنیم. این کار را با شافل کردن داده‌ها انجام می‌دهیم و نسبت را طبق صورت پروژه به نسبت 80 به 20 درصد قرار می‌دهیم.

# پاسخ سوال سوم

## پاسخ بخش اول

preprocess

ما در این تمرین از معماری skip-gram با استفاده از negative sampling برای پیاده‌سازی word2vec استفاده می‌کنیم.

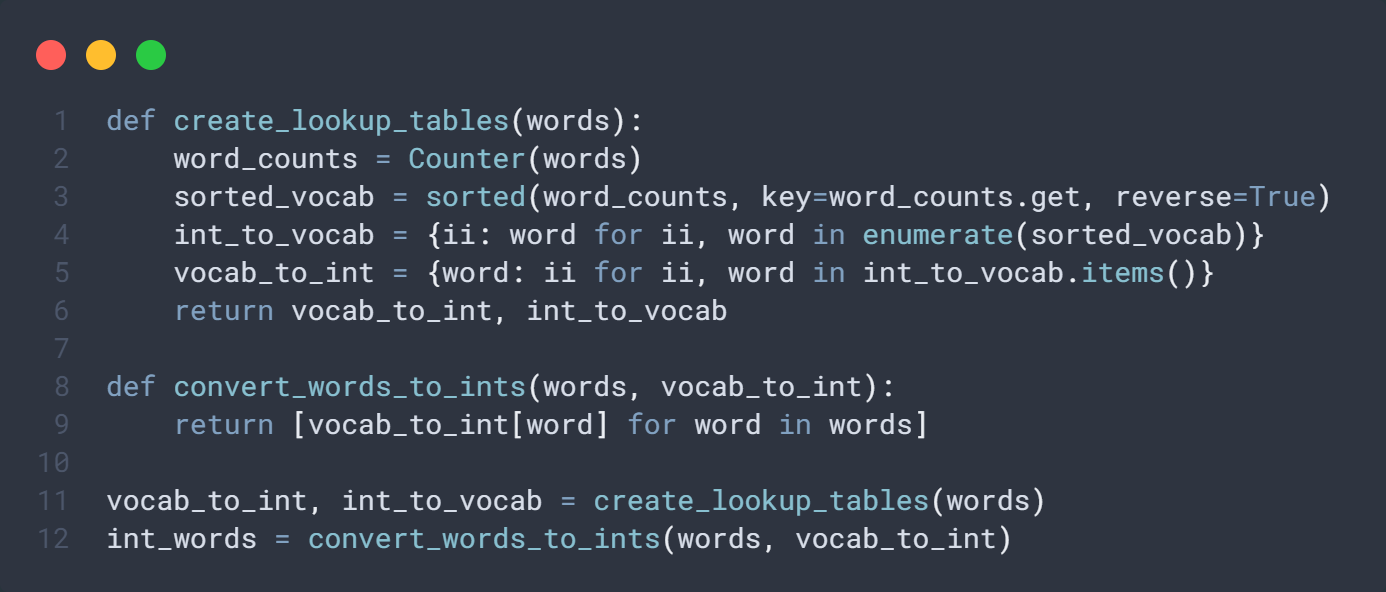
A diagram of a network

Description automatically generated with medium confidence

در این روش، حسن آموزش این روش نسبت به CBOW را به دلیل سرعت بالاتر آموزش آن برجسته می‌کند. در این روش، مدل یک کلمه را دریافت می‌کند و باید کلمات اطراف آن را پیش‌بینی کند. این امکان وجود دارد که شبکه را برای نمایش کلمات در آموزش قرار دهیم، به گونه‌ای که کلمات با یک context مشابه، بردار embedding مشابهی داشته باشند.

create\_lookup\_tables

در این تابع، دو دیکشنری پیاده‌سازی شده است. یکی از این دیکشنری‌ها کلمه را دریافت می‌کند و ایندکس آن را برمی‌گرداند، و دیکشنری دیگر برعکس عمل می‌کند. از این تابع برای ساخت این دو دیکشنری برای کلمات corpus استفاده می‌کنیم.



برخی کلمات مانند "for" یا "a" برای کلمات همسایه، معمولاً context خوبی فراهم نمی‌کنند. بنابراین، حذف آن‌ها به بهبود کیفیت تعبیه سازی کلمات کمک می‌کند و سرعت یادگیری را افزایش می‌دهد. این فرآیند به زیرمجموعه‌گیری اطلاق می‌شود. برای هر کلمه ، ما آن را با احتمال زیر حذف می‌کنیم:

در اینجا، تعداد تکرار کلمه در دیتاست است و t پارامتر حد آستانه است. پس در پیاده‌سازی برای تمام کلمات، تعداد تکرار هر کلمه را حساب می‌کنیم و از فرمول بالا احتمال حذف آن را محاسبه می‌کنیم. سپس با کمک تابع random، اگر عدد تصادفی تولید شده کمتر از این احتمال باشد، آن کلمه را حذف می‌کنیم.

A computer screen shot of a computer code

Description automatically generated

get\_target

دیتای ما حالا مناسب برای دادن به مدل شده است. در معماری skip-gram، برای هر کلمه در متن، باید context اطراف آن را تعریف کنیم و تمام کلماتی که در پنجره به اندازه C اطراف آن قرار دارند را بدست آوریم. به عبارت دیگر، C/2 کلمه قبلی و C/2 کلمه بعدی را به عنوان همسایگان کلمه داده شده در نظر می‌گیریم. در اینجا، سایز C را 4 در نظر گرفتیم. بنابراین، دو کلمه قبل و بعد جزو context حساب می‌شوند.

A computer code with text

Description automatically generated with medium confidence

get\_batches

حالا باید یک تابع بسازیم که برای مدل، batch از ورودی و تارگت بسازد. ورودی‌ها کلمات داخل corpus هستند و تارگت کلمات همسایه آن‌ها هستند که می‌توان برای بدست آوردن آن‌ها از تابع get\_target کمک گرفت. برای این کار، اولاً فقط batchهای کامل را در نظر می‌گیریم، یعنی batch آخر اگر سایز کافی را نداشته باشد، حذف می‌شود. سپس برای هر batch به ازای تمام کلمات آن، همسایگان آن را بدست می‌آوریم..

A computer screen shot of a program code

Description automatically generated

cosine\_similarity

برای اینکه ببینیم مدل ما در هر مرحله چقدر پیشرفت داشته، علاوه بر loss می‌توانیم در هر مرحله شباهت کلمات corpus را با یک سری از کلمات سنجید و چند تای نزدیک آن را چاپ کنیم. برای اندازه‌گیری نزدیکی، از کسینوس دو بردار استفاده می‌کنیم. این یک راه خوب و جانبی برای ارزیابی مدل ما است.

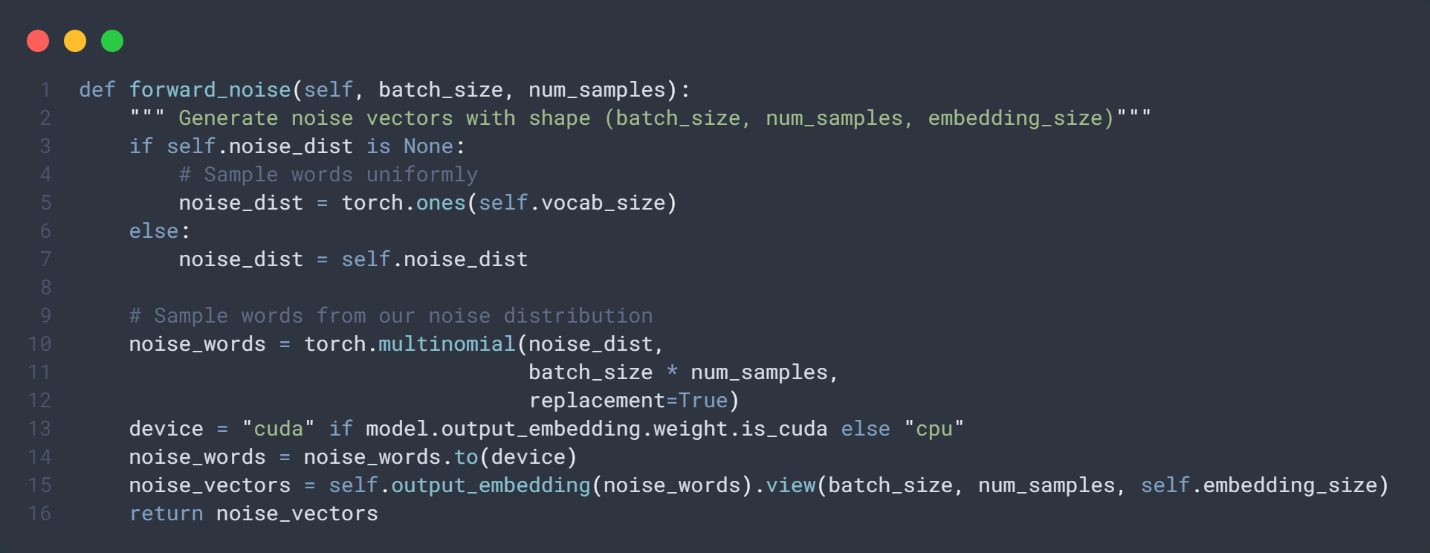
A screen shot of a computer program

Description automatically generated

تابع forward\_noise در كلاس SkipGramNeg

در skip-gram، که فقط از کلمات context استفاده می‌کنیم و بقیه کلمات را non-context در نظر می‌گیریم، دو مشکل وجود دارد:

* برای هر نمونه training، فقط وزن‌های مربوط به کلمات context ممکن است تغییر معناداری کنند، در حالی که در فرآیند back-propagation ما تلاش می‌کنیم تمام وزن‌های لایه‌های مخفی را آپدیت کنیم. وزن‌های مربوط به کلمات non-context بسیار اندک یا اصلاً تغییر نمی‌کنند، یعنی در هر مرحله آپدیت وزن‌ها اسپارس است.
* برای هر نمونه training، محاسبه احتمالات نهایی با استفاده از softmax هزینه بر است، زیرا مخرج کسر شامل جمع امتیازات کل کلمات vocabulary برای نرمالیز کردن می‌شود.



برای غلبه بر این دو مشکل، به جای اینکه brute force عمل کنیم، یعنی همه وزن‌ها را بخواهیم آپدیت کنیم، باید سعی کنیم تعداد وزن‌های آپدیت شده برای هر نمونه آموزشی را کاهش دهیم. تکنیک مورد نظر برای این کار negative sampling است. به جای اینکه احتمال همسایگی را پیش‌بینی کنیم، سعی می‌کنیم احتمال اینکه دو کلمه همسایه هستند یا نیستند را پیش‌بینی کنیم. به عنوان مثال، در روش عادی، احتمال P(word1| word2) را پیش‌بینی می‌کردیم، اما در روش negative sampling احتمال P(1|<word1, word2>) را حساب می‌کنیم. همچنین برای ساده‌تر شدن مسئله، به طور تصادفی تعداد محدودی از کلمات را به عنوان کلمات negative برای آپدیت وزن‌ها انتخاب می‌کنیم که k برابر با تعداد کلمات positive است. بنابراین، loss فقط برای کلمات context و negative عمل backpropagation را انجام می‌دهد.

Loss برای كلاس NegativeSamplingLoss

تابع هدف:

در اینجا، همان بردار جانمایی است و بردار زمینه است. عبارت اول تارگت‌های درست است و عبارت دوم تارگت‌های نویزی یا negative است. پس مدل سعی می‌کند برای کلماتی که context هستند ۱ را پیش‌بینی کند و برای کلمات negative که non-context هستند ۰ را پیش‌بینی کند.

A screenshot of a computer program

Description automatically generated

Trainng

ما ابتدا کورپوس را به تابع get\_batches منتقل می‌کنیم تا برایمان داده‌های آموزشی مناسب را مطابق با توضیحات پیشین تولید کند، و سپس حلقه را روی آن اجرا می‌کنیم. هر بار محاسبات را روی یک batch انجام می‌دهیم.

A screen shot of a computer program

Description automatically generated

ابتدا با استفاده از تابع forward\_input در کلاس SkipgramNeg، بردار ورودی embedding را محاسبه می‌کنیم.

A blue background with white text

Description automatically generated

بعد برای بردار خروجی embedding را با کمک تابع forward\_output از همان کلاس حساب می کنیم.

A blue background with white text

Description automatically generated

آخر بردار کلمات negative را به تابع forward\_noise می دهیم تا embedding آن را برگرداند.

A blue and white text on a dark background

Description automatically generated

حالا با دادن embedding به تابع forward کلاس NegativeSamplingLoss، loss را محاسبه می‌کنیم. سپس با انجام عمل backward propagation و optimization، وزن‌ها را به‌روزرسانی می‌کنیم.

A screen shot of a computer code

Description automatically generated

هايپرپارامترها و ساير تنظيمات :

|  |  |
| --- | --- |
| epochs | 25 |
| optimizer | optim.Adam |
| embedding\_dim | 100 |
| Learning rate | 0.003 |
| Neighborhood window size | 4 |
| number of negative samples for each sample | 4 |

توجه: برای تولید کلمات negative از توزیع unigram استفاده می‌کنیم، اما برای جلوگیری از اینکه تاثیر کلمات نادر ناچیز نباشد، از فرمول زیر برای smoothing احتمالات استفاده می‌کنیم تا شانس انتخاب کلمات نادر کمی بیشتر شود:

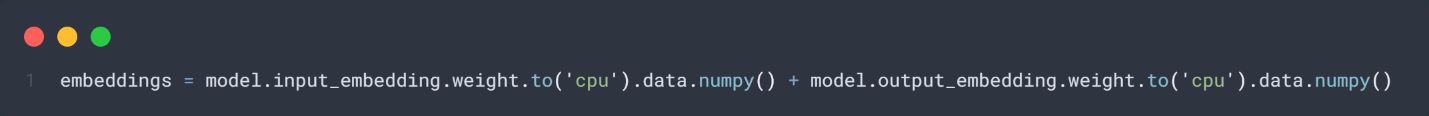
كه U(w) توزيع يكنواخت كلمات است.

همان طور که ملاحظه می‌ شود ایپاک‌های نهایی نسبت Loss کمی دارند و در مقايسه با ايپاك‌های قبلی به طور قابل ملاحظه اي كاهش يافته و اين يعني يادگيري اتفاق افتاده است.

A screenshot of a computer program

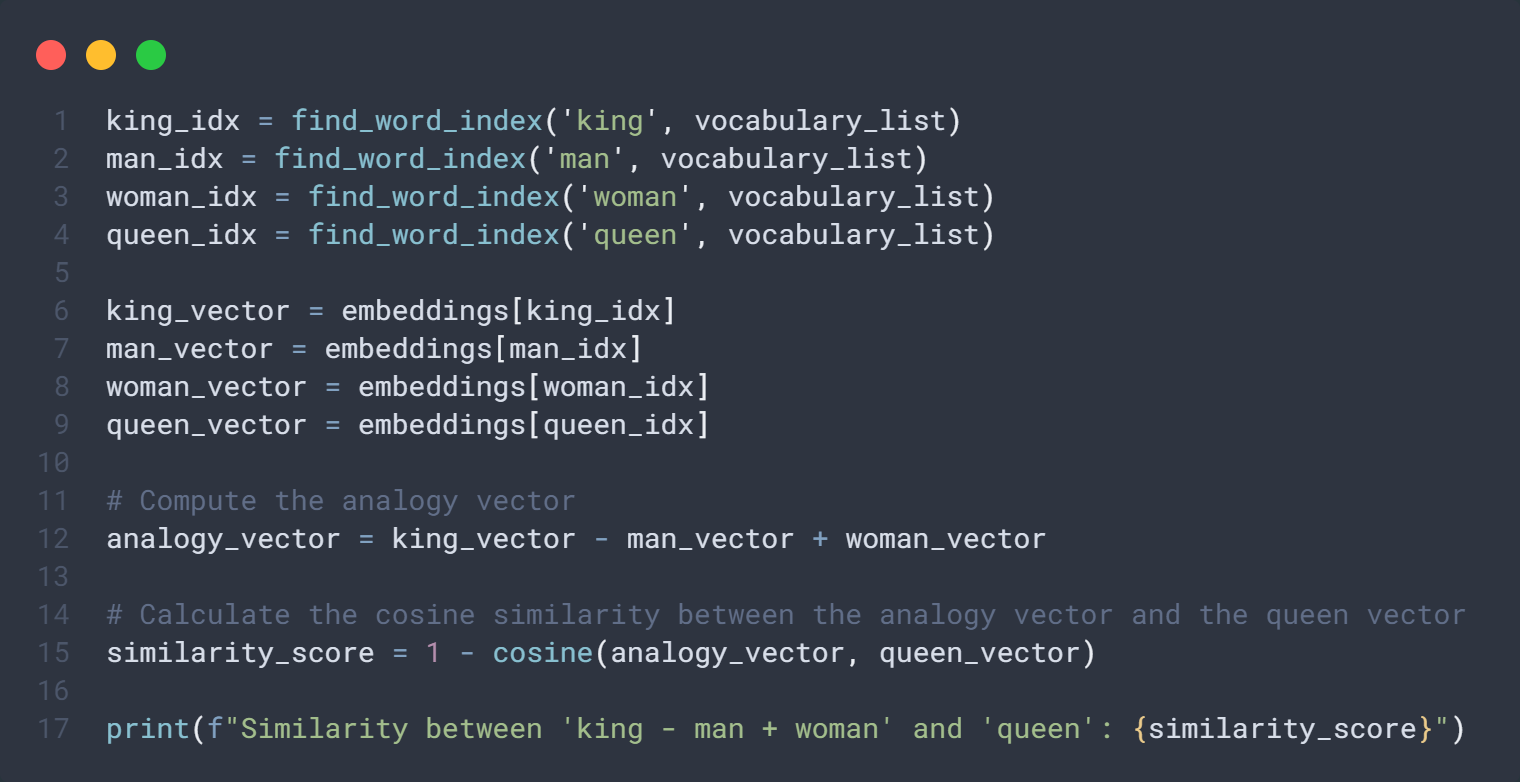
Description automatically generated

پس از آموزش مدل، دو ماتریس جانمایی و زمینه با هم جمع می‌شوند تا بردار embedding نهایی را بسازند.



## پاسخ بخش دوم

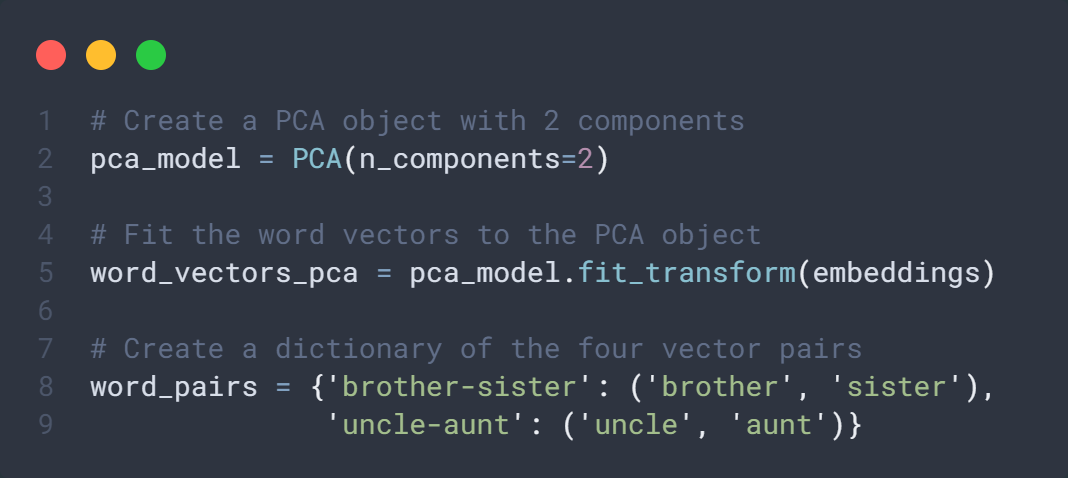
در اینجا، برای ارزیابی نحوه جاسازی کلمات از پیش آموزش دیده، یک کار قیاس کلمه انجام می‌دهیم. با گرفتن بردار برای "پادشاه"، کم کردن بردار برای "مرد" و اضافه کردن بردار برای "زن"، بر اساس فرمول مشخص بردار قیاسی را ایجاد می‌کنیم. سپس شباهت کسینوس بین این بردار قیاس و بردار "ملکه" واقعی محاسبه می‌شود و امتیازی بین 0 و 1 ارائه می‌شود. امتیاز بالا نشان می‌دهد که جاسازی‌ها با موفقیت رابطه جنسیتی را نشان می‌دهند، در حالی که نمره پایین نشان‌دهنده محدودیت‌ها در جاسازی‌ها یا چالش‌هایی با آن کار قیاسی خاص است.



امتیاز حاصل در اجراهای مختلف تغییر می‌کند، اما حدود 0.25 می‌باشد. این عدد حاصل کسینوس بین بردار قیاسی که از "پادشاه - مرد + زن" ساخته شده است و بردار تعبیه شده واقعی برای "ملکه" می‌باشد. این امتیاز نسبتاً پایین است و هر چقدر پایین‌تر باشد، نشان می‌دهد که شباهت کمتری وجود دارد. دلیل این شباهت کم ممکن است از محدودیت‌های موجود در خود embedding ها باشد که نشان می‌دهد آنها ممکن است این الگوی معنایی خاص یا رابطه جنسیتی را به درستی رمزگذاری نکنند. همچنین می‌تواند دلیل آن کم بودن اطلاعات نسبت به این کلمات باشد که کلمه queen تکرار کمی در کورپوس دارد.

## پاسخ بخش سوم

در این مرحله، طبق خواسته سوال، قصد داریم بردار تفاضل جفت کلمات مشخص شده را در یک نمودار دو بعدی نمایش دهیم.



خب، در حال حاضر هر کلمه شامل 100 مقدار است. بنابراین، آن را نمی‌توان در یک نمودار دو بعدی نمایش داد. بنابراین، لازم است که برای هر کلمه، عمل کاهش ابعاد را انجام دهیم. یکی از محبوب‌ترین روش‌ها برای این کار روش PCA است. رویکرد کلی آن این است که ترکیب‌های خطی از ویژگی‌های مختلف داده را جستجو می‌کند و ویژگی‌هایی را انتخاب می‌کند که بیشترین واریانس را در داده‌ها پوشش دهند. برای استفاده از PCA، از ماژولی با همین نام در کتابخانه Sklearn استفاده می‌کنیم و به عنوان ورودی تعداد کامپوننت‌ها را 2 می‌دهیم و سپس داده‌ها را تبدیل به فضای دو بعدی می‌کنیم. حالا برای هر جفت کلمه، مقدار embedding کاهش‌یافته آن‌ها را حساب کرده و از هم کم می‌کنیم تا بردارهای تفاضل آن‌ها بدست آید. سپس بردار تفاضل‌ها را همراه با خود کلمات در فضای دو بعدی جانمایی می‌کنیم، که در شکل زیر نتیجه آورده شده است:

A screenshot of a graph

Description automatically generated

تحلیل این تصاویر:

تحلیل این تصاویر:

احتمالاً این نمودار دقیقاً شبیه چیزی که انتظار داریم نخواهد بود، به این معنا که مثلاً فاصله پادشاه با مرد دقیقاً اندازه فاصله ملکه با زن نباشد. دو عامل می‌توانند علت این امر باشند:

* اندازه کورپوس کم است. در حالی که مدل‌های embedding که استفاده می‌شوند روی چندین میلیون کلمه آموزش می‌بینند و نباید انتظار داشته باشیم که embedding ما بر روی دیتاست خیلی کوچک، کلمات را به صورت عالی تفکیک کند.
* علاوه بر کوچک بودن کورپوس، نوع متن شکسپیر نیز ادبی است. این به این معناست که کلمات خیلی کم تکرار می‌شوند نسبت به متن‌های رایج. این باعث می‌شود مدل سخت‌ترین context مشابه را برای کلمات پیدا کند، و این مشکل کوچک بودن داده را وخیم‌تر می‌کند.

در نهایت، embedding کل کلمات را در فضای دو بعدی نمایش داده‌ایم که در ادامه آورده شده است:

A black circle with white text

Description automatically generated

از این نمودار نیز می‌توان استنباط کرد که مدل به خوبی کلمات را تفکیک نکرده و کلمات بیشتر در گوشه‌ها متمرکز شده‌اند. اگر مدل بهتر عمل می‌کرد، احتمالاً پراکندگی بیشتری داشتیم، که علت ضعف آن در موارد قبلی ذکر شده است.

A screen shot of a computer

Description automatically generated