
بسم الله الرحمن الرحیم

عنوان

گزارشی از نتایج تمرین Word Embedding

استاد راهنما

دکتر نصیری

نگارنده

فاطمه محمدی

چکیده

در میان روش‌های رایج تعبیه کلمات (word embedding)، TF-IDF و Word2Vec به‌عنوان دو رویکرد متفاوت در نمایش کلمات توسط بردار عددی برجسته هستند. در حالی که TF-IDF نمایش‌های تفسیری و مبتنی بر فراوانی ارائه می‌دهد، Word2Vec روابط معنایی عمیق‌تری را مدل‌سازی کرده و برای وظایف پیچیده‌تر NLP مؤثرتر است. چالش‌های پیش‌آمده در فرآیند آموزش و تنظیم مدل‌ها و استراتژی‌های مورد استفاده برای حل آن‌ها نیز مورد بحث قرار گرفته‌اند.

کلید واژه

خوشه بندی، بردار اعداد برای کلمات ، TF-IDF ، word2vec

مقدمه

تبدیل کلمه به بردار عددی (Word Embedding)

توانایی ماشین‌ها در پردازش، تحلیل و تولید متن ، به یکی از ارکان پیشرفت‌های پردازش زبان طبیعی (NLP) تبدیل شده است. در قلب بسیاری از کاربردهای NLP، چالش نمایش کلمات و اسناد به شکلی عددی قرار دارد که خواص زبانی و معنایی آن‌ها را به‌خوبی منعکس کند. به همین منظور، تکنیک‌های تعبیه کلمات (Word

Embedding) معرفی شدند. در میان تکنیک‌های متنوعی که برای تعبیه کلمات توسعه یافته‌اند، TF-IDF و Word2Vec به‌عنوان دو رویکرد متمایز و پرکاربرد شناخته می‌شوند. هر یک از این روش‌ها نمایانگر یک الگوی متفاوت برای فهم و کد گذاری داده‌های متنی هستند. TF-IDF یک روش سنتی و آماری است که به کلمات براساس فراوانی آن‌ها در یک سند نسبت به مجموعه اسناد، وزن‌های عددی اختصاص می‌دهد. Tf-idf کلمات را بر اساس اهمیت آماری آن‌ها در یک سند نمایش می‌دهد و برای وظایفی مانند بازیابی اطلاعات و استخراج کلمات کلیدی مؤثر است. این روش بر اهمیت کلمات نادر و خاص یک سند تأکید دارد و آن را به ابزاری قابل تفسیر و مؤثر برای وظایفی مانند بازیابی اطلاعات و طبقه‌بندی اسناد تبدیل کرده است. با این حال، TF-IDF یک بردار پراکنده ارائه می‌دهد که توانایی آن در بازتاب روابط معنایی یا بافتی عمیق‌تر بین کلمات محدود است. در مقابل، Word2Vec به‌عنوان یک روش پیش بینی کننده و متراکم، با استفاده از شبکه‌های عصبی روابط معنایی و بافتی میان کلمات را مدل‌سازی می‌کند و در وظایفی همچون شباهت معنایی و استدلال قیاسی عملکرد بهتری دارد. Word2Vec که توسط Mikolov و همکارانش در سال ۲۰۱۳ معرفی شد، تغییری به سمت تعبیه‌های متراکم و پیش‌بینی‌کننده ایجاد کرد که از طریق شبکه های عصبی آموخته می‌شوند. با آموزش روی مجموعه‌های متنی بزرگ، Word2Vec روابط معنایی و بافتی میان کلمات را مدل‌سازی می‌کند. تعبیه‌های حاصل، کلمات معنایی مشابه را در فضای برداری به یکدیگر نزدیک‌تر قرار می‌دهند و امکان انجام وظایف متنوعی مانند شباهت کلمات، تحلیل احساسات، و حل قیاس‌ها را فراهم می

کنند. برخلاف TF-IDF، تعبیه‌های Word2Vec تنها به کلمات موجود در یک سند محدود نمی‌شوند، بلکه درک خود از زبان را براساس الگوهای مشاهده‌شده درطول آموزش تعمیم می‌دهند. در این تمرین باید با استفاده از دو روش tf-idf وword2vec نظرات فیلم‌های imdb را به داده‌های عددی تبدیل کنید. در این تمرین از کتابخانه Gensim استفاده شده است.

تعاریف

تبدیل کلمه به بردار عددی (Word Embedding)

پیدایش زبان حرکتی محوری در تکامل بشریت بود. اگرچه همه گونه‌ها راه‌های ارتباطی خود را دارند، اما ما به‌عنوان انسان منحصربه‌فرد هستیم که تنها کسانی هستیم که بر ارتباطات زبان شناختی تسلط داریم. بنابراین در حالی که من می دانم که "موش" به یک جونده مودار کوچک اشاره دارد، کامپیوتر این را نمی داند.

بنابراین، هر کار با هدف پردازش زبان ابتدا باید با تمرکز بر چگونگی نمایش کلمات به طور اساسی آغاز شود.

یک روش مقدماتی یک مدل "bag of word" می باشد. که برای هر سند کلمات آن را یادداشت کرده و تعداد تکرار هر کلمه را هم یادداشت می کند و به طبقه بندی متن می پردازد این روش معایبی دارد که استفاده از این روش را محدود می کند.

این مقاله به مقایسه روش‌های **TF-IDF**و **Word2Vec**برای Emmbedding کلمات

می‌پردازد. در حالی که TF-IDF نمایش‌های تفسیری و مبتنی بر فراوانی ارائه می‌دهد، Word2Vecروابط معنایی عمیق‌تری را مدل‌سازی کرده و برای وظایف پیچیده‌تر NLP مؤثرتر است . خروجی مدل word2vec بردارهایی برای کلمات است که به عنوان ورودی به شبکه های عصبی دیگر مانند CNN داده می شود.

TF-IDF

یک معیار آماری است که اهمیت یک واژه را در یک سند نسبت به مجموعه اسناد (Corpus) ارزیابی می کند.این روش در پردازش زبان طبیعی ، بازیابی اطلاعات و کاوش متن، به کار می رود.

TF (مخفف Term Frequency) : تعداد دفعاتی که یک واژه نسبت به کل واژه های سند در یک سند ظاهر می شود.

$$TF(t) = \frac{\text{تعداد دفعات تکرار واژه } t \text{ در سند}}{\text{کل تعداد واژه های سند}}$$

IDF (مخفف Inverse Document Frequency) : نشان می دهد یک واژه چقدر در کل اسناد خاص و کمیاب است.

$$IDF(t) = \log\left(\frac{\text{کل تعداد اسناد}}{\text{تعداد اسنادی که واژه } t \text{ در آن ها ظاهر شده است}}\right)$$

N : کل تعداد اسناد

Document Frequency : تعداد اسنادی که واژه هدف در آن ها ظاهر شده است

$$TF - IDF = TF(t) \times IDF(t)$$

بردارهای کلمات حاصل از این روش بردارهایی پراکنده (Sparse) هستند که ویژگی های زیر را دارند :

- بردارهایی با اندازه های طولانی هستند (طول بردار به ۲۰۰۰۰ می رسد).
- و پراکنده (Sparse) هستند و یعنی اکثر عناصر صفر هستند و به دلیل همین ویژگی حافظه زیادی برای آموزش نیاز دارند.

Word2vec

در یادگیری زبان فقط دانستن معنی واژگان آن زبان کافی نیست بلکه معانی و ویژگی های نحوی نیز برای درک واقعی معنی از یک زبان مهم بودند. همان طور که بیان شد روش tf-idf به‌عنوان روشی که بردارهای پراکنده برای کلمات ایجاد می کند و مبتنی بر فراوانی و شمارش است، کلمات را بر اساس اهمیت آماری آن‌ها در یک سند نمایش می‌دهد و برای وظایفی مانند بازیابی اطلاعات و استخراج کلمات کلیدی مؤثر است. اما برای بررسی روابط معنایی و نحوی کلمات نمی تواند کاربردی باشد. همینطور روش Bag-of-word نیز ، در نشان دادن روابط معنایی و نحوی بین کلمات بسیار موثر نیست آن ها به صورت جداگانه در یک فضای برداری رمزگذاری می شوند. در این روش نمی توانید بگویید کلمات "love" و "like" مفهومی مشابه دارند. این جایی است که

Embeddingهای کلمه وارد می شوند. Embedding های کلمه اساساً نمایش هایی هستند که در آن مفهوم و شباهت ها با رمزگذاری در فضای برداری ثبت می شوند. در این روش کلمات مشابه بردارهای تقریباً یکسان و بازنمایی های مشابهی دارند. این روش یک تکنیک موثر برای Embedding های کلمات است. بردارهای حاصل از این روش، بردارهای متراکم هستند. از جمله ویژگی های بردار های متراکم می توان به کوتاه بودن آنها و متراکم بودن آنها اشاره کرد. منظور از متراکم بودن یعنی اکثر عناصر غیر صفر هستند و منظور از کوتاه بودن به این معنی است که طول بردار بین ۵۰-۱۰۰ است. به دلایل زیر استفاده از این بردارها کاربردی می باشد:

- استفاده از بردارهای کوتاه به عنوان ویژگی ها در مدل های یادگیری ماشین آسان است زیرا وزن های کمتری را به عنوان پارامتر در نظر می گیرد.
- این بردارها به بردارهای dense معروف هستند بهتر از شمارش های صریح تعمیم (generalize) پیدا کنند.
- این بردارها ممکن است در گرفتن مترادف بهتر عمل کنند.
- در عمل بهتر خود را نشان می دهند.

باید توجه داشت که بردارهای حاصل از روش word2vec "Static Embedding" هستند و به این معنی می باشد که برداری برای کلمه به طور ثابت تولید می کند و نه بر اساس متنی که آن کلمه در آن وجود دارد. به عنوان مثال کلمه "مهر" در متن علمی به عنوان فصل می باشد و در متنی احساسی به عنوان علاقه و مهربانی می باشد.

برای اینکه برداری برای کلمه متناسب با مفهوم متن آن تولید شود از روش های پویا مانند "Contextual Embedding" استفاده می شود.

از جمله ویژگی های روش word2vec :

- روشی پرکاربرد است.
- آموزش مدل بسیار سریع می باشد.
- کد آن در دسترس در وب سایت ها قرار دارد.

ایده روش word2vec پیش بینی احتمال شباهت کلمات به جای شمارش است. به عبارت دیگر به جای شمارش تعداد دفعات استفاده کلمه "W" در کنار کلمه "C" طبقه بندی آموزش داده می شود که یک احتمال را پیش بینی کند. "آیا احتمال دارد کلمه "W" در نزدیکی کلمه "C" باشد ؟ " یا "با چه احتمالی این دو کلمه مشابه هستند؟" در این روش نیازی به برچسب های انسانی نیست و روشی خود ناظر است یعنی روش (self-supervision) است. در این روش معیاری که برای بردار هر کلمه استفاده می شود به وضوح قابل تعیین نیست. آنچه در این روش اهمیت دارد روابط معنایی و نحوی بین کلمات است که می تواند بدون داشتن مشخصه های مشخص برای واحدهای بردار توسط این مدل تعیین شود.

معماری مدل word2vec :

Word2vec در اصل یک شبکه عصبی ۲ لایه کم عمق آموزش دیده است.

ورودی شبکه عصبی word2vec :

اگر کلمه هدف را "t" بگذاریم و کلمات همسایه و غیر همسایه را با "c" نمایش دهیم،

- کلمه هدف "t" و یک کلمه همسایه "c" را به عنوان مثال های مثبت در نظر گرفته می شوند.
- کلمه هدف "t" را با کلمات غیر همسایه "c" که به صورت تصادفی از داخل متن نمونه برداری شدند را هم به عنوان مثال های منفی در نظر گرفته می شوند.

برای هر مثال مثبت، k مثال منفی نمونه برداری بر اساس فراوانی گرفته می شود. این مثال های مثبت و منفی به عنوان داده های آموزشی به مدل داده می شوند.

در واقع ورودی مدل شبکه عصبی word2vec، کلمات متنی (Tokens) هستند که از یک مجموعه داده متنی بزرگ (Corpus) استخراج شده‌اند. اما این کلمات به صورت مستقیم به مدل داده نمی‌شوند؛ بلکه برای پردازش، ابتدا به یک فرم عددی تبدیل می‌شوند. ورودی‌های Word2Vec با روش one hot Encoding به عدد تبدیل می شود. هر کلمه به یک بردار با طول برابر با واژه نامه تبدیل می شود که فقط در یک مکان مقدار ۱ قرار دارد. این کار برای پردازش متون توسط شبکه انجام میشود . در طول آموزش، مدل از ماتریس وزن W برای تبدیل این کلمات به بردارهای معنایی (word Embedding) استفاده می کند.

لایه پنهان اول شبکه عصبی word2vec :

این لایه، یک لایه خطی (Fully Connected Layer) است که وظیفه آن تبدیل

بردار ورودی One hot Encoder به یک بردار متراکم (Dense Vector) است.

تعداد نورون های موجود در لایه پنهان برابر با طول بردار Embedding کلمات است. یعنی اگر بخواهیم همه کلمات بردارهایی به طول ۳۰۰ داشته باشند، لایه پنهان شامل ۳۰۰ نورون خواهد بود. ابعاد ماتریس وزن در لایه اول $V \times d$ است که V تعداد کلمات در واژه نامه است و d تعداد ابعاد بردار کلمه است که یک ابر پارامتر است و باید با بررسی اینکه کدام بردار معنایی بهترین مفهوم را از کلمه می رساند، تعیین شود.

لایه خروجی شبکه عصبی word2vec :

لایه خروجی حاوی احتمالات برای یک کلمه هدف (با توجه به ورودی مدل چه کلمه ای مورد انتظار است) با توجه به ورودی خاص.

تعداد نود های لایه خروجی برابر با تعداد کلمات واژگان است. تفسیر خروجی در دو روش آموزشی word2vec متفاوت است. ابتدا لازم است با دو روش آموزشی مدل word2vecآشنا شویم :

- روش CBOW
- روش Skip gram

در روش CBOW هدف پیش بینی کلمه هدف "t" با استفاده از کلمات همسایه $(C_{pos}, C_{neg1}, \dots \dots, C_{negk})$ است . در لایه خروجی احتمال تعلق هر کلمه از واژگان به کلمه هدف به دست می آید.

در روش دوم هدف پیش بینی کلمات همسایه و غیرهمسایه $(C_{pos}, C_{neg1}, \dots, C_{negk})$ برای یک کلمه هدف است. در لایه خروجی این مدل احتمال تعلق هر کلمه از واژگان به کلمات زمینه (Context Words) به دست می آید.

تابع فعالسازی در لایه آخر تابع سیگوئید می باشد تا بتواند احتمال مشابه بودن دو کلمه را از روی بردارهای آن کلمات محاسبه کند.

تابع زیان و آموزش شبکه عصبی word2vec :

در این شبکه عصبی هدف آموزش یک طبقه بندی است که به آن جفت (**w**ord,**c**ontext) داده می شود و به هر جفت یک احتمال اختصاص داده می شود.

$$P\ (+|w\ ,c)$$

$$P(-|w,c) = 1 - P(+|w,c)$$

از رگرسیون لجستیک برای آموزش طبقه بند استفاده میشود برای اینکه بتواند بین نمونه های مثبت و منفی تمایز داشته باشد و کلماتی که باهم مشابه هستند و همسایه هستند را با احتمال بالای معنایی بیان کند.

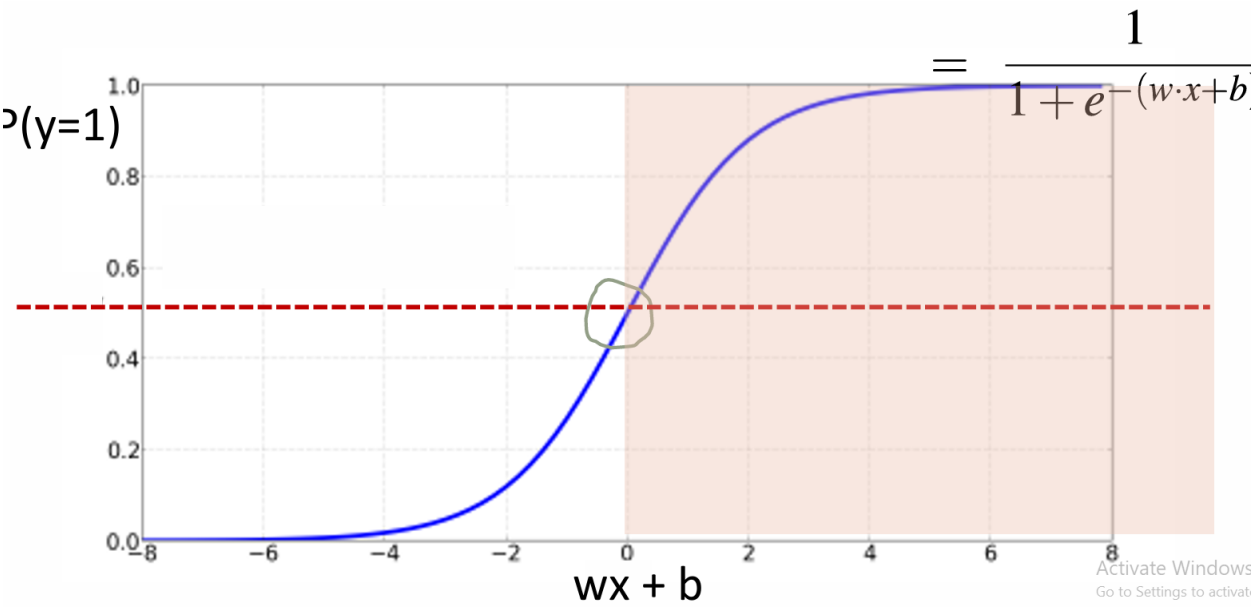
وزن های طبقه بند آموزش داده شده را به عنوان word Wmbedding برای کلمات در نظر می گیریم.

منظور از رگرسیون لجستیک استفاده از تابع لجستیک یا همان سیگوئید در لایه آخر شبکه عصبی word2vec است که احتمال را براساس شباهت بین دو کلمه بیان می کند.

تابع سیگوئید به صورت زیر می باشد:

$$Z = W.X + b$$

$$\sigma(Z) = \frac{1}{1 + e^{-Z}}$$



هدف از یادگیری در مدل word2vec این است که بردارهای کلمه را طوری تنظیم کنیم که :

- شباهت جفت کلمه target و context (W, C_{pos}) که از داده های مثبت گرفته شده اند را به حداکثر برسانید.
- شباهت جفت های (W, C_{neg}) حاصل از داده های منفی را به حداقل برسانید.

به عبارت دیگر :

- شباهت کلمه هدف را با کلمات همسایه به حداکثر برسانید.
- شباهت کلمه هدف را با کلمات غیر همسایه حداقل کنید.

تابع زیان

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w, c_{pos})\prod_{i=1}^{k} P(-|w, c_{neg_i})\right] \\
&= -\left[\log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log P(-|w, c_{neg_i})\right] \\
&= -\left[\log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log\left(1 - P(+|w, c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]
\end{aligned}
$$

توجه به این نکته ضروری است که شباهت از ضرب داخلی محاسبه می شود.

برای بیان شباهت دو کلمه به صورت احتمال یا همان تبدیل ضرب های داخلی به احتمالات باید مقدار این ضرب را از تابعی مانند سیگوئید که خروجی احتمال بین [0,1] می دهد عبور دهیم.

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$
$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$

فرمول بالا احتمال وقوع کلمه w بر اساس شباهت به کلمه $C_{1:k}$ تخمین میزند.

برای محاسبه این احتمالات نیاز به به روزرسانی وزن ها (Embedding) داریم که روابط معنایی بهتری از یک کلمه نشان دهند.

به روش های عددی Gradient Descent یا Stochastic GD وزن ها یا همان بردار کلمات را طوری تنظیم می کنیم که در کل مجموعه آموزشی :

- احتمال شباهت جفت های مثبت بیش تر شود.
- احتمال شباهت جفت های منفی کمتر شود.

روش گرادیان نزولی :

$$w^{t+1} = w^t - \gamma \frac{d}{dw} L(f(X, W), Y)$$

مشتقات تابع زیان :

از بردارهای کلمات مشتق می گیریم که همان مشتق گیری از وزن ها است

زیرا سطرهای ماتریس وزن همان بردارها برای کلمات هستند :

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w)]c_{neg_i}$$

فرمول به روزرسانی پارامترها در روش Stochastic GD :

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta \left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i} \right]
\end{aligned}
$$

دو مجموعه Embeddingآموزش می بینند :

- ماتریس Target Embedding
- ماتریس Contex Embedding

بردار کلمه مورد نظر از جمع این دو بردار به دست می آید :

بردار کلمه i ام : $W_i + C_i$

لازم است به نکات زیر توجه داشته باشد :

- اندازه بردار کلمات یک ابرپارامتر است .
- اندازه پنجره برای هر کلمه یک ابر پارامتر است.

اگر اندازه پنجره کلمات کوچک باشد کلمات از نظر نحوی شبیه به هم هستند

و اگر اندازه کلمات بزرگ باشد تشابه کلمات از نظر معنایی بررسی می شود.

با مدل word2vec می توان روابط قیاسی بین کلمات را در فضای برداری

محاسبه کرد. این مدل روابط معنایی و نحوی بین کلمات را حفظ می کند.

روش شناسی

تبدیل کلمه به بردار عددی (Word Embedding)

گام اول : فراخوانی کتابخانه های مورد نیاز

کتابخانه re برای استفاده از عبارات منظم (regular expressions) در پایتون است. این کتابخانه این امکان را می‌دهد تا الگوهای متنی پیچیده را جستجو، تطبیق و تغییر دهد . برای پاکسازی متن‌ها (مثلاً حذف علائم نگارشی و انجام پیش‌پردازش‌های متن) از عبارات منظم استفاده می‌شود.

کتابخانه sklearn.feature_extraction.text برای تبدیل نظرات فیلم‌ها به داده‌های عددی با استفاده از روش TF-IDF و استخراج کلمات پراهمیت استفاده می‌شود.

کتابخانه gensim.models.Word2Vec برای آموزش مدل Word2Vec و تولید بردارهای تعبیه‌شده (Embedding) برای کلمات استفاده می‌شود. Word2Vec یک مدل یادگیری عمیق است که کلمات را به بردارهای عددی متراکم تبدیل می‌کند.

کتابخانه nltk.tokenize.word_tokenize برای توکن‌سازی (یعنی شکستن جملات

به کلمات) استفاده می‌شود.

کتابخانه sklearn.decomposition.PCA کتابخانه PCA از scikit-learn برای انجام کاهش ابعاد(Dimensionality Reduction) استفاده می‌شود. این روش کمک می کند تا داده‌های با ابعاد بالا را به ابعاد کمتری کاهش دهید، در حالی که بیشتر اطلاعات اصلی حفظ می‌شود.

```
# import required libraries
import pandas as pd
import numpy as np
import re  # For preprocessing
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import nltk
import gensim
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

فراخوانی داده ها :

داده ها نظرات فیلم های IMDB می باشد که دارای دو ستون review و sentiment است . هدف اصلی تنها درست کردن ستون review است.

```
# Load Dataset
data = pd.read_excel('/content/IMDB Dataset.xlsx')
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

کد زیر برای دانلود داده‌های مورد نیاز از کتابخانه NLTK در پایتون است:

```
# Download necessary NLTK data
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

nltk.download('punkt') : داده‌های مربوط به توکن‌سازی متن (مثل تقسیم متن به کلمات یا جملات) را از NLTK دانلود می‌کند.

nltk.download('stopwords') : داده‌های مربوط به کلمات بی‌فایده یا stopwords (مثل "و"، "از"، "به") را دانلود می‌کند که معمولاً در پردازش زبان طبیعی برای فیلتر کردن کلمات غیرمفید استفاده می‌شوند .این داده‌ها برای پردازش متن و تجزیه و تحلیل زبان طبیعی در پروژه‌های مختلف مفید هستند.

حذف ستون sentiment :

```
#Delete the sentiment column
data = data.drop(columns=['sentiment'])
```

پاکسازی جملات موجود در ستون review :

این کار شامل این می شود که همه کلمات را با حروف کوچک نمایش دهید و علائم نگارشی را حذف کنید. در این مرحله اعداد و فاصله ها را هم حذف می کنیم.

ابتدا تابعی تعریف می کنیم که هر کدام از وظایف را انجام دهد:

```
#Clean up sentences in the review column (for example: display all words in lowercase letters, remove punctuation, etc)
def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = text.strip()  # Remove leading/trailing whitespace
    return text
```

و این تابع را بر روی ستون review اعمال می کنیم :

```
data['review'] = data['review'].apply(clean_text)
```

TF-IDF

با استفاده از کد زیر و روش tf-idf کلمات پر اهمیت این متن ها را پیدا می کنیم :

```
#Identify important words using TF-IDF
vectorizer = TfidfVectorizer(max_features=100)  # Limit to top 100 important words
tfidf_matrix = vectorizer.fit_transform(data['review'])
important_words = vectorizer.get_feature_names_out()
```

```
print("Important words using TF-IDF:")
print(important_words)
```

کلمات با اهمیت ستون review را نمایش می دهیم :

```
Important words using TF-IDF:
['about' 'after' 'all' 'also' 'an' 'and' 'any' 'are' 'as' 'at' 'bad' 'be'
 'because' 'been' 'br' 'but' 'by' 'can' 'characters' 'could' 'do' 'dont'
 'even' 'film' 'films' 'first' 'for' 'from' 'get' 'good' 'great' 'had'
 'has' 'have' 'he' 'her' 'him' 'his' 'how' 'if' 'in' 'into' 'is' 'it'
 'its' 'just' 'like' 'made' 'make' 'me' 'more' 'most' 'movie' 'movies'
 'much' 'my' 'no' 'not' 'of' 'on' 'one' 'only' 'or' 'other' 'out' 'people'
 'really' 'see' 'she' 'so' 'some' 'story' 'than' 'that' 'the' 'their'
 'them' 'then' 'there' 'they' 'think' 'this' 'time' 'to' 'too' 'up' 'very'
 'was' 'way' 'we' 'well' 'were' 'what' 'when' 'which' 'who' 'will' 'with'
 'would' 'you']
```

Word2vec

با استفاده از مدل word2vec برداری عددی به کلمات اختصاص می دهیم که روابط

معنایی آن ها را به خوبی نشان دهد : (روش آموزش CBOW را انتخاب می کنیم)

به روش زیر ابتدا جملات را به کلمه ها می شکنیم (tokenize)

```
#Create word vectors using Word2Vec
data['tokens'] = data['review'].apply(lambda x: x.split())
```

روش split()در پایتون برای جدا کردن یک رشته string به قسمت‌های کوچک‌تر (لیست از رشته‌ها) بر اساس یک جداکننده delimiter استفاده می‌شود. به طور پیش‌فرض، این جداکننده فاصله (space) است، اما می‌توان هر کاراکتر یا رشته‌ای را به عنوان جداکننده مشخص کرد.

در مرحله بعد stop word ها را حذف می کنیم :

```
# Remove stopwords in token list
def remove_stopwords(tokens):
    return [word for word in tokens if word not in stop_words]
```

```
[ ]  data['tokens'] = data['tokens'].apply(remove_stopwords)
```

در گام بعد مدل شبکه عصبی word2vec را آموزش می دهیم :

```python
# Train Word2Vec model
w2v_model = Word2Vec(sentences=data['tokens'], vector_size=100, window=5, min_count=1, workers=4)
```

نمونه هایی از بردارهای کلمات :

```python
# Example: Access the vector for a specific word
words_to_check = ['movie', 'film', 'story', 'dialogue']  # لیستی از کلمات
for word in words_to_check:
    if word in w2v_model.wv:
        print(f"Word vector for '{word}':", w2v_model.wv[word])
    else:
        print(f"'{word}' not found in the Word2Vec model.")
```

```
Word vector for 'movie': [-1.3372272   1.2986339  -0.9956534   0.71913224  0.577114    0.8727968
 -0.37680164 -0.4986355  -1.8394786  -1.8773483  -1.3814692  -0.9982465
  2.7793276   0.7759284  -0.31199268  0.44717243  2.032532   -0.18864547
 -0.03209206 -1.5501065   2.6358724   0.20080222 -0.64146453  0.52456087
  0.33340394 -0.61834496 -0.7226876   0.75147545  0.5239467   1.0980023
  1.6155298  -2.105197   -0.40812668 -2.1706116   1.3522213   1.0480683
  1.7225343  -1.5834148   0.42839634  0.3108228   2.6917288  -2.2044923
  1.1126335  -1.8726578   1.0362855  -1.0299906   1.6756582  -1.7156324
  1.088192   -1.0182091   1.4585971  -0.11779904 -0.8010898  -0.6441391
  2.6177955  -1.7571677   0.87186855  0.01019852 -0.6244865   2.4448283
 -0.42839214 -2.6380205   0.19603957  0.07228573 -0.6295387   2.0573711
 -0.4079978  -0.19291873  0.8585869   1.0776825  -1.3735367  -1.5687106
  0.40430427  2.6187842  -1.5832663  -1.8697679   1.1391933   1.3519626
  1.1042709   0.61941063 -1.4294721  -0.03588287 -1.4163284   1.1444365
  0.43545187 -2.278147    0.67774504 -0.19577412  1.1971314   0.00423948
  1.6040745   1.5845087   1.01565    -0.13404514  0.04878108  1.112011
 -0.36876377 -0.46085337 -0.5560394   0.09134947]
Word vector for 'film': [-0.98733467  2.0135598  -0.69693846  0.91582346 -0.12076109  0.6809124
 -1.0586339   0.25204465 -1.5051997  -2.5411212  -0.54574704 -0.7951614
  3.2546802   0.71743035  0.2565592   0.9188039   1.4573994   0.38101754
  0.8626335  -1.9711859   1.08076    -0.43125942 -0.45258567 -0.3608744
 -0.00580381 -0.69951254  0.08479524  0.44430175  0.67517364  1.1738749
  1.6600673  -2.155803   -0.82499385 -2.1129491  -0.40434217  0.32058322
  1.5687033  -1.7453823   0.6257698   0.68879133  1.4802511  -2.3866327
  1.9124558  -0.29206637  0.8394505  -0.09401984  2.6866622  -1.4974564
  1.4385812  -1.8066787   0.6566498   0.11500973 -0.04592198 -1.0924811
  1.6598252  -1.4336948   1.1169904  -0.29823935 -0.89906687  2.1843736
  0.40001526 -1.9098979  -0.5865167   1.5777669  -0.5513278   1.7369235
 -0.19450513 -0.01325505 -0.44341654  1.5467274  -0.94885755 -0.9240851
  0.06641573  2.7390988  -1.8367176  -0.67275685  0.15951988  0.45122984
  1.5027622   0.51597923 -1.3800237  -0.8476635  -1.235552    1.575448
 -0.6254255  -1.8916407   1.4476599  -1.3509005   1.236709    0.9742145
  1.2568458   1.4924066   1.1603532  -0.36809474 -0.05611155  1.3940035
 -0.09686805 -0.02551205  0.58784217  0.08608652]
```

```
Word vector for 'story': [ 2.4849505   2.617358    0.68700325  1.3205403   0.5116185  -1.2411877
 -1.8824111   0.19832835 -2.502436   -1.3259916  -0.6637635   0.41946006
 -0.6791559  -0.39840847  1.7568915   0.03713104 -0.43042445 -0.7946621
  0.3009108  -2.9715075  -1.4004573  -0.80404824  1.3750393  -2.6638641
 -0.23206444 -1.5634068   0.05161232  2.7668264  -0.748375    0.49990276
  0.9193335  -2.742025   -0.04854709 -0.6049364  -1.2627345   0.91293013
  0.92844987  0.5494845  -2.823941   -2.401505    1.3900261  -0.74311996
  1.2894169   0.4370479   1.1656076  -1.872689    3.9778588  -2.6362302
 -0.46387464  1.500111   -0.80552197 -0.16406652  2.588299    1.94171
  2.1010048  -3.0095575  -1.770462   -0.31452015  3.0754318   1.228573
  0.9178074  -2.154499    1.5140885  -1.1435541  -1.5678432   2.7329466
 -1.8274843   3.019993    0.03115311  0.2644655  -1.749854    2.485998
  1.7620332   3.3358788  -0.81492686 -0.65298456  1.2810214   2.979385
  1.0495358   0.34359458 -2.2699115   0.39856696 -0.64864266  0.18477684
  2.82519    -2.6456518   0.0134584  -1.2816713   0.43289253  0.92317927
  0.5511124   2.1907883   2.1440082  -0.75294864  0.02888211 -0.6911716
  0.14139263 -0.4145777  -2.0385218   1.8432202 ]
Word vector for 'dialogue': [ 1.5766842   2.6355853  -0.33193764 -0.20187838 -0.12534887 -0.9332159
 -0.09073268 -1.1063352  -1.6356136  -0.87282276 -0.1798569  -2.1300285
  0.77571994 -0.42950127  1.108322    1.0145264   0.5820089  -0.43386653
  0.47616184 -1.7200607  -0.7473144  -0.6318724  -1.2536898   0.32828277
 -0.325881    0.13358293 -1.5202427   1.212777   -3.4521792   1.078608
  0.33172157  0.28698698 -1.0335124  -1.4226773  -0.4659172   2.6936955
  0.29783845  1.6506175  -2.0603995  -1.1826636  -0.34005696 -0.79684466
 -0.04526537  2.360863    1.0356163  -0.40405804 -0.07434706 -0.15372267
  3.6898315   0.45600736  1.6713269  -0.81302     1.6953684   1.6008419
  2.9477723  -2.4721599   1.1188389  -0.4306408   1.0431519  -2.107711
 -0.91659856 -2.832098   -0.79385704  0.9079955  -0.57766205  0.6832335
  1.2817324   1.8760421  -0.14074387 -0.3399823  -0.0572213   0.36175954
  1.1895308   1.9804032   0.80388325  1.3241476  -0.19491564  1.9005613
  1.3830199   1.5961621  -1.4777695   0.2340478   0.29712546  0.22356318
  1.6243536   0.89059967  0.36806878  0.33420277  2.422958    1.8138307
 -0.5180213   1.2149197  -0.3925014  -2.3395379   1.9119395   1.5181768
 -1.8838645   0.83078194  0.21271864  0.370046  ]
```

در اخر هم شباهت و روابط قیاسی بین کلمات بررسی می شوند :

```
# Step 5: Optional additional steps
# similarity between two words
w2v_model.wv.similarity("story","film")
```

```
0.3989079
```

```
w2v_model.wv.similarity("movie","film")
```

```
0.8696597
```

```
w2v_model.wv.similarity("movie","dialogue")
```

```
0.26658416
```

```
w2v_model.wv.similarity("film","dialogue")
```

```
0.28972524
```

با استفاده از دستور زیر کلمات نزدیک به کلمه مورد نظر را با مقدار شباهت آن ها

بیان می کنیم :

```
print(w2v_model.wv.most_similar('movie'))
```

```
[('film', 0.8696596026420593), ('moviebr', 0.8165470957756042), ('filmbr', 0.7308852076530457)
```

```
print(w2v_model.wv.most_similar('dialogue'))
```

```
[('dialog', 0.9570610523223877), ('dialogues', 0.853699803352356), ('dialogs', 0.8469405770301819)
```

با استفاده از دستور زیر روابط قیاسی بین کلمات را بررسی می کنیم :

```
w2v_model.wv.most_similar(positive=["movie", "dialogue"], negative=["film"], topn=3)
```

```
[('dialog', 0.8912173509597778),
 ('dialogs', 0.7884747385978699),
 ('corny', 0.7478013634681702)]
```

و در اخر هم با روش های کاهش بعد مانند تحلیل مولفه اصلی (PCA) کلمات را در فضای دو بعدی نمایش می دهیم و تشابه کلمات را بررسی می کنیم :

فراخوانی کتابخانه های مورد نیاز برای رسم نمودار :

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

ابتدا تابعی را طراحی می کنیم تا عمل کاهش بعد را روی بردار کلمات انجام دهد به عبارت دیگر این تابع نقاط کلمات را در فضای دو بعدی با استفاده از PCA نمایش می دهد:

```python
def display_pca_scatterplot(w2v_model, words):
    """
    این تابع نقاط کلمات را در فضای دو بعدی با استفاده از PCA نمایش می‌دهد.


    words: لیستی از کلمات موردنظر برای نمایش
    """
    # بررسی وجود کلمات در مدل
    word_vectors = []
    valid_words = []
    for word in words:
        if word in w2v_model.wv:
            word_vectors.append(w2v_model.wv[word])
            valid_words.append(word)

    # بررسی خالی نبودن بردارها
    if not word_vectors:
        print("No valid words found in the model!")
        return
```

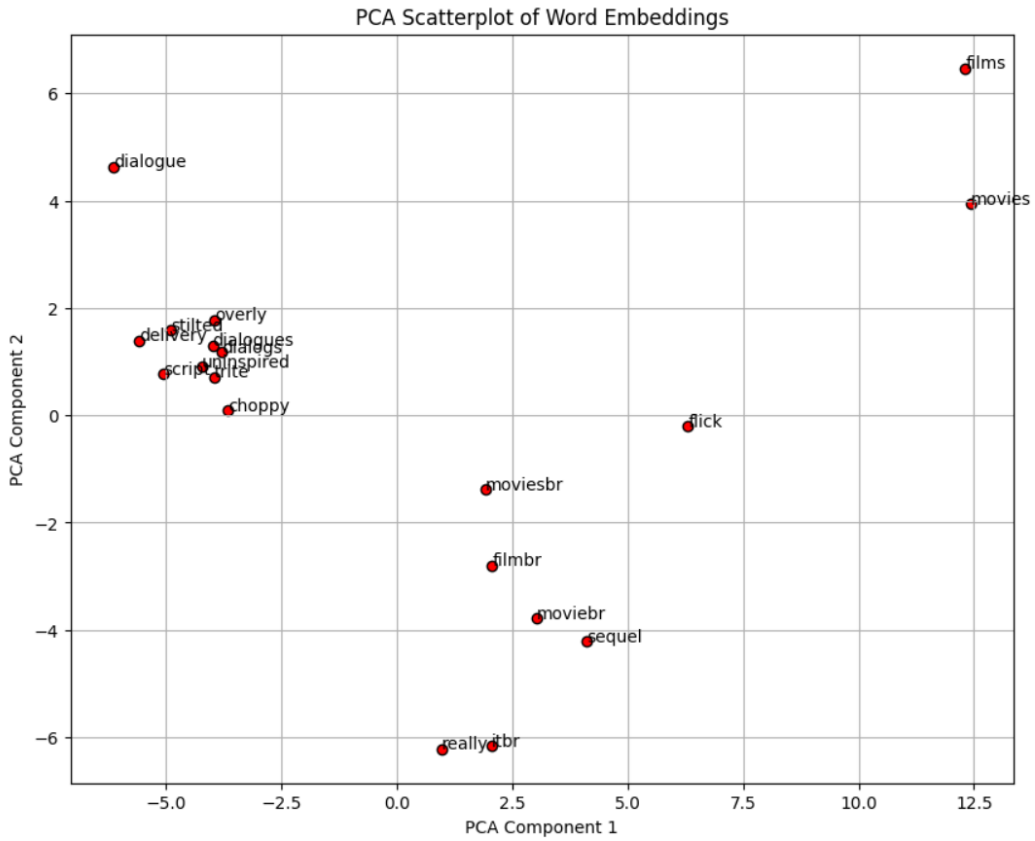
```python
# کاهش ابعاد با PCA
pca = PCA(n_components=2)
word_vectors_2d = pca.fit_transform(word_vectors)

# رسم نمودار
plt.figure(figsize=(10, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1], color='red', edgecolors='k')

# افزودن برچسب‌ها
for i, word in enumerate(valid_words):
    plt.annotate(word, xy=(word_vectors_2d[i, 0], word_vectors_2d[i, 1]), fontsize=10)

plt.title("PCA Scatterplot of Word Embeddings")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.grid()
plt.show()
```

```
[ ] display_pca_scatterplot(w2v_model,['dialogue','dialogs','dialogues','delivery','trite','script','uninspired','choppy','stilted',
                                       'overly','moviebr','filmbr','flick','movies','itbr','really','moviesbr','sequel','films'])
```



در مرحله اخر هم سعی می شود با میانگین گیری از بردار کلمات، برداری را

به عنوان نماینده جمله(متشکل از همان کلمات) به دست آوریم :

```
# Step 5: Optional additional steps
# Generate average word vectors for each review
def average_word_vectors(tokens, model, vector_size):
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if len(vectors) > 0:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(vector_size)

data['avg_word_vector'] = data['tokens'].apply(lambda x: average_word_vectors(x, w2v_model, 100))

# Display the processed data
print(data.head())
```

wv. یکی از متد های مدل word2vec است.

```
                                                 review  \
0  one of the other reviewers has mentioned that ...
1  a wonderful little production br br the filmin...
2  i thought this was a wonderful way to spend ti...
3  basically theres a family where a little boy j...
4  petter matteis love in the time of money is a ...

                                                 tokens  \
0  [one, reviewers, mentioned, watching, oz, epis...
1  [wonderful, little, production, br, br, filmin...
2  [thought, wonderful, way, spend, time, hot, su...
3  [basically, theres, family, little, boy, jake,...
4  [petter, matteis, love, time, money, visually,...

                                       avg_word_vector
0  [0.09470758, 0.113570124, -0.40050623, -0.1958...
1  [0.057546224, 0.55938303, -0.42653385, 0.17194...
2  [0.21072173, 0.20528585, -0.5248351, -0.104948...
3  [-0.0145691205, 0.28338954, -0.73835385, -0.12...
4  [-0.04892629, 0.3181804, -0.39887565, 0.272678...
```

در اخر هم بردارهای کلمات را ذخیره می کنیم تا در مدل های یادگیری عمیق

از جمله CNN از آنها برای طبقه بندی متون، و.. استفاده شوند.

تحلیل و نتایج

تبدیل کلمه به بردار عددی (Word Embedding)

پاکسازی جملات موجود در ستونreview

در این مرحله، باید جملات موجود در ستون review را پاکسازی کنید. این شامل

کارهایی مانند:

- تبدیل تمام کلمات به حروف کوچک.(lowercasing)
- حذف علائم نگارشی (مانند نقطه، ویرگول، علامت سوال، و غیره).
- حذف کلمات بی‌اهمیت یا(stop words)اختیاری، اما معمولاً مفید است.

این کار را می‌توان با استفاده از کتابخانه nltk در پایتون انجام داد:

```
[ ] #Clean up sentences in the review column (for example: display all words in lowercase letters, remove punctuation, etc)
    def clean_text(text):
        text = text.lower()  # Convert to lowercase
        text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
        text = re.sub(r'\d+', '', text)  # Remove numbers
        text = text.strip()  # Remove leading/trailing whitespace
        return text
```

```
[ ] data['review'] = data['review'].apply(clean_text)
```

TF-IDF

برای استخراج ویژگی‌ها با استفاده از TF-IDF، می‌توانید از TfidfVectorizer در کتابخانه scikit-learn استفاده کنید. این روش تعداد و اهمیت کلمات را براساس فراوانی و اهمیت آن‌ها در سندها اندازه‌گیری می‌کند. TF-IDF به کلماتی امتیاز

بیشتری می‌دهد که در یک متن خاص زیاد تکرار شده‌اند اما در کل متون کمتر

رایج هستند.

بسیاری از کلماتی که در لیست هستند، stopword یا کلمات پرکاربرد زبان

انگلیسی محسوب می‌شوند، این کلمات در بسیاری از متون تکرار می شوند و

معمولاً در تحلیل معنایی اهمیت کمی دارند :

['about', 'after', 'and', 'as', 'be', 'by', 'do', 'is', 'it', 'not', 'to'].

تعدادی کلمات محتوایی مرتبط با فیلم‌ها و بررسی فیلم‌ها دیده می‌شود، به موضوع کلی داده‌ها (احتمالاً نقد فیلم) اشاره دارد. مانند :film, films, movie, movies

کلمات مرتبط با عناصر اصلی داستان‌گویی مانند :characters, story

کلمات احساسی که در تحلیل احساسات sentiment analysis نقش کلیدی دارند. مانند :good, great, bad

وجود تعداد زیاد stopwords نشان می‌دهد که فیلتر کردن این کلمات قبل از اعمال TF-IDF انجام نشده است. حذف این کلمات می‌تواند دقت تحلیل را افزایش دهد.

شناسایی موضوعات: حضور کلماتی مثل film, story, characters نشان می‌دهد که داده‌ها احتمالاً مرتبط با نقد فیلم هستند.

تحلیل احساسات: کلماتی مثل good, great, bad می‌توانند برای تعیین لحن (مثبت یا منفی) استفاده شوند.

پیش‌پردازش بهتر: وجود کلمات غیرضروری مانند stopwords نشان می‌دهدکه پاکسازی کامل‌تر متن ضروری است.

```
Important words using TF-IDF:
['about' 'after' 'all' 'also' 'an' 'and' 'any' 'are' 'as' 'at' 'bad' 'be'
 'because' 'been' 'br' 'but' 'by' 'can' 'characters' 'could' 'do' 'dont'
 'even' 'film' 'films' 'first' 'for' 'from' 'get' 'good' 'great' 'had'
 'has' 'have' 'he' 'her' 'him' 'his' 'how' 'if' 'in' 'into' 'is' 'it'
 'its' 'just' 'like' 'made' 'make' 'me' 'more' 'most' 'movie' 'movies'
 'much' 'my' 'no' 'not' 'of' 'on' 'one' 'only' 'or' 'other' 'out' 'people'
 'really' 'see' 'she' 'so' 'some' 'story' 'than' 'that' 'the' 'their'
 'them' 'then' 'there' 'they' 'think' 'this' 'time' 'to' 'too' 'up' 'very'
 'was' 'way' 'we' 'well' 'were' 'what' 'when' 'which' 'who' 'will' 'with'
 'would' 'you']
```

Word2vec

به روش split جملات ستون review را tokenize می کنیم و یعنی آن جمله را به کلماتی می شکنیم :

ستون review شامل متونی است که نظرات کاربران را ذخیره کرده است.

x.split() متن را به لیستی از کلمات جدا می‌کند (بر اساس فاصله‌ها).

با استفاده از apply() این عمل روی تمام سطرهای ستون review انجام شده و نتیجه به صورت یک لیست از کلمات در ستون جدید به نام tokens ذخیره می‌شود.

```
#Create word vectors using Word2Vec
data['tokens'] = data['review'].apply(lambda x: x.split())
```

روش split()در پایتون برای جدا کردن یک رشته string به قسمت‌های کوچک‌تر (لیست از رشته‌ها) بر اساس یک جداکننده delimiter مثل ویرگول (Comma)

استفاده می‌شود. به طور پیش‌فرض، این جداکننده فاصله space است، اما می‌توان هر کاراکتر یا رشته‌ای را به عنوان جداکننده مشخص کرد.

آماده‌سازی داده‌ها برای NLP : تبدیل متن به لیست کلمات یک مرحله ضروری برای پردازش زبان طبیعی است. این مرحله به مدل کمک می‌کند کلمات را به صورت جداگانه تجزیه و تحلیل کند.

تحلیل دقیق‌تر: با داشتن کلمات جدا شده، می‌توانید از روش‌های دیگری مثل حذف stopwords، lemmatization، یا شمارش فراوانی استفاده کنید.

سادگی پردازش: کار با لیستی از کلمات (توکن‌ها) نسبت به متن اصلی ساده‌تر است.

بعد از آموزش مدل word2vec برای کلمات پر استفاده در ستون review برداری عددی تخصیص می دهیم. اندازه این بردار برابر با ۱۰۰ می باشد :

بردار کلمه "dialogue"

```
Word vector for 'dialogue': [ 2.8573692   0.8030517  -0.97014827  3.9758878   0.0937115   1.2472188
  0.7191432  -0.16283682 -0.47351715  1.8515446   1.0786123  -1.0404143
 -0.16416597  1.2737912  -0.98834634  2.1421194   0.25105178 -0.16894846
 -1.9574898  -2.8584619  -1.0475299  -0.2927847   0.25289214 -3.3906152
 -0.9223765  -2.2262628  -1.1628658   1.127664    0.28807852  0.8514329
  1.1058798  -1.4920447  -1.7333882  -0.42037064 -0.03855646 -0.94976383
 -4.4289103   0.12291349 -1.2666286   0.12744816 -0.17820318 -1.0767168
 -2.1743674   1.9389329   1.730452    0.11132885 -0.7943585   0.8115557
 -0.5352207  -0.17989728 -2.120814   -0.01982566  0.54789525  2.0611215
  1.4395134  -0.6990518  -0.74612147  0.96785754 -2.942578   -2.620442
 -0.88618153 -0.1241404  -0.15782042 -0.306743   -3.363915   -1.7228895
 -0.16314341  2.1050053   0.77588004 -0.6462553   1.0859882  -1.0799634
 -0.4554354  -0.2599314  -1.6516055   0.8983984  -0.9386246  -0.68309706
 -0.27400345 -2.1335053  -1.6608114  -2.6301277   0.11005753  0.20966557
  0.25849718 -4.1861935   0.78382844 -2.4862309   0.2758737  -0.7304061
  0.42133456 -0.23795608 -3.5767539   0.82588094  0.42935798  0.7856599
 -2.3803484   2.5401828  -0.8922795   1.1545347 ]
```

مدل word2vec برای بررسی روابط قیاسی و شباهت ها بین کلمات به کار می رود :

از بین شباهت های بین کلمات پرتکرار ، شباهت بین کلمه "movie" و "film" بیشتر بوده است.

```
# Step 5: Optional additional steps
# similarity between two words
w2v_model.wv.similarity("story","film")
```

```
0.3989079
```

```
w2v_model.wv.similarity("movie","film")
```

```
0.8696597
```

```
w2v_model.wv.similarity("movie","dialogue")
```

```
0.26658416
```

```
w2v_model.wv.similarity("film","dialogue")
```

```
0.28972524
```

شباهت بین دو بردار کلمه از ضرب داخلی دو بردار و COS زاویه بین دو بردار به دست می آید .

$$Cosine\ Similarity = \frac{movie\ .\ film}{\|movie\|\|film\|}$$

اگر زاویه بین دو بردار بیشتر باشد، شباهت کمتر است. شباهت کسینوسی Cosine Similarity به این صورت عمل می‌کند که هرچه زاویه بین دو بردار کوچکتر باشد (به صفر درجه نزدیک‌تر)، شباهت بین آن‌ها بیشتر است.

با دستور زیر کلماتی با بیشترین شباهت با کلمه مورد نظر را بیان می کند :

```
print(w2v_model.wv.most_similar('movie'))
```

```
[('film', 0.8696596026420593), ('moviebr', 0.8165470957756042), ('filmbr', 0.7308852076530457), ('
```

```
print(w2v_model.wv.most_similar('dialogue'))
```

```
[('dialog', 0.9570610523223877), ('dialogues', 0.853699803352356), ('dialogs', 0.8469405770301819)
```

```
print(w2v_model.wv.most_similar(positive=["movie"]))
```

```
[('film', 0.8696596026420593), ('moviebr', 0.8165470957756042), ('filmbr', 0.7308852076530457), ('
```

در خروجی بالا کلمات مشابه با کلمه "movie" با مقدار تشابه آن ها به صورت

زیر است :

کلمه "film" با مقدار شباهت "0.869"

کلمه "moviebr" با مقدار شباهت "0.8165"

کلمه "filmbr" با مقدار شباهت "0.7308"

با دستور زیر رابطه قیاسی بین کلمات را بررسی می کنیم :

$$a : a^* \quad : \quad b : b^*$$

$$\widehat{b^*} = \arg\min distance\ (x, a^* - a + b)$$

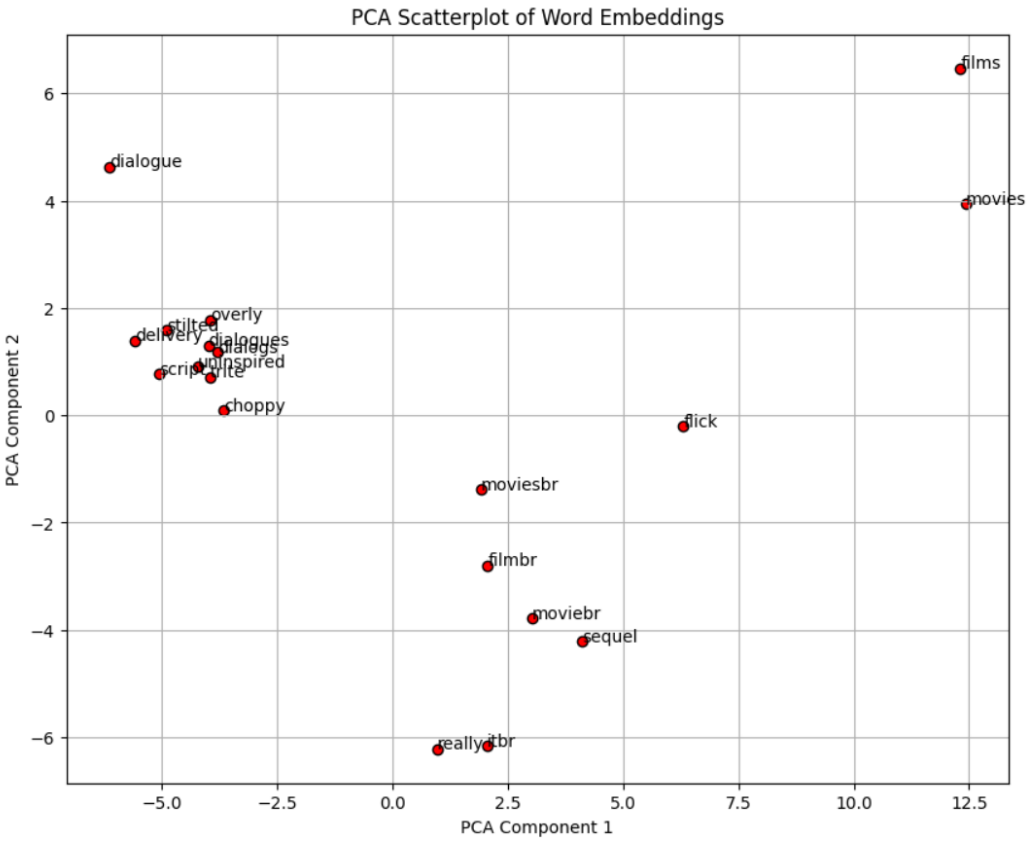
```
w2v_model.wv.most_similar(positive=["movie", "dialogue"], negative=["film"], topn=3)
```

```
[('dialog', 0.8912173509597778),
 ('dialogs', 0.7884747385978699),
 ('corny', 0.7478013634681702)]
```

با توجه به کد بالا ۳ کلمه ای که ممکن است کنار کلمه "film" قرار گیرد تا رابطه ای مانند رابطه بین "dialogue" : "movie" برقرار کند، کلمات زیر هستند و عدد کنار آنها مقدار شباهت بین این کلمات است.

```
[('dialog', 0.8912173509597778),
 ('dialogs', 0.7884747385978699),
 ('corny', 0.7478013634681702)]
```

بعد از روش کاهش بعد خطی تحلیل مولفه اصلی ،بردار کلمات ۱۰۰ بعدی را به فضای دو بعدی برای تحلیل بهتر می بریم.



طبق این نمودار کلماتی که کنار هم قرار گرفته اند بیشترین شباهت را دارند.

این کلمات مترادف های همدیگر نیز می باشند .

طبق خروجی زیر از بردارهای کلمات یک جمله میانگین گیری می شود تا بردار نماینده آن جمله به دست آید . این بردار نهایی برای فهم معنای جملات بسیار مفید می باشد.

```
                                                  review  \
0  one of the other reviewers has mentioned that ...
1  a wonderful little production br br the filmin...
2  i thought this was a wonderful way to spend ti...
3  basically theres a family where a little boy j...
4  petter matteis love in the time of money is a ...

                                                  tokens  \
0  [one, reviewers, mentioned, watching, oz, epis...
1  [wonderful, little, production, br, br, filmin...
2  [thought, wonderful, way, spend, time, hot, su...
3  [basically, theres, family, little, boy, jake,...
4  [petter, matteis, love, time, money, visually,...

                                       avg_word_vector
0  [0.09470758, 0.113570124, -0.40050623, -0.1958...
1  [0.057546224, 0.55938303, -0.42653385, 0.17194...
2  [0.21072173, 0.20528585, -0.5248351, -0.104948...
3  [-0.0145691205, 0.28338954, -0.73835385, -0.12...
4  [-0.04892629, 0.3181804, -0.39887565, 0.272678...
```

در این خروجی فقط پنج سطر اول داده شده است زیرا از دستور data.head() استفاده شده است. در این جا سه ستون وجود دارد :

ستون اول ستون review

ستون دوم مربوط به کلمات و token ها می باشد

ستون میانگین بردار کلمات که نماینده بردار جمله می باشد هم در یک ستون جداگانه قرار می گیرد.

پایان