**Natural Language Processing ( NLP )**

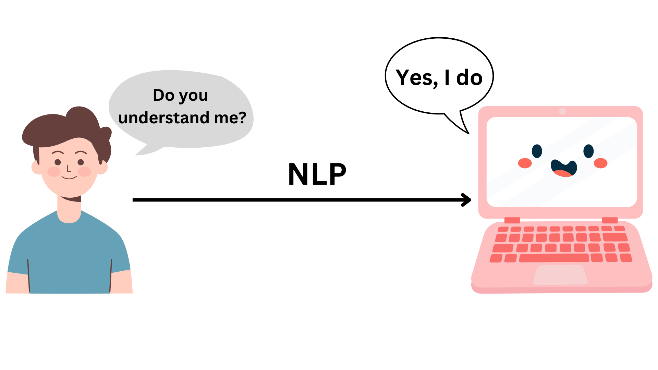
پردازش زبان طبیعی (NLP) به عنوان یک نقطه مشترک از هوش مصنوعی (AI)، زبان‌شناسی و علوم رایانه است که بر تعامل بین رایانه و انسان از طریق زبان طبیعی تمرکز دارد.

هدف نهایی NLP این است که رایانه‌ها را قادر سازد تا زبان‌های انسانی را درک ، تفسیر و تولید کنند که هم ارزشمند و هم معنادار باشد.

این فناوری زیربنای برنامه‌های مختلفی است که ما روزانه از آنها استفاده می‌کنیم،

از موتورهای جستجو و دستیارهای فعال صوتی گرفته تا خدمات ترجمه زبان و ربات‌های گفتگوی خدمات مشتری.

فن آوری های NLP به طور یکپارچه در جنبه های متعدد زندگی روزمره ما ادغام شده اند و راحتی و کارایی را افزایش می دهند.

****

**برخی از برنامه های کاربردی قابل توجه عبارتند از:**

* **موتورهای جستجو : بهبود ارتباط نتایج جستجو با درک هدف پشت پرس و جوها.**
* **دستیارهای مجازی : دستیارهای فعال صوتی مانند سیری، الکسا و دستیار گوگل را تقویت می کنند که به آنها امکان میدهد دستورات صوتی را درک کنند و به آنها پاسخ دهند.**
* **ترجمه زبان : امکان ترجمه بلادرنگ زبان ها، از بین بردن موانع ارتباطی در سراسر جهان.**
* **تجزیه و تحلیل احساسات : تجزیه و تحلیل نظرات، احساسات و عواطف از داده های متنی، که به طور گسترده در نظارت بر رسانه های اجتماعی و مدیریت برند استفاده می شود.**
* **ربات‌های چت و خدمات کلاینت :** NLP **به ربات‌های چت و راه‌حل‌های خدمات مشتری، از جمله مدل‌هایی مانند** ChatGPT**، برای ارائه پاسخ‌های خودکار و فوری به درخواست‌های کلاینت ، قدرت می‌دهد که کارایی خدمات را افزایش می‌دهد و تجربه کاربر را بهبود می‌بخشد.**

نیاز به پیش پردازش متن

قبل از پرداختن به پیچیدگی‌های مدل‌سازی و تجزیه و تحلیل در NLP، درک مرحله حیاتی که قبل از این وظایف است ضروری است: پیش‌پردازش متن.

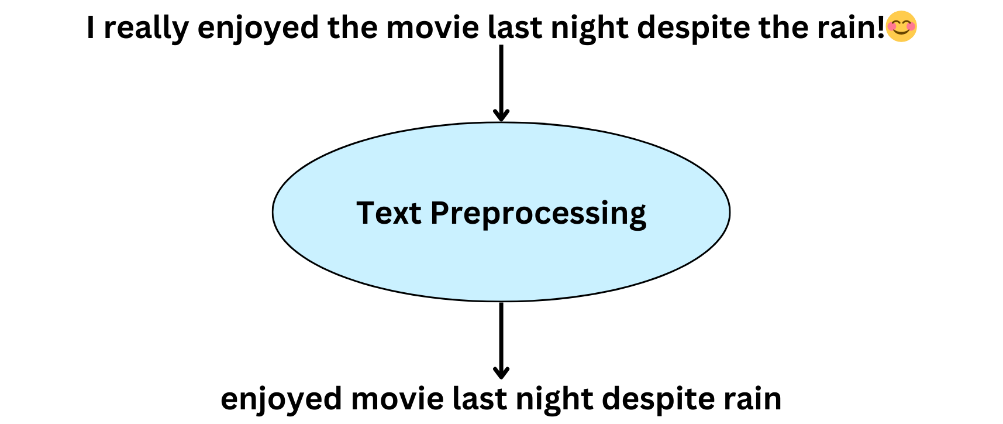
**پیش پردازش متن، فرآیند اساسی آماده سازی داده های متن خام به شکلی تمیز و استاندارد است که می تواند به طور موثر توسط مدل های NLP استفاده شود.**

**داده های متن خام اغلب آشفته و بدون ساختار هستند. ممکن است حاوی خطاها، ناسازگاری‌ها، عامیانه، اختصارات و زبان‌های مختلف باشد که درک و پردازش دقیق متن را برای مدل‌های NLP چالش برانگیز می‌کند.**

**پیش پردازش این متن خام را به شکل قابل مدیریت تری تبدیل می کند، نویز و پیچیدگی را کاهش می دهد، که به مدل ها امکان می دهد وظایفی مانند** classification**، تجزیه و تحلیل احساسات و ترجمه زبان را به طور موثرتری انجام دهند.**

تکنیک های پیش پردازش متن اصلی

* tokenization;
* cleaning and normalization;
* stop words removal;
* stemming and lemmatization;
* part-of-speech tagging.



Why NLTK?

کتابخانه NLTK (Natural Language Toolkit) یک کتابخانه پایتون برای NLP است که ما به طور فعال برای پیش پردازش متن از آن استفاده خواهیم کرد. طراحی بصری و مستندات گسترده آن هم برای مبتدیان و هم متخصصان باتجربه NLP فراهم می کند و اجرای آسان عملیات پیچیده NLP را تسهیل می کند. علاوه بر این، NLTK با مجموعه ای غنی از مجموعه داده ها و آموزش ها به عنوان یک منبع آموزشی ارزشمند عمل می کند که توسط یک جامعه بزرگ و فعال پشتیبانی می شود که به بهبود مستمر آن کمک می کند.

Tokenization

توکن ها اجزای متنی مستقل و حداقلی هستند که نحو و معنای خاصی دارند.

در نتیجه، توکن سازی فرآیند تقسیم متن به توکن است.

به عنوان مثال، یک پاراگراف متن، یک سند متنی یا یک مجموعه متن از اجزای مختلفی تشکیل شده است که می توان آنها را به جملات، عبارات و کلمات تقسیم کرد. در واقع، محبوب‌ترین روش‌های توکن‌سازی شامل توکن‌سازی جمله و کلمه است که برای شکستن یک سند متنی (یا پیکره) به جملات و هر جمله به کلمات استفاده می‌شود.

مجموعه متن (جمع: corpora) مجموعه بزرگ و ساختار یافته ای از متون است که در تحقیقات زبانشناسی و زبانشناسی محاسباتی استفاده می شود. اساساً، این مجموعه ای جامع از مطالب نوشتاری یا گفتاری است که به عنوان نمونه ای از یک زبان، گویش یا حوزه موضوعی خاص عمل می کند.

Sentence Tokenization

با نشانه گذاری جملات (sentence tokenization) شروع می کنیم. nltk تابع sent\_tokenize() را در ماژول tokenize ارائه می کند.

هدف اصلی این تابع : تقسیم متن داده شده به لیستی از جملات است.

sent\_tokenize() از pre-trained modelاستفاده می کند،

مدل یادگیری ماشین بر روی مجموعه بزرگی از متن ، آموزش داده شده است تا مرزهای بین جملات را شناسایی کند.

این مدل کلمات مختلفی را در متن در نظر می‌گیرد ، مانند علائم نگارشی (به عنوان مثال نقطه، علامت تعجب، علامت سوال)، حروف بزرگ و سایر الگوهای زبانی که معمولاً پایان یک جمله و شروع یک جمله را نشان می‌دهند.

nltk.download('punkt') : این دستور به طور خاص مدل های tokenizer "Punkt" را دانلود می کند.

با دانلود مدل های توکن ساز Punkt ، اطمینان حاصل می کنید که NLTK داده های لازم برای انجام دقیق توکن سازی جملات و کلمات را دارد.

علائم نگارشی در پایان هر جمله در جملات گنجانده شده است.

Word Tokenization

در توکن سازی کلمه، چندین روش متداول برای انجام آن وجود دارد.

ساده ترین روش استفاده از تابع split() کلاس رشته است که به طور پیش فرض از نمادهای خط جدید، فاصله ها و تب ها به عنوان جداکننده استفاده می کند.

با این حال، شما همچنین می توانید یک رشته دلخواه را به عنوان آرگومان آن ارسال کنید تا به عنوان جداکننده عمل کند.

words = text.split

برای اطمینان از اینکه توکن‌هایی مانند «This» و «this» یکسان هستند، مهم است که قبل از توکن‌سازی، رشته را به حروف کوچک تبدیل کنید.

با این حال، رویکرد انعطاف‌پذیرتر، استفاده از تابع word\_tokenize() در ماژول tokenize کتابخانه nltk است.

این تابع کلمات را بر اساس فاصله و علائم نگارشی شناسایی و جدا می کند و به طور موثر جملات را به کلمات تشکیل دهنده آنها تجزیه می کند.

word\_tokenize() ، برخلاف split() ، به طور دقیق علائم نگارشی و کاراکترهای خاص را به عنوان نشانه های جداگانه شناسایی می کند.

به درستی علامت دلار را از عدد جدا می کند و نقطه ها را به عنوان نشانه های مستقل تشخیص می دهد.

این tokenization ظریف برای بسیاری از وظایف NLP بسیار مهم است، جایی که تعیین دقیق کلمات و علائم نگارشی می تواند به طور قابل توجهی بر دقت و بینش تجزیه و تحلیل تأثیر بگذارد.

Tokenization Using Regular Expressions

در حالی که توابع word\_tokenize() و sent\_tokenize() از کتابخانه NLTK راه های مناسبی برای توکن کردن متن به کلمات و جملات ارائه می دهند، ممکن است همیشه با نیازهای خاص پردازش متن مطابقت نداشته باشند،

بنابراین یک رویکرد جایگزین توکن سازی با استفاده از عبارات منظم (regex) وجود دارد.

برای جمع بندی ، عبارات منظم دنباله ای از کاراکترها هستند که یک الگوی جستجو را تعریف می کنند.آنها را می توان برای کارهای مختلف پردازش متن، از جمله جستجو، جایگزینی، و تقسیم متن بر اساس الگوهای خاص استفاده کرد.

در زمینه توکن‌سازی ، regex امکان تعریف الگوهای سفارشی را فراهم می‌کند که می‌توانند توکن‌ها را شناسایی کنند و کنترل بیشتری بر فرآیند توکن‌سازی نسبت به توابع از پیش ساخته شده ارائه می‌دهند.

Using regexp\_tokenize()

خوشبختانه، کتابخانه NLTK شامل تابع regexp\_tokenize() در ماژول tokenize است که یک رشته را با استفاده از یک عبارت منظم به زیر رشته تبدیل می کند.

این عملکرد به ویژه زمانی مفید است که شما نیاز به tokenize کردن متن بر اساس الگوهایی دارید که توسط توکن سازهای استاندارد به خوبی مدیریت نمی شوند.

مهمترین پارامترهای regexp\_tokenize() دو پارامتر اول آن هستند : متن (رشته ای که باید نشانه گذاری شود)

و الگو (الگوی عبارت منظم).

Using RegexpTokenizer

یک رویکرد جایگزین برای توکن سازی سفارشی استفاده از کلاس RegexpTokenizer از ماژول NLTK است.

برای شروع، یک نمونه از RegexpTokenizer ایجاد کنید و الگوی عبارت منظم مورد نظر خود را به عنوان آرگومان در اختیار آن قرار دهید. این الگو نحوه توکن شدن متن را مشخص می کند.

برخلاف تابع regexp\_tokenize()، شما متنی را که باید در زمان ایجاد نمونه RegexpTokenizer توکن شود، ارائه نمی کنید.

در عوض، هنگامی که نمونه با الگوی مشخص شده ایجاد شد، از متد tokenize() آن برای اعمال توکن سازی بر روی متن خود استفاده می‌کنید، و متنی را که می‌خواهید به عنوان آرگومان تبدیل کنید به این متد ارسال می‌کنید.

این رویکرد نتایج یکسانی را به همراه دارد و در مواردی که به tokenizer برای متون مختلف نیاز دارید ، می‌تواند بهتر باشد، زیرا به شما امکان می‌دهد یک tokenize را یک بار ایجاد کنید و سپس آن را در ورودی‌های متن مختلف بدون تعریف مجدد الگو هر بار اعمال کنید.

به طور کلی، توکن‌سازی regexp امکان توکن‌سازی بسیار سفارشی‌سازی شده را فراهم می‌کند، و آن را برای مدیریت الگوهای پیچیده و قوانین توکن‌سازی خاص که به راحتی با روش‌های استاندارد مانند word\_tokenize() مدیریت نمی‌شوند، ایده‌آل می‌کند.

Removing Stop Words

در NLP، فرآیند حذف کلمات توقف ( کلمات اضافی ) ، یک مرحله مهم در پیش پردازش متن است.

کلمات توقف کلمات رایجی هستند که معمولاً به معنای یک جمله کمک نمی کنند، حداقل برای اهداف اکثر تحلیل ها و الگوریتم ها.

این شامل کلماتی مانند "the"، "is"، "in" و "on" است.

­­­­ Stop Wordsمعمولاً پس از توکن‌سازی برای وظایف NLP ، مانند تجزیه و تحلیل احساسات ، مدل‌سازی موضوع یا استخراج کلمه کلیدی، فیلتر می‌شوند. منطق پشت حذف Stop Words ، کاهش اندازه مجموعه داده ، در نتیجه بهبود کارایی محاسباتی ، افزایش ارتباط تجزیه و تحلیل با تمرکز بر کلماتی است که دارای معنی مهم هستند.

برای آسان‌تر کردن کارها ، nltk فهرستی جامع از کلمات توقف در چندین زبان ارائه می‌کند که می‌توان به راحتی به آن دسترسی داشت و برای فیلتر کردن کلمات توقف از داده‌های متنی استفاده کرد.

*from nltk.corpus import stopwords*

*nltk.download('stopwords')*

*stop\_words = set(stopwords.words('english'))*

نکته DS : تبدیل لیست دریافتی از stopword ها به یک مجموعه، کارایی جستجوها را بهبود می بخشد، زیرا بررسی عضویت در یک مجموعه سریعتر از یک لیست است.

ابتدا باید کلمات توقف را دانلود کرده و توکن سازی انجام دهیم. مرحله بعدی استفاده از list comprehension برای ایجاد لیستی که فقط حاوی نشانه هایی است که کلمات توقف نیستند.

word.lower() در قسمت if برای تبدیل هر کلمه (token) به حروف کوچک ضروری است، زیرا nltk حاوی کلمات توقف منحصرا با حروف کوچک است.

از طرف دیگر، می‌توانیم از یک حلقه for معمولی به جای بررسی فهرست استفاده کنیم، اما استفاده از list comprehension در اینجا کارآمدتر و مختصرتر است.

 اجزای اصلی یک مجموعه متن :  Documents

document یک قطعه متن جداگانه در یک مجموعه است،

به عنوان مثال، یک ایمیل در مجموعه ای از ایمیل ها.

اساساً، هر text corpus مجموعه ای از اسناد است، بنابراین پیش پردازش مجموعه به معنای پیش پردازش هر یک از اسناد است.

Loading the Corpus

قبلاً، بدنه متنی خود را به عنوان متغیرهای رشته ای داشتیم. با این حال، در سناریوهای دنیای واقعی، یک پیکره متنی اغلب در فایل‌های TXT برای داده‌های صرفا متنی یا در فایل‌های CSV با ستون‌های متعدد هنگامی که داده‌های اضافی با متن مرتبط است، ذخیره می‌شود. فایل‌های CSV یا فایل‌های TXT را میتوانیم استفاده کنیم ، به طوریکه هر سند از یک خط جدید شروع می‌شود.

بنابراین، ما از تابع read\_csv() از کتابخانه pandas برای بارگذاری یک مجموعه متن از یک فایل استفاده خواهیم کرد.

Preprocessing the Corpus

برای پیش پردازش corpus ، ابتدا یک تابع برای پیش پردازش هر یک از اسناد ایجاد می کنیم

سپس این تابع را برای هر یک از سندها در DataFrame خود اعمال می کنیم و یک ستون با اسناد تمیز شده ایجاد کنیم

حال مجموعه متنی ما با موفقیت پیش پردازش شده است،

doc = ' '.join(filtered\_tokens) : توکن های فیلتر شده را دوباره به یک رشته که با فاصله از هم جدا شده اند، می پیوندد.



Stemming (ریشه)

Stemming یک تکنیک عادی سازی متن است که به طور گسترده در NLP برای کاهش کلمات به شکل ریشه آنها استفاده می شود.

هدف این است که کلمات مشابه با عطف های مختلف را به یک شکل پایه مشترک تبدیل کنیم و معنای اصلی کلمه را بدست آوریم.

به عبارت دقیق تر، stemming شامل حذف پسوندها از کلمات برای به دست آوردن شکل ریشه آنها است که به عنوان ریشه شناخته می شود.

به عنوان مثال : ریشه های "ran"و "runner" همگی "run" هستند. همانطور که در بالا ذکر شد، هدف از stemming ساده کردن تجزیه و تحلیل با در نظر گرفتن کلمات مشابه به عنوان یک موجودیت است که در نهایت کارایی و اثربخشی وظایف مختلف NLP را افزایش می دهد.

Stemming with NLTK

NLTK الگوریتم های بنیادی مختلفی را ارائه می دهد که محبوب ترین آنها Porter Stemmer و Lancaster Stemmer است.

این الگوریتم‌ها قوانین خاصی را برای حذف الحاقات و استخراج ریشه یک کلمه اعمال می‌کنند.

Porter Stemmer و Lancaster Stemmer نتایج نسبتا متفاوتی تولید می کنند. این به این دلیل است که لنکستر استمر حدود دو برابر قوانین بیشتری نسبت پورتر استمر دارد و یکی از "تهاجمی ترین" استمرها است.

به طور کلی، Porter Stemmer محبوب ترین گزینه است که نتایج معنی داری را نسبت به Lancaster Stemmer، که تمایل به بیش از حد کلمات را دارد، تولید می کند.

*porter\_stemmed\_tokens = [porter\_stemmer.stem(token) for token in tokens]*

*lancaster\_stemmed\_tokens = [lancaster\_stemmer.stem(token) for token in tokens]*

Lemmatization یک تکنیک عادی سازی متن است که در NLP برای کاهش کلمات به شکل پایه یا ریشه آنها استفاده می شود که به عنوانlemma شناخته می شود.