**فصل 6. کار با متن**

**6.0 مقدمه**

داده‌های متنی بدون ساختار، مانند محتوای یک کتاب یا یک توییت، همزمان یکی از جالب‌ترین منابع ویژگی‌ها و هم یکی از پیچیده‌ترین منابع هستند. در این فصل،‌‌ به استراتژی های تبدیل متن به ویژگی‌های حاوی اطلاعات خواهیم پرداخت. این بدان معنا نیست که دستورالعمل‌های مورد بررسی در اینجا، جامع هستند. رشته‌های علمی تخصصی وجود دارند که بر روی مدیریت این داده‌ها و انواع مشابه تمرکز دارند و محتوای تمام تکنیک‌های آن‌ها می‌تواند یک کتابخانه کوچک را پر کند. با این وجود، تکنیک‌های متداولی وجود دارند و داشتن دانش در این زمینه، ابزارهای ارزشمندی را به جعبه ابزار پیش‌پردازش ما اضافه خواهد کرد.

**6.1 پاک کردن متن**

**مسئله**

شما چند داده متنی بدون ساختار دارید و می‌خواهید برخی از عملیات‌های پایه‌ای پاک‌سازی را انجام دهید.

**راه‌حل**

بیشتر عملیات‌های پایه‌ای پاک‌سازی متن باید فقط از عملیات‌های اصلی رشته‌های پایتون مانند strip، replace و split استفاده کنند.



ما همچنین یک تابع تبدیل سفارشی ایجاد و استفاده می‌کنیم:



در نهایت، می‌توانیم از عبارات منظم (Regular Expressions) استفاده کنیم تا عملیات‌های قدرتمندی روی رشته‌ها انجام دهیم.



**بحث**

بیشتر داده‌های متنی قبل از استفاده برای ساخت ویژگی‌ها نیاز به پاک‌سازی دارند. بیشتر عملیات پایه‌ای پاک‌سازی متن می‌توانند با استفاده از عملیات استاندارد رشته‌های پایتون انجام شوند. در جهان واقعی، احتمالاً یک تابع پاک‌سازی سفارشی (مانند تابع تبدیل حروف به حروف بزرگ) تعریف می‌کنیم که ترکیبی از برخی از کارهای پاک‌سازی را انجام داده و آن را بر روی داده‌های متنی اعمال می‌کنیم.

**منابع بیشتر**

* [Beginners Tutorial for Regular Expressions in Python](http://bit.ly/2HTGZuu)

**6.2 تجزیه و پاک‌سازی HTML**

**مسئله**

شما داده‌های متنی دارید که شامل عناصر HTML هستند و می‌خواهید فقط متن را استخراج کنید.

**راه حل**

از مجموعه گسترده‌ای از گزینه‌های Beautiful Soup برای تجزیه و استخراج از HTML استفاده کنید.



**بحث**

به‌رغم نام عجیب، Beautiful Soup یک کتابخانه قدرتمند پایتون است که برای [[1]](#footnote-1)scraping اطلاعات از صفحات وب در زبان برنامه‌نویسی پایتون طراحی شده است. به طور معمول، Beautiful Soup برای اسکریپت کردن وبسایت‌های زنده استفاده می‌شود، اما ما می‌توانیم به راحتی از آن برای استخراج داده‌های متنی تعبیه ‌شده در HTML استفاده کنیم. طیف کامل عملیات Beautiful Soup فراتر از محدوده این کتاب است، اما حتی روش های معدود مورد استفاده در راه‌حل ما نشان می‌دهد که چگونه می‌توانیم کدهای HTML را برای استخراج داده‌های مورد نیازمان به راحتی تجزیه و تحلیل کنیم.

**منابع بیشتر**

* [Beautiful Soup](http://bit.ly/2pwZcYs)

**6.3 حذف علائم نگارشی (علامت‌گذاری)**

**مسئله**

شما یک ویژگی داده های متنی دارید و می خواهید علائم نگارشی را حذف کنید.

**راه‌حل**

یک تابع تعریف کنید که از *translate* با یک دیکشنری از کاراکترهای علائم نگارشی استفاده کند.



**بحث**

تابع *translate* یک متد پرطرفدار در پایتون است که به دلیل سرعت بالای آن شناخته ‌شده است. در راه حل ما، ابتدا یک فرهنگ لغت، punctuation، ایجاد کردیم که تمام کاراکترهای علائم نگارشی مطابق با یونیکد را به عنوان کلیدها و None به عنوان مقادیر آن داشت. سپس تمام کاراکترهای رشته‌ی متنی که در punctuation قرار دارند، به None ترجمه شدند که عملاً آنها را حذف می‌کند. روش‌های خواناتری برای حذف علائم نگارشی وجود دارند، اما این راه‌حل به نوعی هکی دارای مزیت سرعت بیشتر نسبت به گزینه‌های جایگزین‌ است.

در حذف علائم نگارشی باید به این نکته توجه داشته باشیم که نشانگرها حاوی اطلاعات هستند (مثلاً "صحیح هستید؟" در مقابل "صحیح هستید!"). حذف علائم نگارشی اغلب یک کار ضروری اما ناخواسته برای ایجاد ویژگی‌ها است؛ با این حال، اگر علائم نگارشی اهمیت دارند، باید مطمئن شویم که این موضوع را در نظر می‌گیریم.

**6.4 جداسازی کلمات متن**

**مسئله**

شما متنی دارید و می‌خواهید آن را به کلمات جدا کنید.

**راه‌حل**

ابزارکی با نام Natural Language Toolkit (NLTK) برای زبان برنامه‌نویسی پایتون وجود دارد که مجموعه‌ای قدرتمند از عملیات‌های مرتبط با متن را ارائه می‌دهد، از جمله جداسازی (tokenizing) کلمات. 

ما همچنین می‌توانیم متن را به جملات تقسیم کنیم (توکن‌های جملات).

**بحث**

توکن‌بندی، به ویژه توکن‌بندی کلمات، یک وظیفه رایج پس از پیش‌پردازش داده‌های متنی است زیرا این اولین مرحله در فرایند تبدیل متن به داده‌هایی است که از آنها برای ساخت ویژگی‌های مفید استفاده خواهیم کرد.

**6.5 حذف کلمات توقفی**  **(Stop Words)**

**مسئله**

با توجه به داده های متنی نشانه‌گذاری شده، می خواهید کلمات بسیار متداول (مانند a، is، of، on) که حاوی ارزش اطلاعاتی کمی هستند را حذف کنید.

**راه حل**

از لیست کلمات توقفی (stopwords) در ابزار NLTK استفاده کنید.



**بحث**

اصطلاح «کلمات توقفی» (Stop Words) ممکن است به هر مجموعه‌ای از کلمات که می‌خواهیم قبل از پردازش حذف کنیم، اشاره کند، اما اغلب به کلمات بسیار متداولی اشاره دارد که خودشان ارزش اطلاعاتی کمی دارند. NLTK یک لیست از کلمات توقفی متداول دارد که می‌توانیم برای یافتن و حذف کلمات توقفی در توکن‌های کلماتمان استفاده کنیم.



توجه داشته باشید که کتابخانه NLTK فرض می‌کند که کلمات جداسازی ‌شده (tokenized) همگی با حروف کوچک نوشته شده‌اند.

**6.6 کلمات ریشه ای (Stemming Words)**

**مسئله**

شما کلمات را نشانه‌گذاری کرده‌اید و می‌خواهید آنها را به شکل ریشه‌ای خود تبدیل کنید.

**راه حل**

از PorterStemmer در NLTK استفاده کنید.

**بحث**

عمل stem کردن به معنای تقلیل یک کلمه به ریشه‌ی آن است، با شناسایی و حذف اشتقاق‌ها (مانند اسم‌ها) و نگه‌داشتن معنای اصلی کلمه. به عنوان مثال، هر دو کلمه "tradition" و "traditional" ریشه‌ی "tradit" را دارند که نشان‌دهنده این است که این دو کلمه از یک مفهوم عمومی نشئت می‌گیرند. با استفاده از عمل stem کردن، متن را به چیزی تبدیل می‌کنیم که کمتر قابل خواندن است، اما نزدیک‌تر به معنای اصلی آن می‌شود و از این رو برای مقایسه‌ی مشاهدات مختلف مناسب‌تر می‌شود. ابزار PorterStemmer در ابزار NLTK از الگوریتم stem کردن Porter رایج استفاده می‌کند تا پسوندهای متداول را حذف یا جایگزین کند و به ریشه‌ی کلمات برساند.

**منابع بیشتر**

* [Porter Stemming Algorithm](http://bit.ly/2FB5ZZb)

**6.7 برچسب گذاری قسمت‌های گفتار**

**مسئله**

شما داده‌های متنی دارید و می‌خواهید هر کلمه یا نویسه را با برچسبِ بخشِ گفتار خود برچسب‌گذاری کنید.

**راه حل**

از برچسب‌گذار آماده‌ی بخش‌های گفتار NLTK استفاده کنید.

خروجی یک لیست از تاپل‌ها است که شامل کلمه و برچسب قسمت کلامی آن است. NLTK از برچسب‌های قسمت کلامی Penn Treebank استفاده می‌کند. برخی از مثال‌های برچسب‌های قسمت کلامی Penn Treebank عبارت‌اند از:

|  |  |
| --- | --- |
| Part of speech | Tag |
| Proper noun, singular | NNP |
| Noun, singular or mass | NN |
| Adverb | RB |
| Verb, Past tense | VBD |
| Verb, gerund or present Participle | VBG |
| Adjective | JJ |
| Personal Pronoun | PRP |

بعد از اینکه متن برچسب‌گذاری شده است، می‌توانیم از برچسب‌ها برای پیدا کردن بخش‌های معینی از کلام استفاده کنیم. به عنوان مثال، می‌توانیم همه‌ی اسامی را پیدا کنیم.



یک موقعیت واقعی‌تر این است که ما داده‌هایی داریم که در آن هر مشاهده حاوی یک توییت است و می‌خواهیم آن جملات را به ویژگی‌هایی برای بخش‌های مختلف گفتار تبدیل کنیم (به عنوان مثال، یک ویژگی با 1 در صورت وجود یک اسم خاص، و در غیر اینصورت0):



با استفاده از *classes\_* می توانیم ببینیم که هر ویژگی یک تگ بخشی از گفتار است:



**بحث**

اگر متن ما به زبان انگلیسی باشد و بر روی موضوعات خاصی نباشد (مانند پزشکی)، ساده‌ترین راه‌حل استفاده از برچسب‌گذاری آماده قسمت‌های گفتار (pre-trained parts-of-speech tagger) در ابزار NLTK است. اما اگر برچسب‌گذاری آماده قسمت‌های گفتار بسیار دقیق نباشد، NLTK ما را قادر می‌سازد که خود برچسب‌گذاری‌کننده (tagger) خود را آموزش دهیم. اما مشکل اصلی در آموزش برچسب‌گذاری‌کننده این است که به یک مجموعه بزرگ از متن‌ها نیاز داریم که برچسب قسمت‌های کلامی کلمات در آن‌ها شناخته شده باشد. ساخت این مجموعه متن‌های برچسب‌گذاری‌شده به واضح است کاری زمان‌بر است و احتمالاً آخرین گزینه خواهد بود.

در کل، اگر مجموعه‌ای از متن برچسب‌گذاری‌شده داشته باشیم و می‌خواهیم یک برچسب‌گذار را آموزش دهیم، مثالی از این کار به شرح زیر است. مجموعه‌داده‌ای که ما از آن استفاده می‌کنیم، Corpus Brown است، یکی از محبوب‌ترین منابع متن برچسب‌گذاری‌شده. در اینجا از یک برچسب‌گذار n-gram بازگشتی استفاده می‌کنیم، جایی که n تعداد کلمات قبلی است که مدنظر ما هنگام پیش‌بینی برچسب بخش گفتار یک کلمه قرار می‌گیرند. ابتدا با استفاده از TrigramTagger کلمات قبلی دو کلمه را در نظر می‌گیریم؛ اگر دو کلمه وجود نداشته باشد، به BigramTagger بازگشت می‌کنیم و فقط برچسب کلمه قبلی را در نظر می‌گیریم؛ و در نهایت اگر این هم موجود نباشد، تنها به کلمه خود نگاه می‌کنیم و از UnigramTagger استفاده می‌کنیم. برای ارزیابی دقت برچسب‌گذاری‌کننده، متن را به دو بخش تقسیم می‌کنیم، برچسب‌گذاری‌کننده را بر روی یک بخش آموزش می‌دهیم و میزان دقت پیش‌بینی برچسب‌ها را بر روی بخش دیگر آزمایش می‌کنیم.



**منابع بیشتر**

* [Penn Treebank](http://bit.ly/2HROPo5)
* [Brown Corpus](http://bit.ly/2FzgMTx)

**6.8 انجام شناسایی موجودیت با نام**

**مسئله**

می‌خواهید شناسایی موجودیت نام‌گذاری شده را در متن (مانند «شخص»، «دولت» و غیره) انجام دهید.

**راه‌حل**

برای استخراج موجودیت‌ها از متن، از خط لوله[[2]](#footnote-2) و مدل‌های پیش‌فرض شناسایی موجودیت با نام spaCy استفاده کنید:



**بحث**

شناسایی موجودیت نامی، فرآیند شناسایی موجودیت‌های خاص از متن است. ابزارهایی مانند spaCy ارائه دهنده‌ی روش‌هایی مانند خطوط لوله از پیش پیکربندی شده و حتی مدل‌های یادگیری ماشینی از پیش آموزش دیده یا تنظیم شده هستند که به راحتی می‌توانند این اشیاء[[3]](#footnote-3) را شناسایی کنند. در این مورد، ما از spaCy برای شناسایی یک شخص ("Elon Musk")، سازمان ("Twitter") و ارزش پولی ("21B") از متن خام استفاده می‌کنیم. با استفاده از این اطلاعات، می‌توانیم اطلاعات ساختاریافته را از داده‌های متنی بدون ساختار استخراج کنیم. سپس این اطلاعات را می‌توان در مدل‌های یادگیری ماشین پایین‌رونده[[4]](#footnote-4) یا تجزیه و تحلیل داده‌ها استفاده کرد.

**6.9 رمزگذاری متن به عنوان یک کیسه از کلمات (Bag of Words)**

**مسئله**

شما داده‌های متنی دارید و می‌خواهید مجموعه‌ای از ویژگی‌ها را ایجاد کنید که تعداد دفعاتی را که متن مشاهده حاوی یک کلمه خاص است را نشان می‌دهد.

**راه حل**

از CountVectorizer در scikit-learn استفاده کنید.



این خروجی یک آرایه فضایی (sparse array) است که معمولاً در مواردی که متن بسیار زیادی داریم، لازم است. اما در مثال ساده‌ای که ما داریم، می‌توانیم از تابع toarray استفاده کنیم تا ماتریس شمارش کلمات برای هر مشاهده را ببینیم.



می‌توانیم از متد vocabulary\_ استفاده کنیم تا کلمه مرتبط با هر ویژگی را مشاهده کنیم.

ممکن است این موضوع گیج‌کننده باشد، بنابراین به‌خاطر وضوح بیشتر، ماتریس ویژگی به‌صورت زیر با نام کلمات به‌عنوان نام‌های ستون‌ها است به نمایش درآمده است (هر ردیف یک مشاهده است).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| sweden | love | is | germany | brazil | both | best | Beats |
| 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

**بحث**

یکی از متداول‌ترین روش‌های تبدیل متن به ویژگی، استفاده از مدل کیسه کلمات است. مدل‌های کیسه‌ای کلمات برای هر کلمه منحص به فرد در داده‌های متنی یک ویژگی تولید می‌کنند که هر ویژگی حاوی تعداد زیادی از موارد در مشاهدات است. به عنوان مثال، در راه حل ما ، جمله I love Brazil. Brazil! 2مقدار در ویژگی *"brazil"دارد ،* زیرا کلمه برزیل دو بار ظاهر می شود.

داده های متنی در راه حل ما عمداً کوچک بود. در دنیای واقعی، تنها مشاهده داده های متنی می تواند شامل محتویات یک کتاب کامل باشد! از آنجایی که مدل کیسه کلمات ما یک ویژگی برای هر کلمه منحصر به فرد در داده ها ایجاد می کند، ماتریس حاصل می تواند هزاران ویژگی داشته باشد. این بدان معنی است که اندازه ماتریس گاهی اوقات می تواند در حافظه بسیار بزرگ شود. با این حال، خوشبختانه ما می‌توانیم از ویژگی مشترک ماتریس‌های ویژگی کیسه‌ای کلمات استفاده کنیم تا میزان داده‌هایی که نیاز به ذخیره‌سازی داریم کاهش یابد.

اکثر کلمات احتمالاً در بیشتر مشاهدات ظاهر نمی‌شوند، و بنابراین ماتریس‌های ویژگی کیسه از کلمات بیشتر مقادیر 0 را دارند. ما این نوع ماتریس‌ها را "sparse" می‌نامیم. به جای ذخیره‌سازی همه مقادیر ماتریس، ما تنها مقادیر غیرصفر را ذخیره می‌کنیم و سپس فرض می‌کنیم که سایر مقادیر برابر با 0 هستند. این باعث صرفه‌جویی در حافظه می‌شود هنگامی که ما ماتریس‌های ویژگی بزرگ داریم. یکی از ویژگی‌های خوب CountVectorizer این است که خروجی به صورت پیش‌فرض یک ماتریس sparse است.

CountVectorizer دارای چندین پارامتر مفید است که تولید ماتریس‌های ویژگی کیسه از کلمات را آسان می‌کند. اولین پارامتر، به صورت پیش‌فرض هر ویژگی یک کلمه است، اما این باید همچنین می‌تواند مجموعه‌ای از دو کلمه (2-gram) یا حتی سه کلمه (3-gram) باشد. ngram\_range حداقل و حداکثر اندازه n-gram ما را تعیین می کند. به عنوان مثال، (2،3) همه 2-gram و 3-gram را برمی‌گرداند. دوم، ما می‌توانیم به راحتی کلمات پر اطلاعات کم را با استفاده از stop\_words با یک لیست داخلی یا لیست سفارشی حذف کنیم. در نهایت، ما می‌توانیم کلمات یا عباراتی که می‌خواهیم مورد بررسی قرار دهیم را با استفاده از vocabulary محدود کنیم. به عنوان مثال، ما می‌توانیم یک ماتریس ویژگی کیسه از کلمات برای تنها وقوع نام های کشور ایجاد کنیم.



**منابع بیشتر**

* [*n*-gram](https://en.wikipedia.org/wiki/N-gram)
* [Bag of Words Meets Bags of Popcorn](http://bit.ly/2HRba5v)

**6.10 اهمیت وزن‌دهی به کلمات**

**مسئله**

شما می‌خواهید یک کیسه از کلمات داشته باشید اما با وزن‌دهی به کلمات بر اساس اهمیت آنها برای یک مشاهده.

**راه حل**

برای مقایسه فراوانی کلمه در یک سند (توییت، بررسی فیلم، متن سخنرانی و غیره) با فراوانی کلمه در سایر اسناد از روش معکوس فراوانی فراوانی معکوس ترم‌ها (TF-IDF) استفاده کنید. scikit-learn با استفاده از TfidfVectorizer این روند را آسان می‌کند.



همانند [روش 6.8](#Bagofwords)، خروجی یک ماتریس پراکنده (sparse matrix) است. اما اگر می‌خواهیم خروجی را به عنوان یک ماتریس چگال (dense matrix) مشاهده کنیم، می‌توانیم از `.toarray` استفاده کنیم.



vocabulary\_ کلمه هر ویژگی را به ما نشان می‌دهد:



**بحث**

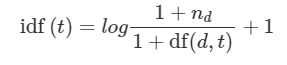
هر چقدر یک کلمه در یک سند بیشتر ظاهر شود، احتمالاً برای آن سند مهم‌تر است. به عنوان مثال، اگر کلمه "اقتصاد" به طور متداول در یک سند ظاهر شود، نشان‌دهنده این است که احتمالاً این سند مربوط به اقتصاد است. این مفهوم به عنوان فراوانی ترم (tf) شناخته می‌شود.

به عکس، اگر یک کلمه در بسیاری از اسناد ظاهر شود، احتمالاً کم‌اهمیت‌تر از هر یک از اسناد فردی است. به عنوان مثال، اگر هر سندی در داده‌های متنی حاوی کلمه "بعد از" باشد، احتمالاً این کلمه یک کلمه غیرمهم است. این مفهوم به عنوان فراوانی سند (df) شناخته می‌شود.

با ترکیب این دو آمار، می‌توانیم به هر کلمه یک امتیاز اختصاص دهیم که نشان‌دهنده اهمیت آن کلمه در یک سند است. به طور خاص، ما فراوانی ترم (tf) را در معکوس فراوانی سند (idf) ضرب می‌کنیم.



که t یک کلمه و d یک سند است. تعدادی از نسخه‌های مختلف برای محاسبه tf و idf وجود دارد. در scikit-learn، tf به سادگی تعداد دفعاتی که یک کلمه در سند ظاهر می‌شود محاسبه می‌شود و idf به صورت زیر محاسبه می‌شود:



که در آن nd تعداد اسناد و df(d,t) فراوانی اسناد t است (یعنی تعداد اسنادی که عبارت در آن ظاهر می‌شود).

به صورت پیش‌فرض،scikit-learn بردارهای TF-IDF را با استفاده از نرم اقلیدوسی (L2 norm) نرمال می‌کند. هر چه مقدار حاصل بیشتر باشد، کلمه مهم‌تری برای سند است.

**منابع بیشتر**

* [scikit-learn documentation : tf-idf term weighting](http://bit.ly/2HT2wmW)

**6.11 استفاده از بردارهای متن برای محاسبه شباهت متن در یک عبارت جستجو**

**مسئله**

شما می‌خواهید از بردارهای tf-idf برای پیاده سازی یک تابع جستجوی متن در پایتون استفاده کنید.

**راه‌حل**

شباهت کسینوس بین بردارهای tf-idf را با استفاده از scikit-learn محاسبه کنید:



**بحث**

بردارهای متنی برای موارد استفاده از NLP مانند موتورهای جستجو بسیار مفید هستند. پس از محاسبه بردارهای tf-idf برای مجموعه‌ای از جملات یا اسناد، می‌توانیم از همان شیء tfidf برای بردار کردن مجموعه‌های متن آینده استفاده کنیم. سپس، می‌توانیم شباهت کسینوس را بین بردار ورودی و ماتریس بردارهای دیگر محاسبه کرده و بر اساس مرتبط‌ترین اسناد مرتب کنیم.

شباهت کسینوس در محدوده [0، 1.0] است که 0 کمترین شباهت و 1 بیشترین شباهت را دارد. از آنجایی که ما از بردارهای tf-idf برای محاسبه شباهت بین بردارها استفاده می‌کنیم، فراوانی وقوع یک کلمه نیز در نظر گرفته می‌شود. با این حال، با یک مجموعه کوچک (مجموعه ای از اسناد) حتی کلمات "متداول" ممکن است اغلب ظاهر نشوند. در این مثال، "سوئد بهترین است" مرتبط ترین متن با عبارت جستجوی ما "برزیل بهترین است" است. از آنجایی که عبارت کوئری به برزیل اشاره می‌کند، ممکن است انتظار داشته باشیم "من عاشق برزیل هستم. برزیل!» مرتبط ترین مورد باشد. با این حال، "سوئد بهترین است" به دلیل کلمات "است" و "بهترین" شبیه ترین است. با افزایش تعداد اسنادی که به مجموعه خود اضافه می‌کنیم، کلماتی که اهمیت کمتری دارند، وزن کمتری خواهند داشت و تأثیر کمتری بر محاسبات شباهت کسینوس ما خواهند داشت.

**همچنین ببینید:**

* [**شباهت کسینوس، GeeksForGeeks**](https://www.geeksforgeeks.org/cosine-similarity/)

**6.12 استفاده از یک طبقه بندی تحلیل احساسات[[5]](#footnote-5)**

**مسئله**

شما می‌خواهید احساس برخی از متون را برای استفاده به عنوان یک ویژگی یا در تجزیه و تحلیل داده‌های پایین‌رونده[[6]](#footnote-6) طبقه بندی کنید.

**راه‌حل**

از طبقه‌بندی‌کننده احساسات کتابخانه transformers استفاده کنید.



**بحث**

کتابخانه Transformers یک کتابخانه‌ی بسیار محبوب برای وظایف NLP است و شامل تعدادی API با استفاده آسان برای مدل‌های آموزشی یا استفاده از مدل‌های از پیش آموزش دیده است. ما در مورد NLP و این کتابخانه در فصل 22 بیشتر صحبت خواهیم کرد، اما این مثال به عنوان مقدمه ای در سطح بالا برای قدرت استفاده از طبقه بندی کننده‌های از پیش آموزش دیده در خطوط لوله یادگیری ماشین شما برای تولید ویژگی ها، طبقه بندی متن یا تجزیه و تحلیل داده‌های بدون ساختار آورده شده است.

**همچنین ببینید:**

* [تور سریع Hugging Face Transformers](https://huggingface.co/docs/transformers/quicktour)

1. عملیاتی است که در آن اطلاعات قابل بازدید یک سایت به صورت اتوماتیک جمع‌آوری شده و برای انجام تحلیل داده مورد استفاده قرار می‌گیرد. [↑](#footnote-ref-1)
2. - pipeline [↑](#footnote-ref-2)
3. - entities [↑](#footnote-ref-3)
4. - downstream [↑](#footnote-ref-4)
5. - Sentiment Analysis [↑](#footnote-ref-5)
6. - downstream [↑](#footnote-ref-6)