Apache OpenNLP

گردآورنده و مترجم: سید علی میر محمد حسینی

استاد: امیرسام بهادر

تیر99

فهرست مطالب

پیش گفتار: 3

چکیده(Abstract): 4

مقدمه(Introduction): 4

شکل 1-ارتباط پردازش زبان های طبیعی با علوم کامپیوتر و زبان شناسی 4

مواد و روش ها(Materials and methods): 5

مرحله 1 5

مرحله 2 5

ارزیابی 21

Tokenization: 21

Document Categorizer: 33

Lemmatizer : 39

Chunker: 43

Training API: 45

Parser: 47

Coreference Resolution: 51

نتیجه‌گیری (Conclusion): 51

تقدیر و تشکر (Acknowledgment): 51

مراجع(References) : 51

پیش گفتار:

OPENNLP یک ابزار یادگیری ماشین برای پرازش متن در زبان های طبیعی می باشد این ابزار محصول شرکت Apache که یک سازمان غیر انتفاعی که حوزه فعالیتش پشتبانی و توسعه پروژه های مختلف متن باز است می باشد.

برای آشنایی خواننده با هر بخش از این مقاله یک نمونه کد مرتبط با هر بخش آورده شده تا در کنار آشنایی تئوري با مطالب به برسی مثال های عملی نیر بپردازیم

در این مقاله از Maven که یک ابزار مدیریت پروژه است که شامل یک Project object model (POM) و یکسری از استاندارد ها ی طول عمر پروژه (Lifecycle) ، سیستم مدیریت وابستگی(dependency) و منطقی برای اجرا اهداف Plugin در فاز های تعریف شده طول عمر می باشد استفاده شده تا کتابخانه ها و ابزار های مورد نیاز را به پروژه اضافه کنیم.

در این مقاله سعی شده که تمام ویژگی های ذکر شده در چکیده پوشش داده شود برای کسب اطلاعات تکمیلی در مورد این پروژه می توانید به سایت آن مراجعه فرمایید.

در پایان از استاد ارجمند جناب آقاي مهندس امیر سام بهادر که از دوره جاوا مقدماتی تا تکمیلی خدمت ایشان بودم , دانش برنامه نویسی به زبان جاوا را از ایشان آموخته ام و از راهنمایی هاي ایشان جهت تدوین این مقاله بهره مند شده ام، سپاسگزارم.

سید علی میر محمد حسینی

چکیده(Abstract):

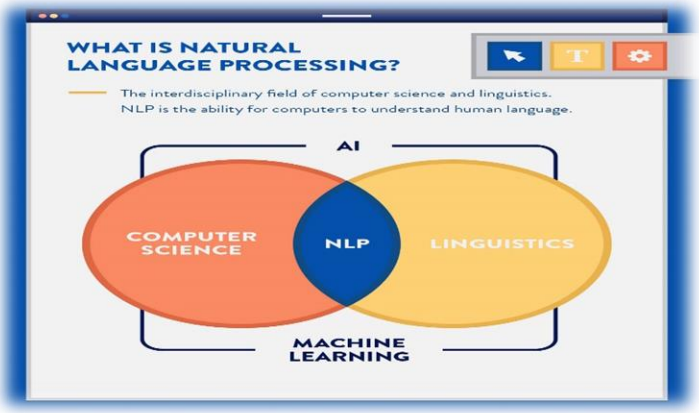
در این مقاله مروری بر پردازش متن در زبان های طبیعی انجام شده است که به بیان فنون پردازش زبان در ابزار OpenNLP از جمله Language Detection ، Sentence Detection ، Tokenization ، Name Finder ، Document Categorizer ،Part of speech tagger ، Lemmatizer ، Chunker ، Parser و توضیح مختصری در مورد language model و ساخت مدل سفارشی می پردازیم.

این امکانات معمولا برای تولید سرویس های پردازش متن پیشرفته مورد نیاز است ، OpenNLP شامل ویژگی های ماکزیم آنتروپی (که معیاری عددی از میزان اطلاعات یا میزان تصادفی بودن یک متغیر تصادفی است) و پرسپترون (که یک الگوریتم یادگیری ماشین و در دسته یادگیری با نظارت قرار می‌گیرد است) می شود. هدف اصلی این پروژه با توجه به مواردهای ذکر شده یک ابزار کامل برای هدف های فوق می باشد علاوه برآن هدف تهیه تعداد زیادی مدل از پیش ساخته برای انواع مختلف زبانها و همچنین منابع متنی حاشیه نویسی که از آن مدل ها گرفته شده است می باشد.

کلید واژه ها: Language Detection,language model، OpenNLP ، Sentence Detection ، Tokenization ، Name Finder ، Document Categorizer ،Part of speech tagger ، Lemmatizer ، Chunker

مقدمه(Introduction):

پردازش زبان‌های طبیعی یکی از زیرشاخه‌های بااهمیت در حوزهٔ گستردهٔ علوم رایانه، هوش مصنوعی، که به تعامل بین کامپیوتر و زبان‌های (طبیعی) انسانی می‌پردازد؛ بنابراین پردازش زبان‌های طبیعی بر ارتباط انسان و رایانه، متمرکز است. پس چالش اصلی و عمده در این زمینه درک زبان طبیعی و ماشینی کردن فرایند درک و برداشت مفاهیم بیان‌شده با یک زبان طبیعیِ انسانی است. به تعریف دقیق‌تر، پردازش زبان‌های طبیعی عبارت است از استفاده از رایانه برای پردازش زبان گفتاری و زبان نوشتاری. بدین معنی که رایانه‌ها را قادر سازیم که گفتار یا نوشتار تولید شده در قالب و ساختار یک زبان طبیعی را تحلیل و درک نموده یا آن را تولید نمایند. در این صورت، با استفاده از آن می‌توان به ترجمهٔ زبان‌ها پرداخت، از صفحات وب و بانک‌های اطلاعاتیِ نوشتاری جهت پاسخ دادن به پرسش‌ها استفاده کرد، یا با دستگاه‌ها، مثلاً برای مشورت گرفتن به گفت‌وگو پرداخت. این‌ها تنها مثال‌هایی از کاربردهای متنوع پردازش زبان‌های طبیعی هستند. در این مقاله با توجه به این که ما از ابزار OpenNLP استفاده می کنیم از پرازش زبان طبیعی، پردازش متن مورد هدف است



شکل 1-ارتباط پردازش زبان های طبیعی با علوم کامپیوتر و زبان شناسی

مواد و روش ها(Materials and methods):

در این قسمت مراحل انجام یک پروژه ساده را با OpenNLP بررسی می کنیم :

مرحله 1: در این مرحله باید prerequisite (ملزومات)های انجام پروژه را قبل از آن به پروژه اضافه کنیم:

در قسمت از maven که یک ابزار build است استفاده می کنیم برای استفاده از آن یک فایل از جنس xml با نام pom.xml اسفاده می کنیم.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.javachannel</groupId>
    <artifactId>opennlp</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.opennlp</groupId>
            <artifactId>opennlp-tools</artifactId>
            <version>1.8.4</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>[6.8.21,)</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

مرحله 2 :در این مرحله به توضیح مختصر language model, نحوه ساخت مدل سفارشی , انواع تکنیک های پردازش متن و روش های متعدد آن را توضیح می دهیم:

:Language Model

یک مدل زبان آماری توزیع احتمال در توالی کلمات است. با توجه به چنین توالی مثلاً به طول m ، یک احتمال $P(w_1,w_2....w_n)$را به این ترتیب به کل دنباله اختصاص می دهد.

مدل زبان زمینه تمایز بین کلمات و عباراتی را که شبیه به هم هستند فراهم می کند. به عنوان مثال ، در انگلیسی آمریکایی ، عبارات "recognize speech " و " wreck a nice beach " آوای یکسانی دارند ، اما معنا های مختلف دارند.

کمبود داده یک مشکل اساسی در ساخت مدل های زبان است. بیشتر توالی کلمات ممکن در آموزش نظارت نمی شود.

یک راه حل این فرضیه است که احتمال یک کلمه فقط به کلمات n قبلی بستگی دارد. این به عنوان یک مدل n-gram یا مدل unigram شناخته می شود که n = 1. مدل unigram به عنوان مدل bag of words نیز شناخته می شود.

تخمین احتمال نسبی عبارات مختلف , در بسیاری از برنامه های پردازش زبان طبیعی ، به ویژه مواردی که متن را به عنوان یک خروجی تولید می کنند ، مفید است.

از مدل سازی زبان در speech recognition,machine translation,part-of-speech tagging, parsing, Optical Character Recognition, handwriting recognition, information retrieval, و برنامه های دیگر استفاده می شود.

در تشخیص گفتار ، صداها با توالی کلمه مطابقت دارند. ابهامات آسان تر می شوند وقتی شواهد مدل زبان, با یک مدل تلفظ و یک مدل صوتی ادغام می شوند.

از مدل های زبان در بازیابی اطلاعات در مدل query likelihood استفاده می شود. در آنجا ، یک مدل زبان جداگانه با هر سند در یک مجموعه همراه است. اسناد براساس احتمال Q quert در مدل زبان اسناد رتبه بندی می شوند$M_d=P(Q|M_d)$. معمولاً برای این منظور از مدل زبان unigram استفاده می شود.

:Model types in language model

:Unigram

یک مدل unigram را می توان به عنوان ترکیبی از چند one-state finite automata دانست.

این احتمالات اصطلاحات مختلف را در یک زمینه به عنوان مثال ، تقسیم می کند. از:

$P(t_1t_2t_3)=P(t_1)P(t_2|t_1)P(t_3|t_1t_2)$

به:

$P_{uni}(t_1t_2t_3)=P(t1)P(t2)P(t3)$

در این مدل ، احتمال هر کلمه فقط به احتمال خاص خود آن کلمه در سند بستگی دارد ، بنابراین ما فقط به صورت واحد one-state finite automata داریم.

خود automaton دارای توزیع احتمال در کل واژگان مدل که جمع شده برابر با 1 است. در زیر نمونه ای از یک مدل unigram از یک سند است.

Terms	Probability in doc
a	0.1
world	0.2
likes	0.05
we	0.05
share	0.3
...	...

$$\sum_{\text{term in doc}} P(\text{term}) = 1$$

احتمال ایجاد شده برای یک درخواست خاص با فرمول زیر محاسبه می شود:

$$P(\text{query}) = \prod_{\text{term in query}} P(\text{term})$$

اسناد مختلف از مدل های unigram برخوردار هستند که احتمالاً کلمات مختلف در آن وجود دارد. توزیع احتمال از اسناد مختلف برای تولید احتمال ضربه برای هر پرس و جو استفاده می شود. اسناد را می توان برای پرس و جو با توجه به احتمالات رتبه بندی کرد. نمونه مدلهای unigram از دو سند:

Terms	Probability in Doc1	Probability in Doc2
a	0.1	0.3
world	0.2	0.1
likes	0.05	0.03
we	0.05	0.02
share	0.3	0.2
...	...	...

در زمینه های بازیابی اطلاعات ، مدل های زبان unigram اغلب صاف می شوند تا از مواردی که P(term)=0. وجود دارد جلوگیری شود. یک رویکرد رایج تولید یک مدل حداکثر احتمال برای کل مجموعه و به صورت خطی مدل مجموعه با یک مدل حداکثر احتمال درج می باشد. هر سند برای صاف کردن مدل است.

n-gram:

در یک مدل n-gram، احتمال $P(w_1,\ldots,w_m)$ از نظارت جمله $w_1,\ldots,w_m$ به عنوان تقریب است:

$$P(w_1,\ldots,w_m) = \prod_{i=1}^{m} P(w_i \mid w_1,\ldots,w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)},\ldots,w_{i-1})$$

فرض بر این است که احتمال مشاهده i^{th} کلمه w i در تاریخچه واژه های i – 1 قبلی با احتمال مشاهده آن در تاریخچه متن کوتاه شده از کلمات n - 1 قبلی تقریب می یابد . (n^{th} در markov property)

$$P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\mathrm{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\mathrm{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$$

اصطلاحات مدل های bigram و trigram به ترتیب مدلهای n-gram را با n = 2 و n = 3 نشان می دهند.

به طور معمول ، احتمالات مدل n-gram مستقیماً از شمارش دنباله گرفته نمی شود ، زیرا مدل های بدست آمده از این روش هنگام مواجهه با هر n-گرم که قبلاً به صراحت دیده نشده است ، دچار مشکلات جدی می شوند. در عوض ، نوعی smoothing ضروری است ، و برخی از کل احتمالات وزن آن را به کلمات دیده نشده یا n-gram اختصاص می دهند. روشهای مختلفی استفاده می شود ، از یکنواختی ساده "اضافه کردن یک" (تعداد 1 عدد n-gram نادیده بگیرید ، به عنوان قبل از عدم اطلاع) تا مدل های پیچیده تر ، مانند مدل های Good-Turing discounting یا back-off .

بازنمود:
بازنمودهای دو طرفه در هر دو لایه قبل و پس زمینه (به عنوان مثال ، کلمات) شرط دارند.

مثال:

در یک مدل زبانی bigram (n = 2) ، احتمال جمله ای که من دیدم خانه قرمز است تقریباً برابر است با:

$$P(\mathrm{I, saw, the, red, house}) \approx P(\mathrm{I} \mid \langle s \rangle) P(\mathrm{saw} \mid \mathrm{I}) P(\mathrm{the} \mid \mathrm{saw}) P(\mathrm{red} \mid \mathrm{the}) P(\mathrm{house} \mid \mathrm{red}) P(\langle /s \rangle \mid \mathrm{house})$$

در حالی که در یک مدل زبان trigram(3 نفر) ، تقریبا برابر است با:

$$P(\mathrm{I, saw, the, red, house}) \approx P(\mathrm{I} \mid \langle s \rangle, \langle s \rangle) P(\mathrm{saw} \mid \langle s \rangle, I) P(\mathrm{the} \mid \mathrm{I, saw}) P(\mathrm{red} \mid \mathrm{saw, the}) P(\mathrm{house} \mid \mathrm{the, red}) P(\langle /s \rangle \mid \mathrm{red, house})$$

توجه داشته باشید که متن اولین n - 1 n-gram ها با نشانگرهای شروع جمله پر می شود ، که معمولاً به <s> اشاره می شود.

علاوه بر این ، بدون نشانگر پایان جمله ، احتمال دنباله غیرمعمول * من می دیدم که همیشه بالاتر از جمله طولانی تر که خانه قرمز را دیدم بیشتر است.

نمایی:
مدل های زبان حداکثر آنتروپی با استفاده از توابع ویژگی رابطه بین یک کلمه و تاریخ n-gram را رمزگذاری می کنند. معادله برابر است با:

$$P(w_m | w_1, \ldots, w_{m-1}) = \frac{1}{Z(w_1, \ldots, w_{m-1})} \exp(a^T f(w_1, \ldots, w_m))$$

جایی که $Z(w_1,\ldots,w_{m-1})$, a بردار پارامتر است ، و $f(w_1,\ldots,w_m)$ عملکرد ویژگی است در ساده ترین حالت ، عملکرد ویژگی فقط نشانگر وجود n-gram خاص است. استفاده از مقدمات قبلی درباره a یا نوعی تنظیم کردن مفید است.

مدل log-bilinear نمونه دیگری از یک مدل زبان نمایی است.

شبکه عصبی:
مدل های زبان عصبی (یا مدل های زبان مستمر فضا) از بازنمایی مداوم یا تعبیه کلمات برای پیش بینی خود استفاده می کنند. این مدل ها از شبکه های عصبی استفاده می کنند. تعبیه های مداوم در فضا کمک می کند تا نفرین ابعاد در مدل سازی زبان کاهش یابد:

همانطور که مدل های زبانی بر روی متون بزرگتر و بزرگتر آموزش داده می شوند ، تعداد کلمات منحصر به فرد (واژگان) افزایش می یابد. تعداد توالی های احتمالی کلمات با اندازه واژگان بصورت نمایی افزایش می یابد و به دلیل وجود توالی های اکتشافی ، مشکل کمبود داده ایجاد می کند.

بنابراین ، برای برآورد درست احتمالات ، آماری لازم است. شبکه های عصبی با بیان کلمات به صورت توزیع شده از این مشکل جلوگیری می کنند ، به عنوان ترکیب غیر خطی وزن ها در یک شبکه عصبی. توضیحات جایگزین این است که یک شبکه عصبی عملکرد زبان را تقریب می دهد. معماری شبکه عصبی ممکن است فیدر یا مکرر باشد ، و در حالی که اولی ساده تر است ، دومی رایج تر است.

به طور معمول ، مدل های زبان خالص عصبی به عنوان طبقه بندی کننده های احتمالی که یاد می گیرند توزیع احتمال را یاد بگیرند ساخته و آموزش داده می شوند

$$P(w_t|\text{context})\ \forall t \in V.$$

به عنوان مثال ، شبکه با توجه به برخی از متن های زبانی ، برای پیش بینی توزیع احتمال بر روی واژگان آموزش داده است. این کار با استفاده از الگوریتم های آموزش خالص عصبی استاندارد مانند نزول شیب تصادفی با استفاده از پشت پرده انجام می شود. متن ممکن است یک پنجره با اندازه ثابت از کلمات قبلی باشد ، به طوری که شبکه پیش بینی می کند.

$$P(w_t|w_{t-k},\ldots,w_{t-1})$$

از یک بردار ویژگی که کلمات k قبلی را نشان می دهد. گزینه دیگر استفاده از کلمات "آینده" و همچنین کلمات "گذشته" به عنوان ویژگی است ، به طوری که احتمال برآورد شده وجود دارد.

$$P(w_t|w_{t-k},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+k}).$$

به این مدل کیسه ای از کلمات گفته می شود. هنگامی که بردارهای ویژگی کلمات موجود در متن با یک عمل مداوم ترکیب می شوند ، از این مدل به عنوان معماری پیوسته کلمات (CBOW) یاد می شود.

گزینه سوم که آهسته تر از CBOW تمرین می کند اما عملکرد کمی بهتر دارد ، برطرف کردن مشکل قبلی و ایجاد یک شبکه عصبی با استفاده از یک کلمه است. رسماً ، با توجه به توالی کلمات آموزش $w_1,w_2,w_3,\ldots w_T$ ، یک میانگین احتمال ورود به سیستم را به حداکثر می رساند.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-k\leq j\leq k, j\neq 0}\log P(w_{t+j}|w_t)$$

جایی که k ، اندازه متن آموزش ، می تواند تابعی از کلمه مرکزی w_t باشد. به این مدل الگوی زبان skip-gram گفته می شود. مدل های bag-of-word و skip-gram اساس برنامه word2vec هستند.

به جای استفاده از مدل های زبان خالص عصبی برای تولید احتمالات واقعی ، استفاده از نمایه توزیع شده رمزگذاری شده در لایه های "پنهان" شبکه ها به عنوان بازنمایی کلمات ، معمول است. سپس هر کلمه بر روی یک بردار واقعی n بعدی به نام کلمه جاسازی قرار می گیرد که در آن n اندازه لایه درست قبل از لایه خروجی است. بازنمودها در مدلهای گریدپردازی از ویژگی مشخصی برخوردار است که روابط معنایی بین کلمات را به عنوان ترکیب خطی مدل می کنند و نوعی ترکیب بندی را ضبط می کنند. به عنوان مثال ، در برخی از چنین مدل ها ، اگر v تابعی باشد که کلمه w را به نمایندگی بردار n-d خود ترسیم می کند ، پس از آن :

$$v(\text{king}) - v(\text{male}) + v(\text{female}) \approx v(\text{queen})$$

که در آن ≈ با تصریح اینکه سمت راست آن باید نزدیکترین همسایه از ارزش سمت چپ باشد ، دقیق ساخته شده است.

دیگر:
یک مدل زبان موقعیتی احتمال کلمات داده شده در متن را که در نزدیکی یکدیگر اتفاق می افتند ، لزوماً بلافاصله مجاور ارزیابی نمی کند. به طور مشابه ، مدلهای مفاهیم کیفی ، معنای معانی مرتبط با عبارات چند کلمه ای مانند buy_christmas_present را به خود جلب می کنند ، حتی وقتی از آنها در جملات پرمحتوا استفاده می شود مانند "امروز من بسیاری از هدایای کریسمس بسیار خوبی را خریداری کردم".

علیرغم موفقیت های محدود در استفاده از شبکه های عصبی ، نویسندگان در مدل سازی زبانهای نشان دهنده نیاز به تکنیک های دیگر را تأیید می کنند.

:Create Custom model
برای ساخت یک مدل سفارشی در OpenNLP بهتر است از ابزار opennlp-addons(modelbuilder addon) که در مسیر https://github.com/apache/opennlp-addons/tree/master/modelbuilder-addon

وجود دارد استفاده کنید, شما به آن فایلی از اسامی می دهید ، که برای آموزش یک مدل از اسامی و برخی از داده های شما (جملات) استفاده می کند. با این حال ، اگر به دنبال نام های خاصی از اشخاص عموماً ناشناخته هستید ، شاید بهتر باشد فقط از یک لیست و چیزی مانند regex برای کشف نام استفاده کنید نه NER.

در قالب یک مثال(named entity recognition (NER)) مراحل ساخت مدل توسط این ابزار را بررسی می کنیم

نکته: ویژگی named entity recognition (NER) در OpenNLP پشتیبانی می شود که در ادامه به بررسی این ویژگی می پردازیم

برای ساخت مدل سفارشی:

1-احتیاج به فایل آموزشی حاوی `annotated spans` در مورد نوع کلاس در این قالب داریم:

```
Ali
```

2-برنامه باید یک sample data stream را باز کند.
3-متد NameFinderME.train را فراخوانی کنید.
4-TokenNameFinderModel برای استفاده در آینده در فایل ذخیره کنید.

در کد زیر نحوه آموزش مدل را بررسی می کنیم:

```
package openNLP;

import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.charset.Charset;
import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.NameSample;
import opennlp.tools.namefind.NameSampleDataStream;
import opennlp.tools.namefind.TokenNameFinderFactory;
import opennlp.tools.namefind.TokenNameFinderModel;
```

```
import opennlp.tools.util.ObjectStream;
import opennlp.tools.util.PlainTextByLineStream;
import opennlp.tools.util.TrainingParameters;

public class CustomClassifierTrainer{
//Where to save the model once trained
  static String onlpModelPath = "/Users/user_name/eclipse-
workspace/openNLP/OpenNLP_models/en-ner-asiannames.bin;"
 //    location of the training data set
    static String trainingDataFilePath = "/Users/user_name/eclipse-
workspace/openNLP/trainingData/asiannames.txt;"

       public static void main(String[] args) throws IOException {

           Charset charset = Charset.forName("UTF-8");

           ObjectStream lineStream=
            new PlainTextByLineStream(() -> new
FileInputStream(trainingDataFilePath), charset);

           ObjectStream sampleStream = new NameSampleDataStream(lineStream);

           TokenNameFinderModel model;
           TokenNameFinderFactory nameFinderFactory = new
TokenNameFinderFactory() ;

           try {
             model = NameFinderME.train("en" ,"asian.person" ,

                  sampleStream ,
                  TrainingParameters.defaultParams(),
                  nameFinderFactory);
           }
           Finally{
             sampleStream.close() ;
           }
//Saving the model
           BufferedOutputStream modelOut = null;
               try{
             modelOut = new BufferedOutputStream(new
FileOutputStream(onlpModelPath));
             model.serialize(modelOut);
           } finally{
             if (modelOut != null)
                modelOut.close()      ;
        }
       }
}
```

باید در کد بالا از 2 مورد توجه داشته باشید:

1- شما باید در نوع InputStreamFactory را به عنوان اولین آرمان به .train پاس دهیم.وقتی به Javadoc این اینترفیس نگاه می کنید.می بینید که یک تک متد createInputStream() که هیچ آرگمان ورودی نمیگیرد و ساخت و برگرداندن ()createInputStream را پشتیبانی می کند.

بنابراین یک مقدار معتبر به شکل زیر خواهد بود :

```
() -> new FileInputStream(trainingDataFilePath)
```

یعنی یک لامبدا که هیچ ورودی ندارد و یک جریان ورودی جدید ایجاد می کند ، و بنابراین می تواند استباط شود که یک InputStreamFactory است.

2- قرار نیست هنگام فراخوانی یک متد انواع آرگومان ها را مشخص کنید. فقط هنگام تعریف یک متد انجام میشود . بنابراین:

```
NameFinderME.train("en","asian.person" ,
,sampleStream
,TrainingParameters.defaultParams ()   )
TokenNameFinderFactory nameFinderFactory ;
```

باید باشد:

```
NameFinderME.train("en","asian.person"
,sampleStream ,
TrainingParameters.defaultParams() ,
nameFinderFactory);
```

بنابراین ، شما نیاز به تعریف و شی سازی از `nameFinderFactory` قبل از فراخوانی تابع NameFinderME.train() دارید.

و اکنون زمان آن رسیده است که مدل دلخواه خود را آزمایش کنید و در اینجا نحوه انجام این کار آمده است:

```
to test the builtin name finder model //
public class nameFinder{

public void findName(String paragraph) throws IOException{
InputStream inputStream = new
FileInputStream("/Users/Fahad/eclipse-workspace/openNLP/OpenNLP_models/en-
ner-person.bin") ;
TokenNameFinderModel model = null;
 try       {
model = new
TokenNameFinderModel(inputStream);
} (catch (IOException e) {
TODO Auto-generated  // block
catch
 e.printStackTrace ();
}
NameFinderME nameFinder = new NameFinderME(model);
String[] tokens = tokenise(paragraph);

Span nameSpans[] = nameFinder.find(tokens);
for(Span s: nameSpans) {
System.out.println(tokens[s.getStart()]) ;
System.out.println(s.getType()+" :
"+tokens[s.getStart()]+"\t [probability="+s.getProb()+"]") ;
{
{
       to test our newly built custom model //
public void asianFindName(String paragraph) throws IOException {
```

```
InputStream inputStream = new
FileInputStream("/Users/Fahad/eclipse-workspace/openNLP/OpenNLP_models/en-
;ner-asiannames.bin")
;TokenNameFinderModel model = null
try       {
model = new
TokenNameFinderModel(inputStream);
} (catch (IOException e {
TODO Auto-generated  // block
catch block
e.printStackTrace     ();
}
NameFinderME nameFinder = new NameFinderME(model) ;
String[] tokens = tokenise(paragraph)  ;

Span nameSpans[] = nameFinder.find(tokens) ;
for(Span s: nameSpans)  {
System.out.println(tokens[s.getStart()]);
System.out.println(s.getType()+" :
"+tokens[s.getStart()]+"\t [probability="+s.getProb()+"]");
{
{
both methods above need to tokenise the sentence first before extracting  //
NER
public String[] tokenise(String sentence) throws IOException   {
InputStream inputStreamTokenizer = new
FileInputStream("/Users/Fahad/eclipse-workspace/openNLP/OpenNLP_models/en-
token.bin");
TokenizerModel tokenModel = new
TokenizerModel(inputStreamTokenizer);
TokenizerME tokenizer = new TokenizerME(tokenModel);
return tokenizer.tokenize(sentence);
}

public static void main(String[] args) throws Exception{

nameFinder namefinder = new nameFinder();
namefinder.findName("Where is Charlie and Mike.")       ;
namefinder.findName("Fraser is my son.")       ;
namefinder.findName("I love Dominoes.")       ;
namefinder.findName("I love Seb.")       ;

namefinder.asianFindName("Salah is not my relative.");
namefinder.asianFindName("Salah is not my relative, will be going
school soon.");
namefinder.asianFindName("I love Dominoes.");
namefinder.asianFindName("I love Mr. Noor.");
namefinder.asianFindName("Mr. Ching is my son.");
namefinder.asianFindName("Where is Charlie and Mike.");
}
}
```

پارامترهای پیش فرض TrainingParamet تعریف شده است مثل:

```
public static TrainingParameters defaultParams  () {
```

```
TrainingParameters mlParams = new TrainingParameters () ;
mlParams.put(TrainingParameters.ALGORITHM_PARAM, "MAXENT");
mlParams.put(TrainingParameters.TRAINER_TYPE_PARAM,
EventTrainer.EVENT_VALUE);
mlParams.put(TrainingParameters.ITERATIONS_PARAM, 100) ;
mlParams.put(TrainingParameters.CUTOFF_PARAM, 5)  ;

return mlParams ;
}
```

همچنین می توانید پارامترهای آموزش پیش فرض را با استفاده از `TrainingParameters` تغییر دهید:

```
TrainingParameters paramaters = new TrainingParameters() ;
paramaters.put(TrainingParameters.ITERATIONS_PARAM, 100);
paramaters.put(TrainingParameters.CUTOFF_PARAM, 1);
paramaters.put(TrainingParameters.ALGORITHM_PARAM, "NAIVEBAYES");
```

```
TokenNameFinderFactory tnff = new TokenNameFinderFactory();
model = NameFinderME.train("en", modelName, sampleStream, parameters, tnff);
```

نتیجه مدل این است که Noor را برای asian.names استخراج می کند اما از سایرین پرش می کند. دلیل این است که داده های آموزش بسیار اندک است. برای یادگیری اسامی و متن حداقل باید حداقل 15000 جمله در نظر بگیرید.

Language Detector:2-1

آشکارساز زبان OpenNLP با توجه به قابلیت های مدل ، سندی را با استفاده از استاندارد زبان های ISO-639-3 طبقه بندی می کند.

یک مدل را می توان با الگوریتم های Maxent، Perceptron یا Naive Bayes آموزش داد. به طور پیش فرض یک متن را عادی می کند و text generator، n-gramهای اندازه 1 ، 2 و 3 را استخراج می کند.

اندازه های n-gram، نرمال سازی و text generator با گسترش LanguageDetectorFactory قابل تنظیم است.

ابزار Language Detector:
ساده ترین روش برای آزمایش language detector ، ابزار خط فرمان است. این ابزار فقط برای نمایش و آزمایش در نظر گرفته شده است. دستور زیر نحوه استفاده از ابزار language detector را در سیستم عامل لینوکسی نشان می دهد.

```
$ bin/opennlp LanguageDetector model
```

ورودی از ورودی استاندارد خوانده می شود و خروجی به خروجی استاندارد نوشته می شود ، مگر اینکه آنها هدایت شوند و یا pipe شوند.

مثال زیر را در خط فرمان در سیستم عامل لینوکس توضیع CentOS اجرا کردیم:

```
[root@localhost bin]# ./opennlp LanguageDetector langdetect-183.bin  < /home/ali/Desktop/ali.txt
```

که با اجرای این مدل به خروجی زیر میرسیم:

```
Loading Language Detector model ... done (2.323s)
eng     Usually Sentence Detection is done before the text is tokenized and that's the way the pre-trained models on the web site are trained, but it
 is also possible to perform tokenization first and let the Sentence Detector process the already tokenized text. The OpenNLP Sentence Detector canno
t identify sentence boundaries based on the contents of the sentence. A prominent example is the first sentence in an article where the title is mist
akenly identified to be the first part of the first sentence. Most components in OpenNLP expect input which is segmented into sentences.



Average: 22.7 doc/s
Total: 1 doc
Runtime: 0.044s
Execution time: 2.755 seconds
```

: Language Detector API

برای انجام طبقه بندی به یک مدل یادگیری ماشین نیاز خواهید داشت - اینها در کلاس LanguageDetectorModel از ابزارهای OpenNLP محصور شده اند.

ابتدا باید بایت ها را از serialized model در InputStream بگیرید . این کار را برای شما انجام می دهیم ، زیرا شما این کار را به صورت serialized انجام دادید.:

```
InputStream is = ...
LanguageDetectorModel m = new LanguageDetectorModel(is);
```

با LanguageDetectorModel داریم :

```
String inputText = ...
LanguageDetector myCategorizer = new LanguageDetectorME(m);

// Get the most probable language
Language bestLanguage = myCategorizer.predictLanguage(inputText);
System.out.println("Best language: " + bestLanguage.getLang());
System.out.println("Best language confidence: " +
bestLanguage.getConfidence());

// Get an array with the most probable languages
Language[] languages = myCategorizer.predictLanguages(null);
```

توجه داشته باشید که هر دو API یا CLI متن کامل را برای انتخاب محتمل ترین زبان ها در نظر می گیرند. برای دستیابی به زبان مختلط می توان بخش های کوچکتری از متن را برای یافتن مناطق زبان تجزیه و تحلیل کرد.

: Training

Language Detectorرا می توان در مطالب آموزشی حاشیه نویسی آموزش داد. داده ها می توانند در قالب آموزش OpenNLP Language Detector باشند. این یک سند در هر خط است ، حاوی کد زبان ISO-639-3 و متن توسط یک tab جدا شده است. سایر قالب ها نیز می توانند در دسترس باشند. نمونه زیر نمونه بالا را با فرمت مورد نیاز نشان می دهد.

```
spa     A la fecha tres calles bonaerenses recuerdan su nombre (en Ituzaingó,
Merlo y Campana). A la fecha, unas 50 \
naves y 20 aviones se han perdido en esa área particular del
océano Atlántico.
deu     Alle Jahre wieder: Millionen Spanier haben am Dienstag die Auslosung
in der größten Lotterie der Welt verfolgt.\
Alle Jahre wieder: So gelingt der stressfreie Geschenke-
Umtausch Artikel per E-Mail empfehlen So gelingt der \
```

```
stressfre ie Geschenke-Umtausch Nicht immer liegt am Ende das
unter dem Weihnachtsbaum, was man sich gewünscht hat.
srp     Већина становника боравила је кућама од блата или шаторима, како би
радили на својим удаљеним пољима у долини \
Јордана и напасали своје стадо оваца и коза. Већина
становника говори оба језика.
lav     Egija Tri-Active procedūru īpaši iesaka izmantot siltākajos
gadalaikos, jo ziemā aukstums var šķist arī \
nepatīkams. Valdība vienojās, ka izmaiņas nodokļu politikā
tiek konceptuāli atbalstītas, tomēr deva \
nedēļu laika Ekonomikas ministrijai, Finanšu ministrijai un
Labklājības ministrijai, lai ar vienotu \
pozīciju atgrieztos pie jautājuma izskatīšanas.
```

توجه: شکافهای خط مشخص شده با یک backslash فقط برای اهداف قالب بندی درج شده اند و نباید در داده های آموزش گنجانده شود.

توجه: line break مشخص شده با یک backslash فقط برای اهداف قالب بندی درج شده اند و نباید در داده های train گنجانده شود.

Training API:

درزیر نحوه آموزش مدل را به وسیله API نشان داده شده:

```
InputStreamFactory inputStreamFactory = new
MarkableFileInputStreamFactory(new File("corpus.txt"));

ObjectStream<String> lineStream =
  new PlainTextByLineStream(inputStreamFactory, StandardCharsets.UTF_8);
ObjectStream<LanguageSample> sampleStream = new
LanguageDetectorSampleStream(lineStream);

TrainingParameters params = ModelUtil.createDefaultTrainingParameters();
params.put(TrainingParameters.ALGORITHM_PARAM,
  PerceptronTrainer.PERCEPTRON_VALUE);
params.put(TrainingParameters.CUTOFF_PARAM, 0);

LanguageDetectorFactory factory = new LanguageDetectorFactory();

LanguageDetectorModel model = LanguageDetectorME.train(sampleStream, params,
factory);
model.serialize(new File("langdetect.bin"));
}
```

که در اینجا corpus.txt را میگیریم و خروجی مدل langdetect.bin است.

Sentence Detector:

OpenNLP Sentence Detector می تواند تشخیص دهد که یک علامت نگارشی پایان یک جمله را علامت گذاری می کند یا خیر. در این مفهوم یک جمله به عنوان طولانی ترین دنباله شخصیت کوتاه شده فضای سفید بین دو علامت نقطه گذاری تعریف می شود. جمله اول و آخر از این قاعده مستثنی است. فرض بر این است که اولین شخصیت غیر فضای سفید شروع یک جمله است و آخرین شخصیت غیر فضای سفید انتهای یک جمله فرض می شود. متن نمونه زیر باید به جملات آن تقسیم شود.

```
Pierre Vinken, 61 years old, will join the board as a nonexecutive director
Nov. 29. Mr. Vinken is
```

```
chairman of Elsevier N.V., the Dutch publishing group. Rudolph Agnew, 55
years
old and former chairman of Consolidated Gold Fields PLC, was named a director
of this
British industrial conglomerate.
```

بعد از تشخیص مرزهای جمله ، هر جمله به خط خودش نوشته می شود.

```
Pierre Vinken, 61 years old, will join the board as a nonexecutive director
Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
Rudolph Agnew, 55 years old and former chairman of Consolidated Gold Fields
PLC,
    was named a director of this British industrial conglomerate.
```

معمولاً Sentence Detect قبل از نشانه گذاری متن انجام می شود و این روش آموزش مدل های از پیش آموزش شده در وب سایت است ، اما همچنین می توان ابتدا نمادین آن را انجام دهید و اجازه دهید Sentence Detector متن موردنظر را پردازش کند. آشکارساز احکام OpenNLP نمی تواند مرزهای جمله را بر اساس محتوای جمله تعریف کند. یک مثال برجسته اولین جمله در مقاله ای است که در آن عنوان؛ به اشتباه در اولین قسمت جمله اول مشخص می شود. بیشتر مؤلفه های OpenNLP از ورودی انتظار دارند که به جملات تقسیم می شود.

ابزار Sentence Detection:

ساده ترین روش برای آزمایش Sentence Detector ابزار خط فرمان است. این ابزار فقط برای نمایش و آزمایش در نظر گرفته شده است. مدل ردیاب جمله انگلیسی را بارگیری کنید و با این دستور ابزار آشکارساز جمله را شروع کنید:

```
$ opennlp SentenceDetector en-sent.bin
```

فقط متن نمونه را از بالا به کنسول کپی کنید. Sentence Detector آن را خوانده و از یک خط در هر خط به کنسول تکرار می شود. معمولاً ورودی از یک پرونده خوانده می شود و خروجی به پرونده دیگری هدایت می شود. این امر با دستور زیر قابل دستیابی است.

```
$ opennlp SentenceDetector en-sent.bin < input.txt > output.txt
```

درمثال زیر را در خط فرمان در سیستم عامل لینوکس توضیع CentOS اجرا کردیم:

```
[ali@localhost bin]$ ./opennlp SentenceDetector en-sent.bin < /home/ali/Desktop/ali.txt
```

که با اجرای این مدل به خروجی زیر میرسیم:

```
Loading Sentence Detector model ... done (0.137s)
Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
Rudolph Agnew, 55 years
old and former chairman of Consolidated Gold Fields PLC, was named a director of this
British industrial conglomerate.



Average: 750.0 sent/s
Total: 3 sent
Runtime: 0.004s
```

Sentence Detection API:

Sentence Detector را می توان به راحتی از طریق API در یک برنامه ادغام کرد. برای Sentence Detection، ابتدا باید مدل جمله بارگیری شود.

```
try (InputStream modelIn = new FileInputStream("en-sent.bin")) {
  SentenceModel model = new SentenceModel(modelIn);
}
```

بعد از لود کردن مدل (که در بالا en-sent.bin است) از SentenceDetectorME شی می سازیم:

```
SentenceDetectorME sentenceDetector = new SentenceDetectorME(model);
```

Sentence Detector یک آرایه ای از string را به خروجی تحویل می دهد. که هر string یک جمله می باشد.

```
String sentences[] = sentenceDetector.sentDetect("  First sentence. Second
sentence. ");
```

خروجی آرایه حاوی دو ورودی است اولین string First sentence و دومین string Second sentence است و whitespace قبل ، بین و بعد از رشته رشته حذف می شود.API همچنین روشی را ارائه می دهد که به سادگی طول جمله را در رشته ورودی برمی گرداند.

```
Span sentences[] = sentenceDetector.sentPosDetect("  First sentence. Second
sentence. ");
```

در مثال زیر در Intelij عملکرد های متد های بالا را بررسی میکنیم

در داخل کلاس SimpleNLP داریم:

```java
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import opennlp.tools.tokenize.*;
import opennlp.tools.util.ObjectStream;
import opennlp.tools.util.PlainTextByLineStream;
import com.sun.media.sound.InvalidFormatException;
import opennlp.tools.util.Span;

import java.io.*;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;


public class SimpleNLP {
    //Sentence Detector
    public void SentenceDetect(int sentence_number) throws InvalidFormatException,
            IOException {
```

```java
String paragraph = "Deep learning methods employ multiple processing layers to learn hierarchical representations of data," +
        " and have produced state-of-the-art results in many domains. Recently," +
        " a variety of model designs and methods have blossomed in the context of natural language processing (NLP)." +
        " In this paper, we review significant deep learning related models and methods that have been employed for numerous " +
        "NLP tasks and provide a walk-through of their evolution." +
        " We also summarize, compare and contrast the various models and put forward a detailed understanding of the past," +
        " present and future of deep learning in NLP.";
```

```java
// always start with a model, a model is learned from training data
InputStream is = new FileInputStream("D:[redacted]\\Simple OpenNLP Project\\src\\BinFiles\\en-sent.bin");
SentenceModel model = new SentenceModel(is);
SentenceDetectorME sdetector = new SentenceDetectorME(model);

String sentences[] = sdetector.sentDetect(paragraph);
//returns the span of the sentence
Span sentences1[] = sdetector.sentPosDetect(paragraph);
System.out.println(sentences[sentence_number]);
System.out.println("Span of sentence is : " + sentences1[sentence_number]);
is.close();
```

در کلاس main داریم:

```java
package com.company;

import java.io.IOException;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws Exception {
        SimpleNLP nlp = new SimpleNLP();
```

```java
        System.out.println("Sentence Detection: 1" );

        System.out.println("Enter a number of element in list that you want: ");
        Scanner listnuminput = new Scanner(System.in);
        int listnum = listnuminput.nextInt();

            if (listnum == 1) {
                System.out.println("Enter a number of sentence you want: ");
                Scanner scan = new Scanner(System.in);
                int num = scan.nextInt();
                nlp.SentenceDetect(num);
            }


        else
            {
            System.out.println("Your number is out of the list");
            }
    }
}
```

خروجی برنامه زیر به شکل زیر است:

```
"C:\Program Files\Java\jdk1.8.0_40\bin\java.exe" ...
Sentence Detection: 1
Enter a number of element in list that you want:
1
Enter a number of sentence you want:
0
Deep learning methods employ multiple processing layers to learn hierarchical representations of data, and have produced state-of-the-art results in many domains.
Span of sentence is : [0..162)

Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk1.8.0_40\bin\java.exe" ...
Sentence Detection: 1
Enter a number of element in list that you want:
1
Enter a number of sentence you want:
1
Recently, a variety of model designs and methods have blossomed in the context of natural language processing (NLP).
Span of sentence is : [163..279)

Process finished with exit code 0
```

آرایه نتیجه دوباره شامل دو ورودی است. ورودی اول موجود در شاخص 2 و در 17 به پایان می رسد. ورودی دوم از 18 شروع می شود و در 34 پایان می یابد. از روش ابزار Span.getCoveredText می توان برای ایجاد یک بستر استفاده کرد که فقط نواحی موجود در دهانه را پوشش می دهد.

Training API:

Sentence Detector همچنین یک API را برای آموزش یک مدل کشف جملات جدید ارائه می دهد. اصولاً سه مرحله برای آموزش آن لازم است:

- برنامه باید یک نمونه data stream باز شود.
- متد SentenceDetectorME.train فراخوانی شود.
- SentenceModel را در یک فایل ذخیره یا به طور مستقیم از آن استفاده کنید

مثال زیر نحوه اجرای این مراحل را نشان می دهد:

```
ObjectStream<String> lineStream =
  new PlainTextByLineStream(new FileInputStream("en-sent.train"),
StandardCharsets.UTF_8);

SentenceModel model;

try (ObjectStream<SentenceSample> sampleStream = new
SentenceSampleStream(lineStream)) {
  model = SentenceDetectorME.train("en", sampleStream, true, null,
TrainingParameters.defaultParams());
}

try (OutputStream modelOut = new BufferedOutputStream(new
FileOutputStream(modelFile))) {
  model.serialize(modelOut);
}
```

ارزیابی

ابزار ارزیابی:

این دستور نشان می دهد که چگونه می توان ابزار ارزیاب را اجرا کرد:

```
$ opennlp SentenceDetectorEvaluator -model en-sent.bin -data en-sent.eval -
encoding UTF-8

Loading model ... done
Evaluating ... done

Precision: 0.9465737514518002
Recall: 0.9095982142857143
F-Measure: 0.9277177006260672
```

فایل en-sent.eval با فرمت داده های آموزش یکسان است .

Tokenization:

Tokenizerهای OpenNLP توالی کاراکتر ورودی را به نشانه ها تقسیم می کنند. نشانه ها معمولاً کلمات ، نقطه گذاری ، اعداد و غیره هستند

```
Pierre Vinken, 61 years old, will join the board as a nonexecutive director
Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
Rudolph Agnew, 55 years old and former chairman of Consolidated Gold Fields
    PLC, was named a director of this British industrial conglomerate.
```

نتیجه زیر tokenهای فردی را در یک نمای جدا whitespace نشان می دهد.

```
Pierre Vinken , 61 years old , will join the board as a nonexecutive director
Nov. 29 .
Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group .
Rudolph Agnew , 55 years old and former chairman of Consolidated Gold Fields
PLC ,
    was named a nonexecutive director of this British industrial conglomerate
.
A form of asbestos once used to make Kent cigarette filters has caused a high
    percentage of cancer deaths among a group of workers exposed to it more
than 30 years ago ,
    researchers reported .
```

OpenNLPپیاده سازی چندین نشانه گذاری ارائه می دهد:

- Whitespace Tokenizer– یک whitespace Tokenizer ، از توالی های بدون whitespace به عنوان Token مشخص می شوند
- Simple Tokenizer – یک نشانه گذاری کلاس کاراکتر ، توالی های کلاس کارارتر یکسان هستند
- Learnable Tokenizer - یک حداکثر Tokenizer آنتروپی ، مرزهای Token را بر اساس مدل احتمال تشخیص می دهد

اکثر part of speech tagger ها، parser ها و غیره ، با متن هایی که به این روش نشانه گذاری شده اند کار می کنند. این مهم است که اطمینان حاصل کنید که Tokenizerشما نشانه ای از نوع مورد انتظار توسط اجزای پردازش متن بعدی شما را تولید می کند.

با OpenNLP (مانند بسیاری از سیستم ها) ، علامت گذاری یک فرایند دو مرحله ای است: ابتدا مرزهای جمله مشخص می شوند ، سپس نشانه ها در هر جمله مشخص می شوند.

ابزار Tokenizer:

ساده ترین روش برای امتحان کردن Tokenizerها ابزارهای خط فرمان هستند. ابزارها فقط برای نمایش و آزمایش در نظر گرفته شده اند

دو ابزار برای Tokenize کردن وجود دارد ، یکی برای Simple Tokenizer و دیگری برای tokenizer قابل یادگیری. ابزار خط فرمان برای Whitespace Tokenizer وجود ندارد ، زیرا خروجی جدا شده از فضای سفید با ورودی یکسان است.

دستور زیر نحوه استفاده از ابزار Simple Tokenizer را نشان می دهد:

```
$ opennlp SimpleTokenizer
```

برای استفاده از learnable Tokenizer، مدل توکن های انگلیسی را از وب سایت https://opennlp.apache.org/ بارگیری کنید:

```
$ opennlp TokenizerME en-token.bin
```

برای تست Tokenizer نمونه را از بالا به کنسول کپی کنید. نشانه های جدا از فضای سفید به کنسول ارسال می شود.

معمولاً ورودی از یک file خوانده می شود و برای یکfile نوشته می شود.

```
$ opennlp TokenizerME en-token.bin < article.txt > article-tokenized.txt
```

درمثال زیر را در خط فرمان در سیستم عامل لینوکس توضیع CentOS اجرا کردیم:

```
[ali@localhost bin]$ ./opennlp TokenizerME  en-token.bin < /home/ali/Desktop/ali.txt >/home/ali/Desktop/ali1.txt
Loading Tokenizer model ... done (0.225s)


Average: 444.4 sent/s
Total: 4 sent
Runtime: 0.009s
Execution time: 0.428 seconds
```

متن اصلی در ali.txt به صورت زیر است:

```
Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group. Rudolph Agnew, 55 years
old and former chairman of Consolidated Gold Fields PLC, was named a director of this
British industrial conglomerate.
```

پس از اعمال دستور بالا در لینوکس به شکل زیر در می آید:

```
  GNU nano 2.3.1                                    File: ali1.txt

Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group . Rudolph Agnew , 55 years
old and former chairman of Consolidated Gold Fields PLC , was named a director of this
British industrial conglomerate .
```

همان طور که مشاهده می کنید کلمات و علامت های نگارشی از هم به وسیله tab از هم جدا شده اند که به هر قسمت یک Token میگویند.

این کار به همین روش برایSimple Tokenizer قابل انجام است.

Tokenizer API:

Tokenizer را می توان با API تعریف شده در یک برنامه ادغام کرد. نمونه مشترک WhitespaceTokenizer را می توان از یک زمینه ایستا WhitespaceTokenizer.INSTANCE بازیابی کرد. نمونه مشترک SimpleTokenizer را می توان به همان روش از SimpleTokenizer.INSTANCE بازیابی کرد. برای معرفی سریع TokenizerME (نشانه قابل یادگیری) ابتدا باید یک مدل توکن ایجاد شود. نمونه کد زیر نحوه بارگیری یک مدل را نشان می دهد.

```
try (InputStream modelIn = new FileInputStream("en-token.bin")) {
  TokenizerModel model = new TokenizerModel(modelIn);
}
```

پس از لود شدن مدل ، از TokenizerME می توان شی ساخت:

```
Tokenizer tokenizer = new TokenizerME(model);
```

Tokenizer دو روش رمزگذاری را ارائه می دهد ، هر دو از یک شی ورودی رشته ای انتظار می رود که حاوی متن Tokenize نشده باشد. در صورت امکان یک جمله باید باشد ، اما بسته به آموزش نشانه قابل یادگیری این مورد لازم نیست. اولین مجموعه ای از رشته ها را برمی گرداند ، جایی که هر رشته یک علامت است.

```
String tokens[] = tokenizer.tokenize("An input sample sentence.");
```

خروجی با این Token ها آرایه ای خواهد بود.

```
"An", "input", "sample", "sentence", "."
```

در روش دوم ، tokenizePos آرایه ای از Spans را برمی گرداند ، هر Span حاوی مجموعه های شروع و انتهای کاراکتر Token در ورودی رشته است.

```
    Span tokenSpans[] = tokenizer.tokenizePos("An input sample sentence.");
```

آرایه tokenSpans اکنون حاوی 5 عنصر است. برای به دست آوردن برای گرفتن متن برای span Span.getCoveredText که متن و span را از ورودی میگیرد فرا بخوانید . TokenizerME قادر است خروجی های محتمل برای Token های شناسایی شده را به دست آورد. متد getTokenProbables باید مستقیماً پس از متد tokenize فراخوانی شود.

```
TokenizerME tokenizer = ...

String tokens[] = tokenizer.tokenize(...);
double tokenProbs[] = tokenizer.getTokenProbabilities();
```

آرایه tokenProbs اکنون حاوی یک مقدار مضاعف برای هر توکن است ، مقدار بین 0 تا 1 است ، در جایی که 1 بالاترین احتمال ممکن و 0 کمترین احتمال ممکن است.

در مثال زیر tokenization را روی متن داده شده انجام می دهیم:

در کلاس SimpleNLP داریم:

```java
public void Tokenize() throws InvalidFormatException, IOException {
    InputStream is = new FileInputStream( name: "D:\\Java litec\\Projects\\Simple OpenNLP Project\\src\\BinFiles\\en-token.bin");

    TokenizerModel model = new TokenizerModel(is);

    Tokenizer tokenizer = new TokenizerME(model);
    // Scanner input = new Scanner(System.in);

    String tokens[] = tokenizer.tokenize( s: "Deep learning methods employ multiple processing layers to learn hierarchical" +
            " representations of data, and have produced state-of-the-art results in many domains. Recently," +
            " a variety of model designs and methods have blossomed in the context of natural language processing (NLP). " +
            "In this paper, we review significant deep learning related models and methods that have been employed for numerous NLP t
            " compare and contrast the various models and put forward a detailed understanding of the past, present and future of dee

    for (String a : tokens)
        System.out.println(a);

    System.out.println("Detokenize\n");


    is.close();
}
```

در کلاس main داریم:

```java
if (listnum == 2) {
    nlp.Tokenize();
}
```

خروجی آن به شکل زیر است:

```
"C:\Program Files\Java\jdk1.8.0_40\bin\java.exe" ...
Sentence Detection: 1
Word detection using Token : 2
 Detokenize : 3
Enter a number of element in list that you want:
2
Deep
learning
methods
employ
multiple
processing
layers
to
learn
hierarchical
representations
of
data
,
and
have
produced
state-of-the-art
results
```

```
in
many
domains
.
Recently
,
a
variety
of
model
designs
and
methods
have
blossomed
in
the
context
of
natural
language
processing
(
NLP)
```

```
.
In
this
paper
,
we
review
significant
deep
learning
related
models
and
methods
that
have
been
employed
for
numerous
NLP
tasks
and
provide
```

```
a
walk-through
of
their
evolution
.
We
also
summarize
,
compare
and
contrast
the
various
models
and
put
forward
a
detailed
understanding
of
the
```

```
past
,
present
and
future
of
deep
learning
in
NLP.
Detokenize

Your number is out of the list

Process finished with exit code 0
```

Training API:

Tokenizer ها یک API برای مدل جدید Tokenization ارایه می دهد. اصولاً سه مرحله برای آموزش آن لازم است:

- برنامه باید یک datastream نمونه را باز کند
- به وسیله فرا خوانی متد TokenizerME.train
- TokenizerModel را در یک file ذخیره کنید یا مستقیماً از آن استفاده کنید

این مراحل را در نمونه کد های زیر می بینید:

```
ObjectStream<String> lineStream = new PlainTextByLineStream(new
FileInputStream("en-sent.train"),
    StandardCharsets.UTF_8);
ObjectStream<TokenSample> sampleStream = new TokenSampleStream(lineStream);

TokenizerModel model;

try {
  model = TokenizerME.train("en", sampleStream, true,
TrainingParameters.defaultParams());
}
finally {
  sampleStream.close();
}

OutputStream modelOut = null;
try {
  modelOut = new BufferedOutputStream(new FileOutputStream(modelFile));
  model.serialize(modelOut);
} finally {
  if (modelOut != null)
     modelOut.close();
}
```

Detokenizing:

Detokenizationدرست برعکس Tokenizationاست ، رشته اصلی بدون Tokenization باید خارج از توالی ساختار یک Token ساخته شود . پیاده سازی OpenNLP برای از بین بردن Tokenizationدر داده آمزشی Tokenizerایجاد شده است.

پیاده سازی کاملاً مبتنی بر قاعده است و چگونگی Tokenها را باید تعریف کند که چگونه Token باید به جمله به صورت کاراکتری متصل شود .

فرهنگ لغت قانون یک عملی را به هر Token اختصاص می دهد که توصیف می کند چگونه باید آن را به یک دنباله کاراکتر وصل کرد.

قوانین زیر را می توان به یک Token اختصاص داد:

- MERGE_TO_LEFT - نشانه را به سمت چپ ادغام می کند
- MERGE_TO_RIGHT –نشانه را به سمت راست ادغام می کند
- RIGHT_LEFT_MATCHING – ادغام نشانه به سمت راست در وقوع اول و ادغام نشانه به سمت چپ در دومین وقوع

نمونه زیر چگونگی Detokenization با فرهنگ نامه کوچک قانون را نشان می دهد (فرمت خروجی، نه فرمت داده xml):

```
. MERGE_TO_LEFT
" RIGHT_LEFT_MATCHING
```

این فرهنگ لغت باید برای Tokenizationمجدد جمله‌ای که در whitespace صورت گرفته است ، استفاده شود:

```
He said " This is a test " .
```

Tokenها این tagها را بر اساس فرهنگ لغت دریافت می کنند:

```
  He -> NO_OPERATION
said -> NO_OPERATION
 " -> MERGE_TO_RIGHT
This -> NO_OPERATION
  is -> NO_OPERATION
   a -> NO_OPERATION
test -> NO_OPERATION
  " -> MERGE_TO_LEFT
  . -> MERGE_TO_LEFT
```

که منجر به دنباله کارکاتر های زیر می شود:

```
He said "This is a test".
```

Detokenizing API:

این قسمت documentation توسطه apache ارایه نشده ومن روی مثال هایی در این باره توضیح می دهم:

Detokenizer Dictionary:

در کلاس SimpleNLP داریم:

```java
public static String detokenize(String[] tokens, DetokenizationDictionary.Operation operation) {
    DetokenizationDictionary.Operation[] operations = new DetokenizationDictionary.Operation[tokens.length];

    for (int i = 0; i < tokens.length; i++) {
        operations[i] = operation;
    }

    DetokenizationDictionary dictionary = new DetokenizationDictionary(
            tokens, operations);
    DictionaryDetokenizer detokenizer = new DictionaryDetokenizer(
            dictionary);

    return detokenizer.detokenize(tokens, splitMarker: " ");
}
```

در کلاس main داریم:

```
if(listnum ==3){



    String tokens[] = {"Deep learning " ,"methods" ,"employ" ,"multiple"," processing"," layers"," to" ,"learn hierarchical",
            " representations", "of"," data", "and ","have", "produced", "state-of-the-art", "results", "in"," many"," domains",".",}


        String data =nlp.detokenize(tokens, DetokenizationDictionary.Operation.MOVE_LEFT);
        System.out.println(data);
}
```

و در خروجی داریم:

```
"C:\Program Files\Java\jdk1.8.0_40\bin\java.exe" ...
Sentence Detection: 1
Word detection using Token : 2
 Detokenize : 3
Enter a number of element in list that you want:
3
Deep learning  methods employ multiple  processing  layers  to learn hierarchical  representations of  data and  have produced state-of-the-art results in  many  domains .

Process finished with exit code 0
```

:NameFinder

Name Finder می تواند مولفه های نامی و اعداد را در متن تشخیص دهد. برای تشخیص مولفه متنی Name Finder نیاز به مدل دارد. مدل به نوع زبان و مولفه ای که برای آن آموزش داده شده است بستگی دارد.

پروژه های OpenNLP تعدادی Name Finderاز قبل آموزش دیده را ارائه می دهد مدل های آموزش داده شده ای که در شرکت های مختلف آزادانه در دسترس آموزش دیده است. آنها را می توانید در صفحه بارگیری مدل ما بارگیری کنید.

برای پیدا کردن نام در متن خام متن باید به Tokenها و جملات تقسیم شود. توضیحات مفصل در قسمت Tokenizer, Sentence Detectorآورده شده است . آموزش Tokenizerمهم است چرا که رمزگذاری داده های آموزش و متن ورودی باید یکسان باشد.

:Name Finder Tool

آسان ترین روش برای تست Name finder استفاده از CLI است. این ابزار تنها برای تایید و تست مورد استفاده قرار می گیرد.

English model person را دانلود کنید و با این دستور با ابزار Name finder کار کنید.

```
$ opennlp TokenNameFinder en-ner-person.bin
```

در حال حاظر Name Finder متن tokenize شده را خط به خط ازطریق stdin (standard input) دریافت می کند. خطوط خالی اشاره به انتهای متن در اسناد ما دارد و مولد های ویژگی های منطبق را دوباره مقدار دهی می کند. متن زیر را در ترمینال کپی و پیست کنید:

```
Pierre Vinken , 61 years old , will join the board as a nonexecutive director
Nov. 29 .
     Mr . Vinken is chairman of Elsevier N.V. , the Dutch publishing group .
Rudolph Agnew , 55 years old and former chairman of Consolidated Gold Fields
PLC , was named
a director of this British industrial conglomerate .
```

در این مرحله Name finder اسامی اشخاص را در خروجی مشخص می کند:

```
<START:person> Pierre Vinken <END> , 61 years old , will join the board as a
nonexecutive director Nov. 29 .
Mr . <START:person> Vinken <END> is chairman of Elsevier N.V. , the Dutch
publishing group .
<START:person> Rudolph Agnew <END> , 55 years old and former chairman of
Consolidated Gold Fields PLC ,
was named a director of this British industrial conglomerate .
```

تست زیر را در لینوکس گرفته شده است:

```
[ali@localhost bin]$ ./opennlp TokenNameFinder en-ner-person.bin < /home/ali/Desktop/ali.txt
```

و خروجی آن به شکل زیر است:

```
Loading Token Name Finder model ... done (1.576s)
<START:person> Pierre Vinken, <END> 61 years old, will join the board as a nonexecutive director Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group. <START:person> Rudolph Agnew, <END> 55 years
old and former chairman of Consolidated Gold Fields PLC, was named a director of this
British industrial conglomerate.


Average: 32.5 sent/s
Total: 4 sent
Runtime: 0.123s
Execution time: 2.011 seconds
```

Name Finder API:

برای استفاده از Name finder در سطح production به شدت توصیه می شود از آن به جای استفاده داخل CLI از آن داخل برنامه هایتان استفاده کنید. در گام نخست مدل Name finder با از دیسک یا دیگر منابع داخل مموری بارگزاری می شود. در مثال زیر از دیسک بارگیری می کنیم:

```
try (InputStream modelIn = new FileInputStream("en-ner-person.bin")){
model = new TokenNameFinderModel(modelIn); TokenNameFinderModel
}
```

دلایلی وجود دارد که بارگزاری مدل با مشکل روبرو شود:

- مشکل در لایه های زیرین ورودی و خروجی(I/O)
- منطبق نبودن ورژن مدل و OpenNLP با یک دیگر
- بارگزاری اشتباه مدل با کامپوننت اشتباه به طور مثال مدل tokenizer با کلاس TokenizerNameFinderModel بارگزاری شود.
- معتبر نبودن محتوای مدل به دلایل دیگر

بعد از بارگزاری مدل از کلاس NameFinderME می توانید شیء بسازید.

```
NameFinderME nameFinder = new NameFinderME(model);
```

شی ء سازی در این قسمت تمام شده و Name Finder قابل استفاده است. Name Finder به صورت thread safe نیست و فقط باید توسط یک thread فراخوانی شود.

```
for (String document[][] : documents) {
        for (String[] sentence : document) {
                Span nameSpans[] = nameFinder.find(sentence);
                // do something with the names
        }
        nameFinder.clearAdaptiveData()
}
```

تیکه تیکه شده های زیر متنی را برای متد find فراهم می کند:

```
String sentence[] = new String[]{
"Pierre",
"Vinken",
"is",
"61",
"years"
"old",
"."
};
Span nameSpans[] = nameFinder.find(sentence);
```

آرایه های nameSpans اکنون دقیقاً یک Span دارند که نام پیر وینکن را نشان می دهد. عناصر بین شروع و پایان جبران خاتمه tokenهای نام هستند. در این حالت مقدار شروع 0 وافست انتهایی2. شی Span نیز نوع آن موجدیت را می داند.

در این حالت فرد است(که توسط مدل تعریف شده است). می توان با فراخوانی Span.getType () بازیابی شود. علاوه بر

نام آماری finder ، OpenNLP همچنین یک Dictionaryو پیاده سازی عبارات منظم درName finder را به شما ارائه می دهد

Training API:

برای آموزش یافتن Name finderاز درون یک برنامه ، توصیه می شود به جای ابزار خط فرمان از API آموزش استفاده کنید.

اصولاً سه مرحله برای آموزش آن لازم است:

- برنامه باید یک data stream نمونه را باز کند
- با فراخوانی متد NameFinderME.train
- TokenNameFinderModel را در یک فایل ذخیره کنید

سه مرحله با کد نمونه زیر نشان داده شده است:

```
ObjectStream<String> lineStream =
new PlainTextByLineStream(new FileInputStream("en-ner-person.train"),
StandardCharsets.UTF_8);
TokenNameFinderModel model;
try (ObjectStream<NameSample> sampleStream = new
NameSampleDataStream(lineStream)) {
model = NameFinderME.train("en", "person", sampleStream,
TrainingParameters.defaultParams(),
TokenNameFinderFactory nameFinderFactory);
}
try (modelOut = new BufferedOutputStream(new FileOutputStream(modelFile)){
model.serialize(modelOut);
}
```

Custom Feature Generation:

OpenNLP یک نسل ویژگی پیش فرض را تعریف می کند که در صورت مشخص شدن هیچ ویژگی از ویژگی های سفارشی ، مورد استفاده قرار نمی گیرد. کاربرانی که می خواهند آزمایش با تولید ویژگی می تواند یک مولد ویژگی سفارشی را ارائه دهد. از طریق API یا از طریق یک فایل توصیف کننده xml.

Evaluation:

ارزیابی ساخته شده می تواند عملکرد شناسایی موجودیت نام یاب را اندازه گیری کند. عملکرد بر روی مجموعه داده های آزمون یا از طریق اعتبار متقاطع اندازه گیری می شود.

Evaluation Tool:

دستور زیر نحوه اجرای ابزار را نشان می دهد:

```
$ opennlp TokenNameFinderEvaluator -model en-ner-person.bin -data en-ner-
person.test -encoding UTF-8

Precision: 0.8005071889818507
Recall: 0.7450581122145297
F-Measure: 0.7717879983140168
```

توجه: رابط خط فرمان از ارزیابی متقابل در نسخه فعلی پشتیبانی نمی کند.

Evaluation API:

ارزیابی می تواند بر روی یک مدل از قبل آموزش دیده و یک مجموعه داده تست یا از طریق اعتبار سنجی متقابل انجام شود. در حالت اول ، مدل باید بارگیری شود و باید یک NameSample ObjectStream ایجاد شود (به نمونه کد در بالا مراجعه کنید) ، با فرض اینکه این دو شی وجود داشته باشند ، کد زیر نحوه انجام ارزیابی را نشان می دهد:

```
TokenNameFinderEvaluator evaluator = new TokenNameFinderEvaluator(new
NameFinderME(model));
evaluator.evaluate(sampleStream);

FMeasure result = evaluator.getFMeasure();

System.out.println(result.toString());
```

در مورد اعتبار سنجی متقابل ، باید تمام استدلال های آموزش ارائه شود (به بخش API Training فوق مراجعه کنید). برای انجام

اعتبار سنجی متقارن ObjectStream باید تغییر مکان داده شود.

Document Categorizer:

Classifying:

Document Categorizer, در OpenNLP می تواند متن را به دسته های از پیش تعریف شده طبقه بندی کند.
این در چارچوب حداکثر آنتروپی استوار است. برای شخصی که علاقه مند به Gross Margin است ، متن نمونه مندرج در زیر می تواند به عنوان GMDecrease طبقه بندی شود

```
   Major acquisitions that have a lower gross margin than the existing network
```

```
also had a negative impact on the overall gross margin, but it should improve
following the implementation of its integration strategies.
```

و متن زیر می تواند به عنوان GMIncrease طبقه بندی شود:

```
The upward movement of gross margin resulted from amounts pursuant to
adjustments to obligations towards dealers.
```

برای طبقه بندی متن ، Document Categorizer به یک مدل نیاز دارد. طبقه بندی ها دارای الزامات خاص هستند و از این رو هیچ مدل از پیش ساخته شده برای طبقه بندی اسناد در پروژه OpenNLP وجود ندارد.

Document Categorizer Tool:

ساده ترین راه برای امتحان دسته بندی اسناد ابزار خط فرمان(CLI) است. این ابزار فقط برای نمایش و آزمایش در نظر گرفته شده است. دستور زیر نحوه استفاده از ابزار Document Categorizer را نشان می دهد.

```
$ opennlp Doccat model
```

ورودی از ورودی استاندارد خوانده می شود و خروجی به خروجی استاندارد نوشته می شود ، مگر اینکه آنها هدایت شوند. طبق بسیاری از مؤلفه های OpenNLP ، طبقه بندی کننده اسناد انتظار ورودی را دارد که به جملات تقسیم می شود.

Document Categorizer API:

برای انجام طبقه بندی ، به یک Maxent model نیاز خواهید داشت . اینها در کلاس DoccatModel از ابزارهای OpenNLP محصور شده اند. ابتدا باید بایت ها را از مدل serialized شده در InputStream بگیرید - ما این کار را برای شما انجام می دهیم ، زیرا شما کسی بودید که برای شروع به کار آن را serialized کردید. اکنون برای قسمت آسان:

```
InputStream is = ...
DoccatModel m = new DoccatModel(is);
```

با DoccatModel موجود که فقط در اینجا وجود دارد:

```
String inputText = ...
DocumentCategorizerME myCategorizer = new DocumentCategorizerME(m);
double[] outcomes = myCategorizer.categorize(inputText);
String category = myCategorizer.getBestCategory(outcomes);
```

Training:

Document Categorizer را می توان در مورد مطالب آموزشی حاشیه نویسی آموزش داد. داده ها می توانند در قالب آموزش Document Categorizer در OpenNLP باشند. این یک سند در هر خط است ، شامل طبقه بندی و متن جدا شده توسط یک فضای سفید هستند. سایر قالب ها نیز می توانند در دسترس باشند. نمونه زیر نمونه بالا را با فرمت مورد نیاز نشان می دهد. در اینجا GMDecrease و GMIncrease دسته بندی می شوند.

```
GMDecrease Major acquisitions that have a lower gross margin than the
existing network also \
had a negative impact on the overall gross margin, but it should
improve following \
the implementation of its integration strategies .
GMIncrease The upward movement of gross margin resulted from amounts pursuant
to adjustments \
```

```
to obligations towards dealers .
```

توجه: line break مشخص شده با یک backslash فقط برای اهداف قالب بندی درج شده اند و نباید در داده های آموزش گنجانده شود.

Training Tool:

دستور زیر Document Categorizer را آموزش داده و مدل را برای en-doccat.bin می نویسد:

```
$ opennlp DoccatTrainer -model en-doccat.bin -lang en -data en-doccat.train
encoding UTF-8 -
```

علاوه بر این می توان تعداد تکرارها و قطع آن را مشخص کرد.

Training API:

بنابراین ، به طور طبیعی برای آموزش مدل خود به بسیاری از رویدادهای از پیش طبقه بندی شده دسترسی خواهید داشت.

کلاس opennlp.tools.doccat.DocumentSample یک سند متنی و طبقه بندی آن را کپسوله می کند.

DocumentSample دارای دو سازنده است. هر یک از متن را به عنوان یک آرگومان در نظر می گیرند. آرگومان دیگر می تواند متن خام یا آرایه ای از token ها باشد.
به طور پیش فرض ، متن خام توسط فضای سفید به token ها تقسیم می شود. بنابراین ، بیایید بگوییم که اطلاعات آموزش شما در یک فایل متنی موجود است ، جایی که فرمت به شرح فوق است. سپس ممکن است بخواهید چیزی شبیه به این را بنویسید تا مجموعه DocumentSamples را ایجاد کنید:

```
DoccatModel model = null;
InputStream dataIn = null;

try (dataIn = new FileInputStream("en-sentiment.train")) {
ObjectStream<String> lineStream =
new PlainTextByLineStream(dataIn, StandardCharsets.UTF_8);
ObjectStream<DocumentSample> sampleStream = new
DocumentSampleStream(lineStream);

model = DocumentCategorizerME.train("en", sampleStream);
}
```

اکنون ممکن است زمان مناسبی برای بررسی Hulu یا موارد مشابه باشد ، زیرا اگر یک مجموعه آموزشی بزرگ داشته باشید ، این ممکن است مدتی طول بکشد. ممکن است خروجی زیادی نیز ببینید. پس از اتمام کار ، می توانید خیلی سریع به طبقه بندی مستقیم بروید ، اما ابتدا سریال سازی را پوشش خواهیم داد.

```
try (OutputStream modelOut = new BufferedOutputStream(new
FileOutputStream(modelFile))) {
  model.serialize(modelOut);
}
```

: Part-of-Speech Tagger

:Tagging

Part of Speech Tagger نشانه ها را با نوع واژگان مربوط به خود و بر اساس متن آن مشخص می کند. بسته به نشان و متن ، یک نشان ممکن است دارای چندین pos tag باشد. OpenNLP POS Tagger از یک مدل احتمال استفاده می کند تا برچسب صحیح pos را از مجموعه tagها پیش بینی کند. برای محدود کردن برچسب های احتمالی برای یک نشانه از tag dictionary استفاده می شود که باعث افزایش tag زدن و عملکرد زمان اجرای برچسب می شود.

:POS Tagger Tool

ساده ترین روش برای آزمایش POS Tagger ابزار خط فرمان است. این ابزار فقط برای نمایش و آزمایش در نظر گرفته شده است. مدل انگلیسی maxent pos را بارگیری کنید و ابزار POS Tagger را با این دستور شروع کنید:

```
$ opennlp POSTagger en-pos-maxent.bin
```

POS Tagger اکنون جملات tokenize شده در هر خط از stdin را می خواند. این دو جمله را در کنسول کپی کنید:

```
Pierre Vinken , 61 years old , will join the board as a nonexecutive director
Nov. 29 .
Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group .
```

POS Tagger اکنون جملات را با pos tag ها به کنسول تکرار می کند:

```
Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_, will_MD join_VB the_DT
board_NN as_IN
a_DT nonexecutive_JJ director_NN Nov._NNP 29_CD ._.
Mr._NNP Vinken_NNP is_VBZ chairman_NN of_IN Elsevier_NNP N.V._NNP ,_, the_DT
Dutch_NNP publishing_VBG group_NN
```

مجموعه برچسب مورد استفاده توسط مدل pos انگلیسی ، مجموعه تگ Penn Treebank است

:POS Tagger API

POS Tagger می تواند از طریق API در یک برنامه تعبیه شود. ابتدا مدل pos باید از دیسک یا منبع دیگری در حافظه بارگیری شود. در نمونه زیر آن از دیسک بارگذاری شده است.

```
try (InputStream modelIn = new FileInputStream("en-pos-maxent.bin"){
POSModel model = new POSModel(modelIn);
}
```

پس از لود شدن مدل ، می توان از POSTaggerME شی ساخت.

```
POSTaggerME tagger = new POSTaggerME(model);
```

در حال حاظر شی ساخته شده توسط Pos Tagger برای تگ کردن اطلاعات آماده است. این جمله به عنوان جمله tokenize شده برای ورودی تحت عنوان رشته ای از آرایه در نظر گرفته می شود . هر شی ای از رشته به عنوان یک token شناخته می شود.

در کد زیر نحوه ی معین کردن دنباله ای از pos tag ها برای جمله در نظر گرفته شده نشان داده شده است:

```
String sent[] = new String[]{"Most", "large", "cities", "in", "the", "US",
"had",
```

```
"morning", "and", "afternoon", "newspapers",
"."};
String tags[] = tagger.tag(sent);
```

آرایه برچسب ها حاوی یک برچسب قسمت گفتار برای هر token در آرایه ورودی است. برچسب مربوطه را می توانید در همان index ای که به عنوان توکن در آرایه ورودی قرار دارد پیدا کنید. نمرات اطمینان برای برچسب های برگشت یافته را می توان به راحتی از POSTaggerME بازیابی کرد

با روش زیرفراخوانی کنید:

```
double probs[] = tagger.probs();
```

فراخوانی probs حالت پذیر است و همیشه احتمالات آخرین جمله tag خورده را برمی گرداند. فقط متد probs قبل از فراخوانی متد tag، فراخوانی شود ، در غیر این صورت رفتار تعریف نشده است.

برخی از برنامه ها نیاز به بازیابی توالی n-بهترین postag و نه تنها بهترین توالی دارند. روش topKSequences که قادر به بازگشت توالی های برتر است. می توان آن را به روشی مشابه تگ نامید.

```
Sequence topSequences[] = tagger.topKSequences(sent);
```

هر موضوع توالی شامل یک دنباله است. دنباله را می توان از طریق Sequence.getOutcomes() دریافت کرد که tag ها را در قالب array برمی گرداند و Sequence.getProbs () آرایه احتمال این دنباله را برمی گرداند.

:Training

Post tagger توسط یکسری از ابزار های معین شده قابل آموزش است. ابزار آموزش مجموعه ای از جملات tokenize شده است که هر token یک part-of-speech tag اختصاص داده شده دارد. ابزار اصلی آموزش Post tagger به صورت زیر است:

```
About_IN 10_CD Euro_NNP ,_, I_PRP reckon_VBP ._.
That_DT sounds_VBZ good_JJ ._.
```

هر جمله باید در یک خط باشد. جفت های token/tag به وسیله "_" به یک دیگر متصل می شوند. جفت های token/tag توسط فضای سفید از یکدیگر جدا شده اند. فرمت داده ها نمی تواند انتهای سند را مشخص کند.

اگر می خواهید اخر سند را مشخص کنید تصویه می شود خط خالی به عنوان انتهای سند در ابزار آموزش گنجانده شود.

Part-of-speech tagger را می تواند توسط CLI ویا training API آموزش داد.

:Training API

API آموزش Part-of-speech tage آموزش مدل جدید POS را پشتیبانی می کند.اساسا سه گام اصلی برای آموزش آن مورد نیاز است

- برنامه باید یک data stream نمونه را باز کند.
- متد POSTagger.train فراخوانی شود.
- ذخیره سازی PosModel در یک فایل

کد زیر نشان می دهد که:

```
POSModel model = null;
try (InputStream dataIn = new FileInputStream("en-pos.train")){
ObjectStream<String> lineStream = new PlainTextByLineStream(dataIn,
StandardCharsets.UTF_8);
ObjectStream<POSSample> sampleStream = new WordTagSampleStream(lineStream);
model = POSTaggerME.train("en", sampleStream,
TrainingParameters.defaultParams(), null, null);
}
```

کد بالا دو گام اول را انجام می دهد.باز کردن داده و آموزش مدل.مدل آموزش داده شده باید هنوز باید در outputstream ذخیره شود .در مثال زیر در یک فایل ذخیره شده است:

```
try (OutputStream modelOut = new BufferedOutputStream(new
FileOutputStream(modelFile))){
model.serialize(modelOut);
}
```

Tag Dictionary:

فرهنگ لغت برچسب یک فرهنگ لغت کلمه است که مشخص می کند کدام token برچسب خاص می تواند داشته باشد. استفاده از فرهنگ لغت برچسب دو تا مزایا دارد:

برچسب های نامناسب را نمی توان به token ها در فرهنگ لغت اختصاص داد و الگوریتم جستجوی پرتو باید کمتر در نظر بگیرد

امکان پذیر است و می توانید سریعتر جستجو کنید.

فرهنگ لغت در قالب xml تعریف شده است و می توانید با کلاس POSDedding ایجاد و ذخیره شوید. لطفاً اکنون Javadoc و منبع کد آن کلاس را بررسی کنید .

توجه: قالب باید مستند باشد و کد نمونه باید نحوه استفاده از فرهنگ لغت را نشان دهد.

Evaluation:

ارزیابی از پیش ساخته شده می تواند صحت pos tagger را اندازه گیری کند. صحت می تواند توسط دیتاست تستی ویا cross validation اندازه گیری شود.

Evaluation Tool:

یک ابزار خط فرمان برای ارزیابی یک مدل داده شده در مجموعه داده های آزمون وجود دارد. دستور زیر نحوه اجرای ابزار را نشان می دهد:

```
  $ opennlp POSTaggerEvaluator -model pt.postagger.bin -data pt.postagger.test
                                                                -encoding utf-8
```

متن زیر نتیجه نمره دقت را نشان می دهد ، به عنوان مثال:

```
Loading model ... done
Evaluating ... done
Accuracy: 0.9659110277825124
```

یک ابزار خط فرمان برای عبور از تأیید مجموعه داده های آزمون وجود دارد. دستور زیر نحوه اجرای ابزار را نشان می دهد:

```
$ opennlp POSTaggerCrossValidator -lang pt -data pt.postagger.test -encoding utf-8
```

متن زیر نمره دقت نتیجه نشان می دهد ، به عنوان مثال:

```
Accuracy: 0.9659110277825124
```

Lemmatizer :

lemmatizer برای یک فرم خاص در کلمه (توکن) و برچسب Part of Speech ، فرم فرهنگ لغت یک کلمه را که معمولاً

به عنوان lemma آن گفته می شود. یک token می تواند به صورت مبهم از چندین شکل اساسی یا کلمات فرهنگ لغت گرفته شود که به همین دلیل است

برای یافتن lemma، کلمه پست لازم است. به عنوان مثال ، شکل "نمایش" ممکن است به فعل "نشان دادن" یا به "اشاره" باشد

اسم "نمایش". در حال حاضر OpenNLP lemmatizer ی سازنده های آماری و مبتنی بر فرهنگ لغت را پیاده سازی می کند.

Lemmatizer Tool:

آسان ترین روش برای امتحان Lemmatizer استفاده از ابزار خط فرمان است.که دسترسی به داده های آماری Lemmatizer را فراهم می کند .این نکته را در نظر داشته باشید که این ابزار برای شرح دادن وتست در نظر گرفته شده است.

همانطور که در زیر میبینید زمانی که مدل Lemmatizer آموزش داده شد می توانید از این ابزار طبق دستور زیر استفاده کنید:

```
$ opennlp LemmatizerME en-lemmatizer.bin < sentences
```

در این دستورLemmatizer جملات pos tag شده را از ورودی استاندارد به صورت خط به خط می خواند برای مثال جمله زیر را در کنسول کپی کنید.

```
Rockwell_NNP International_NNP Corp._NNP 's_POS Tulsa_NNP unit_NN said_VBD it_PRP signed_VBD a_DT tentative_JJ agreement_NN extending_VBG its_PRP$ contract_NN with_IN Boeing_NNP Co._NNP to_TO provide_VB structural_JJ parts_NNS for_IN Boeing_NNP 's_POS 747_CD jetliners_NNS ._.
```

در این قسمت Lemmatizer توصیف هر جفت کلمه post tag در کنسول بازنشر شده است:

```
Rockwell NNP rockwell
International NNP international
Corp. NNP corp.
's POS 's
Tulsa NNP tulsa
unit NN unit
said VBD say
it PRP it
signed VBD sign
...
```

:Lemmatizer API

Lemmatizer توسط API در برنامه میتواند ترکیب شود.در حال حاظر داده های آماری و DictionaryLemmatizer در دسترس هستند.

```
LemmatizerModel model = null;
try (InputStream modelIn = new FileInputStream("en-lemmatizer.bin"))) {
model = new LemmatizerModel(modelIn);
}
```

زمانی که مدل بارگزاری شد می توان از LemmatizerME شی ساخت :

```
LemmatizerME lemmatizer = new LemmatizerME(model);
```

نمونه Lemmatizer اکنون آماده است تا داده ها را اصلاح کند. این انتظار را دارد که یک جمله tokenize شده به عنوان ورودی باشد ، که به عنوان یک آرایه String نشان داده می شود ، هر یک از اشیاء رشته در یک آرایه یک token است ، و برچسب های POS با هر یک از tokenها مرتبط است.

کد زیر نحوه تعیین محتمل ترین lemma برای یک جمله را نشان می دهد.

```
  String[] tokens = new String[] { "Rockwell", "International", "Corp.", "'s",
       "Tulsa", "unit", "said", "it", "signed", "a", "tentative", "agreement",
                "extending", "its", "contract", "with", "Boeing", "Co.", "to",
               "provide", "structural", "parts", "for", "Boeing", "'s", "747",
                                                          "jetliners", "." };
    String[] postags = new String[] { "NNP", "NNP", "NNP", "POS", "NNP", "NN",
             "VBD", "PRP", "VBD", "DT", "JJ", "NN", "VBG", "PRP$", "NN", "IN",
       "NNP", "NNP", "TO", "VB", "JJ", "NNS", "IN", "NNP", "POS", "CD", "NNS",
                                                                      "." };
                        String[] lemmas = lemmatizer.lemmatize(tokens, postags);
```

آرایه lemmas شامل یک lemma برای هر token در آرایه ورودی است. برچسب و lemma مربوطه را می توانید در همان شاخصی که توکن در آرایه ورودی دارد.

آرایه lemmas شامل یک lemma برای هر token در آرایه ورودی است. برچسب و lemma مربوطه را می توانید درهمان شاخصی که توکن در آرایه ورودی دارد یافت.

```
show NN show
showcase NN showcase
showcases NNS showcase
showdown NN showdown
showdowns NNS showdown
shower NN shower
showers NNS shower
showman NN showman
showmanship NN showmanship
showmen NNS showman
```

```
showroom NN showroom
showrooms NNS showroom
shows NNS show
shrapnel NN shrapnel
```

از طرف دیگر ، اگر یک جفت (کلمه ، postag) قادر به خروجی چندین lemma باشد ، فرهنگ لغت lemmatizer از یک فایل متنی تشکیل شده است. که برای هر سطر حاوی ، یک کلمه ، posttag آن و lemma های مربوط به آن با "#" جدا شده است:

```
muestras NN muestra
cantaba V cantar
fue V ir#ser
entramos V entrar
```

ابتدا فرهنگ لغت باید از دیسک یا منبع دیگری در حافظه بارگذاری شود. در نمونه زیر آن از دیسک بارگیری می شود.

```
InputStream dictLemmatizer = null;
try (dictLemmatizer = new FileInputStream("english-lemmatizer.txt")) {
}
```

پس از بارگیری فرهنگ لغت DictionaryLemmatizer را بارگزاری کرد می توان از ان شی ساخت.

```
   DictionaryLemmatizer lemmatizer = new DictionaryLemmatizer(dictLemmatizer);
```

نمونه DictionaryLemmatizer اکنون آماده است. این دو آرایه رشته را به عنوان ورودی انتظار دارد ، حاوی token ها و موارد دیگر حاوی postag های قابل مالاحظه است.

کد زیر نحوه یافتن یک lemma با استفاده از DictionaryLemmatizer را نشان می دهد

```
  String[] tokens = new String[]{"Most", "large", "cities", "in", "the", "US",
                                                                    "had",
                          "morning", "and", "afternoon", "newspapers", "."};
                                          String[] tags = tagger.tag(sent);
                      String[] lemmas = lemmatizer.lemmatize(tokens, postags);
```

آرایه برچسب ها حاوی یک برچسب قسمت گفتار برای هر token در آرایه ورودی است. برچسب ها و lemma های مربوطه را می توانید در اینجا پیدا کنید همان شاخصی که توکن در آرایه ورودی دارد.

:Lemmatizer Training

داده های آموزش شامل سه ستون است که توسط فضاها از هم جدا شده اند. هر کلمه در یک خط جداگانه قرار داده شده است و یک خالی وجود دارد

خط بعد از هر جمله ستون اول شامل کلمه فعلی ، دوم برچسب قسمت گفتار و سوم آن lemma آن است. اینجا نمونه ای از قالب فایل:

```
He PRP he
reckons VBZ reckon
the DT the
current JJ current
accounts NNS account
deficit NN deficit
will MD will
narrow VB narrow
to TO to
only RB only
# # #
1.8 CD 1.8
millions CD million
in IN in
September NNP september
. . O
```

Training API:

Lemmatizer یک API برای آموزش مدل جدید lemmatizer ارائه می دهد. ابتدا باید یک فایل پارامترهای آموزشی نیاز به شی سازی دارد:

```
TrainingParameters mlParams =
CmdLineUtil.loadTrainingParameters(params.getParams(), false);
if (mlParams == null) {
mlParams = ModelUtil.createDefaultTrainingParameters();
}
```

سپس داده های آموزش را می خوانیم:

```
InputStreamFactory inputStreamFactory = null;
try {
inputStreamFactory = new MarkableFileInputStreamFactory(
new File(en-lemmatizer.train));
} catch (FileNotFoundException e) {
e.printStackTrace();
}
ObjectStream<String> lineStream = null;
LemmaSampleStream lemmaStream = null;
try {
lineStream = new PlainTextByLineStream(
(inputStreamFactory), StandardCharsets.UTF 8);
lemmaStream = new LemmaSampleStream(lineStream);
} catch (IOException e) {
CmdLineUtil.handleCreateObjectStreamError(e);
}
```

مرحله بعدی که برای آموزش مدل پردازش می شود:

```
LemmatizerModel model;
try {
LemmatizerFactory lemmatizerFactory = LemmatizerFactory
.create(params.getFactory());
model = LemmatizerME.train(params.getLang(), lemmaStream, mlParams,
lemmatizerFactory);
} catch (IOException e) {
throw new TerminateToolException(-1,
"IO error while reading training data or indexing data: "
+ e.getMessage(),
e);
} finally {
try {sampleStream.close();
} catch (IOException e) {
}
}
```

Lemmatizer Evaluation:

ارزیابی داخلی ابزار می تواند دقت lemmatizer آماری را اندازه گیری کند. دقت را می توان در مجموعه داده های تست اندازه گیری کرد.

یک ابزار خط فرمان برای ارزیابی یک مدل داده شده در مجموعه داده های آزمون وجود دارد. دستور زیر نحوه اجرای ابزار را نشان می دهد:

```
$ opennlp LemmatizerEvaluator -model en-lemmatizer.bin -data en-
lemmatizer.test -encoding utf-8
```

نمره دقت نتیجه در زیر نشان داده شده است، به عنوان مثال:

```
Loading model ... done
Evaluating ... done
Accuracy: 0.9659110277825124
```

Chunker:

Chunking:

به منظور تقسیم یک متن به چند کلمه که از لحاظ قواعدی به یک دیگر مرتبط هستند,مثل گروه اسمی و فعلی اما بدون اینکه ساختار درونی ویا حتی نقش آن ها در جمله مشخص شود.

Chunker Tool:

آسان ترین روش برای تست chunker استفاده از ابزار خط فرمان است. این ابزار تنها برای تشریح و تست یه کار می رود

مدل english maxent chunker را از وبسایت بارگیری کرده وبا دستور زیر شروع به استفاده از ابزار chunker کنید:

```
$ opennlp ChunkerME en-chunker.bin
```

Chunker در حال حاظر جملات pos tag شده را خط به خط از ورودی استاندارد می خواند. این جملات را داخل کنسول کپی کنید:

```
    Rockwell_NNP International_NNP Corp._NNP 's_POS Tulsa_NNP unit_NN said_VBD
                                                                it_PRP signed_VBD
```

```
a_DT tentative_JJ agreement_NN extending_VBG its_PRP$ contract_NN with_IN
Boeing_NNP Co._NNP
to_TO provide_VB structural_JJ parts_NNS for_IN Boeing_NNP 's_POS 747_CD
jetliners_NNS ._.
Rockwell_NNP said_VBD the_DT agreement_NN calls_VBZ for_IN it_PRP to_TO
supply_VB 200_CD
additional_JJ so-called_JJ shipsets_NNS for_IN the_DT planes_NNS ._.
```

chunker در جملات token های گروه شده را در کنسول به صورت زیر بازنشر می دهد:

```
Rockwell_NNP International_NNP Corp._NNP 's_POS Tulsa_NNP unit_NN said_VBD
it_PRP signed_VBD
a_DT tentative_JJ agreement_NN extending_VBG its_PRP$ contract_NN with_IN
Boeing_NNP Co._NNP
to_TO provide_VB structural_JJ parts_NNS for_IN Boeing_NNP 's_POS 747_CD
jetliners_NNS ._.
Rockwell_NNP said_VBD the_DT agreement_NN calls_VBZ for_IN it_PRP to_TO
supply_VB 200_CD
additional_JJ so-called_JJ shipsets_NNS for_IN the_DT planes_NNS ._.
```

Chunking API:

Chunker می تواند به وسیله ی API اش در برنامه ها مورد استفاده قرار گیرد. در گام اول مدل chunker باید در مموری دیسک یا هر منبع دیگر بارگزاری شود.در مثال زیر از دیسک بارگیری شده است:

```
InputStream modelIn = null;
ChunkerModel model = null;
try (modelIn = new FileInputStream("en-chunker.bin")){
model = new ChunkerModel(modelIn);
}
```

بعد از اینکه مدل بارگیری شد میتوانیم از chunker شی بسازیم:

```
ChunkerME chunker = new ChunkerME(model);
```

در حال حاظر شی ساخته شده میتواند داده ها را برچسب گزاری کند.این شی جملات tokenize شده را به عوان ورودی می گیرد که تحت عنوان ارایه ای از رشته ارایه می شود .هر شی رشته ای در داخل ارایه یک token است و pos tag ها به هر token اختصاص داده می شود

در کد زیر چگونگی مشخص کردن chunk tag را نشان می دهد:

```
String sent[] = new String[] { "Rockwell", "International", "Corp.", "'s",
"Tulsa", "unit", "said", "it", "signed", "a", "tentative", "agreement",
"extending", "its", "contract", "with", "Boeing", "Co.", "to",
"provide", "structural", "parts", "for", "Boeing", "'s", "747",
```

```
                                                                "jetliners", "." };
        String pos[] = new String[] { "NNP", "NNP", "NNP", "POS", "NNP", "NN",
             "VBD", "PRP", "VBD", "DT", "JJ", "NN", "VBG", "PRP$", "NN", "IN",
      "NNP", "NNP", "TO", "VB", "JJ", "NNS", "IN", "NNP", "POS", "CD", "NNS",
                                                                        "." };
                                        String tag[] = chunker.chunk(sent, pos);
```

آرایه برچسب ها شامل یک chunk tag برای هر token در آرایه ورودی می شود.برچسب خروجی را می توان در همان ایندکسی که به عنوان token در آرایه ورودی داریم یافت. درجه اطمینان را می توان را از خروجی برچسب هایی که به راحتی از خروجی فراخوانی متد باز میگردد یافت.

```
                                                double probs[] = chunker.probs();
```

```
                        Sequence topSequences[] = chunk.topKSequences(sent, pos);
```

Chunker Training:

مدل های از قبل آموزش دیده ممکن است برای زبان مورد نظر در دسترس نباشد ، نمی تواند مولفه مهم را تشخیص دهد یا عملکرد آن در خارج از حوزه گزارش شده به اندازه کافی خوب نیست. اینها دلیل اصلی برای انجام آموزش های سفارشی چانکر در یک جسد یا جسد است که توسط بخش خصوصی تمدید می شود.داده های آموزش گرفته شده از داده هایی که باید تجزیه و تحلیل شوند.

داده های آموزش را می توان به قالب آموزش Chunker OpenNLP تبدیل کرد ، که بر اساس CoNLL2000 است. قالبهای دیگر نیز ممکن است در دسترس باشد. داده های آموش شامل سه ستون است که یک فضای واحد را از هم جدا کرده اند. هر کلمه در یک خط جداگانه قرار داده شده است و بعد از هر جمله یک خط خالی وجود دارد. ستون اول شامل کلمه فعلی است ، دومین برچسب قسمت گفتار آن وسوم برچسب تکه برچسب های تکه دارای نام نوع تکه است ، به عنوان مثال I-NP برای کلمات اصطلاح اسم و I-VP برای فعل کلمات عبارتی بیشتر انواع تکه دارای دو نوع برچسب تکه است ، B-CHUNK برای کلمه اول تکه و I-CHUNK برای هر کلمه دیگر در تکه در اینجا نمونه ای از فرمت پرونده آورده شده است:

جمله نمونه داده های آموزش:

```
He PRP B-NP
reckons VBZ B-VP
the DT B-NP
current JJ I-NP
account NN I-NP
deficit NN I-NP
will MD B-VP
narrow VB I-VP
to TO B-PP
only RB B-NP
# # I-NP
1.8 CD I-NP
billion CD I-NP
in IN B-PP
September NNP B-NP
. . O
```

Training API:

Chunker یک API را برای آموزش مدل جدید chunker ارایه می دهد.کد زیر نحوه استفاده از آن را نشان می دهد:

```
ObjectStream<String> lineStream =
new PlainTextByLineStream(new FileInputStream("en-chunker.train"),
StandardCharsets.UTF_8);
ChunkerModel model;
try(ObjectStream<ChunkSample> sampleStream = new
ChunkSampleStream(lineStream)) {
model = ChunkerME.train("en", sampleStream,
new DefaultChunkerContextGenerator(), TrainingParameters.defaultParams());
}
try (OutputStream modelOut = new BufferedOutputStream(new
FileOutputStream(modelFile))) {
model.serialize(modelOut);
}
```

Chunker Evaluation:

Evaluation از پیش ساخته شده می تواند عملکرد chunker را ارزیابی کند.عملکرد به وسیله تست دیتاست ویا cross validation قابیل اندازه گیری است.

Chunker Evaluation Tool:

دستور زیر نحوه اجراي اين ابزار را نشان مي دهد:

```
$ opennlp ChunkerEvaluator
Usage: opennlp ChunkerEvaluator[.ad] -model model [-misclassified true|false]
\
[-detailedF true|false] -lang language -data sampleData [-encoding
charsetName]
```

در زیر یک نمونه از دستورات با توجه به اینکه شما یک نمونه داده با نام en-chunker.eval دارید و مدلی به نام enchunker.bin را آموزش داده اید آورده شده است:

```
      $ opennlp ChunkerEvaluator -model en-chunker.bin -data en-chunker.eval -
                                                                  encoding UTF-8
```

و این خروجی مثال است:

```
Precision: 0.9255923572240226
Recall: 0.9220610430991112
F-Measure: 0.9238233255623465
```

همچنین می توانید از این ابزار برای انجام اعتبار سنجی متقاطع 10 برابری Chunker استفاده کنید. دستور زیر نشان می دهد که چگونه می توان ابزار را اجرا کرد:

```
$ opennlp ChunkerCrossValidator
Usage: opennlp ChunkerCrossValidator[.ad] [-params paramsFile] [-iterations
num] [-cutoff num] \
[-misclassified true|false] [-folds num] [-detailedF true|false] \
-lang language -data sampleData [-encoding charsetName]
Arguments description:
-params paramsFile
```

```
training parameters file.
-iterations num
number of training iterations, ignored if -params is used.
-cutoff num
minimal number of times a feature must be seen, ignored if -params is used.
-misclassified true|false
if true will print false negatives and false positives.
-folds num
number of folds, default is 10.
-detailedF true|false
if true will print detailed FMeasure results.
-lang language
language which is being processed.
-data sampleData
data to be used, usually a file name.
-encoding charsetName
encoding for reading and writing text, if absent the system default is used.
```

لازم نیست که مدل را پاس دهیم .این ابزار به صورت اتوماتیک داده را به داده آموزش و ارزیابی تقسیم می کند:

```
$ opennlp ChunkerCrossValidator -lang pt -data en-chunker.cross -encoding
UTF-8
```

:Parser

:Parser Tool

آسان ترین روش برای تست parser ستفاده از ابزار خط فرمان است. این ابزارتنها برای تست و تشریح مورد استفاده قرار می گیرد. مدل English chunking parser را از سایت بارگیری کنید و با دستور زیر شروع به استفاده از ابزار parser با دستور زیر کنید:

```
$ opennlp Parser en-parser-chunking.bin
```

مراقب باشید! بارگزاری مدل بزرگ parser زمان بر است.نمونه زیر را داخل کنسول کپي کنید:

```
The quick brown fox jumps over the lazy dog .
```

Parser باید متن زیر را در کنسول چاپ کند:

```
 (TOP (NP (NP (DT The) (JJ quick) (JJ brown) (NN fox) (NNS jumps)) (PP (IN
over) (NP (DT the)
(JJ lazy) (NN dog))) (. .)))
```

با دستور زیر متن مي تواند از فايل ورودي خوانده و نتيجه در فايل خروجي نوشته شود:

```
$ opennlp Parser en-parser-chunking.bin < article-tokenized.txt > article-
parsed.txt.
```

فايل article-tokenized.txt بايد شامل يك جمله در هر خط كه توسط مدل tokenizer انگليسي موجود در سايت tokenize شده, باشد. براي كسب اطلاعات از tokenizer به بخش ان مراجعه فرماييد.

:Parsing API

Parser به سادگی به وسیله API اش قابل ترکیب شدن در برنامه ها است برای شی ساختن از parser ابتدا مدل parser باید بارگزاری شود:

```
InputStream modelIn = new FileInputStream("en-parser-chunking.bin");
try {
ParserModel model = new ParserModel(modelIn);
}
catch (IOException e) {
```

```
e.printStackTrace();
}
finally {
if (modelIn != null) {
try {
modelIn.close();
}
catch (IOException e) {
}
}
}
```

برخلاف دیگر کامپوننت های دیگر به جای استفاده از عملگر new برای شی سازی از parser از متد factory استفاده می شود.مدل parser به وسیله chunking parser ویا tree insert parser از پیش آموزش داده است , باید پیاده سازی parser به درستی انتخاب شود.متد factory پارامتر type (نوع) را از ورودی میخواند و باتوجه به پیاده سازی parser شی می سازد.

```
Parser parser = ParserFactory.create(model);
```

در حال حاظر tree insert parser آزمایشی است و مدل از پیش آموزش داده شده ای برای ان وجود ندارد.Parser فضای سفید جمله tokenized شده را در نظر می گیرد.متد برنامه برنامه موجود در ابزار خط فرمان می تواند رشته جمله را parse کند.

کد زیر نحوه فراخوانی parser را نشان می دهد:

```
String sentence = "The quick brown fox jumps over the lazy dog .";
Parse topParses[] = ParserTool.parseLine(sentence, parser, 1);
```

آرایه topParses به دلیل اینکه عدد parse به 1 مقدار دهی شده است تنها حاوی یک parse است. شی parse حاوی parse tree است.برای نمایش parse tree متد show را فراخوانی می کند و خروجی را در کنسول ویا StringBuffer ساخته شده چاپ می کند.شبیه به Exception.printStackTrace .

:Parser Training

OpenNLP دو پیاده سازی مختلف parser را اریه می دهد , chunking parser و treeinsert parser. یکی از این دو آزمایشی است و استفاده از آن در محیط واقعی توصیه نمی شود.آموزش می تواند هم به وسیله خط فرمان و هم به وسیله ی API آموزش, انجام شود. درگام اول داده آموزش باید در چارچوب OpenNLP باشد که این چارچوب Penn Treebank نام دارد و محدودیت جمله در هر خط را دارد.

```
 (TOP (S (NP-SBJ (DT Some) )(VP (VBP say) (NP (NNP November) ))(. .) ))
(TOP (S (NP-SBJ (PRP I) )(VP (VBP say) (NP (CD 1992) ))(. .) ('' '') ))
```

برای کسب اطلاعات بیشتر درباره نشانه گزاری Penn Treebank می توانید به لینک زیر
https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

مراجعه کنید.مدل parser درداخل خود مدلpos tagger را هم شامل می شود, بنابر میزان داده های آموزشی توصیه می شود مدل tagger خود را به مدل از پیش آموزش داده شده برای حجم داده های بزرگ تغییر دهیم. مدل از پیش آموزش داده شده parser که در وبسایت موجود است این مهم را انجام داده که باعث کارایی بهتر می شود.

:Training API

Parser training API آموزش دادن مدل جدید parser را پشتیبانی می کند.برای آموزش آن چهار گام زیر ضروری است:

- نیاز به ساخت شی از کلاس HeadRules دارید: درحال حاظر EnglishHeadRules و AncoraSpanishHeadRules در دسترس هستند.
- برنامه باید یک نمونه data stream باز کند.
- فراخوانی متد parse train که می تواند parser از نوع CHUNKING ویا TREEINSERT باشد.
- ذخیره سازی ParseModel در یک فایل

ریز کد زیر نحوه شی سازی از HeadRules را نشان می دهد:

```
static HeadRules createHeadRules(TrainerToolParams params) throws IOException
{
ArtifactSerializer headRulesSerializer = null;
if (params.getHeadRulesSerializerImpl() != null) {
headRulesSerializer =
ExtensionLoader.instantiateExtension(ArtifactSerializer.class,
params.getHeadRulesSerializerImpl());
}
else {
if ("en".equals(params.getLang())) {
headRulesSerializer = new
opennlp.tools.parser.lang.en.HeadRules.HeadRulesSerializer();
}
else if ("es".equals(params.getLang())) {
headRulesSerializer = new
opennlp.tools.parser.lang.es.AncoraSpanishHeadRules.HeadRulesSerializer();
}
else {
// default for now, this case should probably cause an error ...
headRulesSerializer = new
opennlp.tools.parser.lang.en.HeadRules.HeadRulesSerializer();
}
}
Object headRulesObject = headRulesSerializer.create(new
FileInputStream(params.getHeadRules()));
if (headRulesObject instanceof HeadRules) {
return (HeadRules) headRulesObject;
}
else {
throw new TerminateToolException(-1, "HeadRules Artifact Serializer must
create an object of type HeadRules!");
}
}
```

کد زیر سه گام دیگر, نام گزاری و باز کردن داده وآموزش مدل وذخیره کردن parseModel در داخل فایل خروجی است.

```
ParserModel model = null;
File modelOutFile = params.getModel();
CmdLineUtil.checkOutputFile("parser model", modelOutFile);
try {
HeadRules rules = createHeadRules(params);
InputStreamFactory inputStreamFactory = new
MarkableFileInputStreamFactory(new File("parsing.train"));
ObjectStream<String> stringStream = new
PlainTextByLineStream(inputStreamFactory, StandardCharsets.UTF_8);
ObjectStream<Parse> sampleStream = new ParseSample(stringStream);
```

```
ParserType type = parseParserType(params.getParserType());
if (ParserType.CHUNKING.equals(type)) {
model = opennlp.tools.parser.chunking.Parser.train(
params.getLang(), sampleStream, rules,mlParams);
} else if (ParserType.TREEINSERT.equals(type)) {
model = opennlp.tools.parser.treeinsert.Parser.train(params.getLang(),
sampleStream, rules,
mlParams);
}
}
catch (IOException e) {
throw new TerminateToolException(-1, "IO error while reading training data or
indexing data: "
+ e.getMessage(), e);
}
finally {
try {
sampleStream.close();
}
catch (IOException e) {
// sorry that this can fail
}
}
CmdLineUtil.writeModel("parser", modelOutFile, model);
```

:Parser Evaluation

ابزار درونی evaluation می تواند عملکرد parser را ارزیابی کند. عمکرد روی dataset تست انجام می شود.

:Parser Evaluation Tool

دستور زیر نحوه اجرای این دستور را نشان می دهد:

```
$ opennlp ParserEvaluator
Usage: opennlp ParserEvaluator[.ontonotes|frenchtreebank] [-misclassified
true|false] -model model \
-data sampleData [-encoding charsetName]
```

در نمونه دستور زیر نمونه dataset شما که en-parser-chunking.eval نامیده شده در نظر گرفته شده و شما یک مدل به نام en-parser-chunking.bin آموزش داده اید.

```
$ opennlp ParserEvaluator -model en-parser-chunking.bin -data en-parser-
chunking.eval -encoding UTF-8
```

و این نمونه خروجی است:

```
Precision: 0.9009744742967609
Recall: 0.8962012400910446
F-Measure: 0.8985815184245214
```

:Evaluation API

Evaluation می تواند روی مدل از پیش آموزش داده شده و یک dataset تستی و یا به وسیله cross validation اجرا شود. در گام اول باید مدل بارگزاری شود ویک parse ObjectStream باید ساخته شود.

دو شی زیر که در کد زیر وجود دارد را درنظر بگیرید که چگونگی پیاده سازی evaluation را نشان می دهد:

```
Parser parser = ParserFactory.create(model);
ParserEvaluator evaluator = new ParserEvaluator(parser);
evaluator.evaluate(sampleStream);
FMeasure result = evaluator.getFMeasure();
System.out.println(result.toString());
```

در موارد cross validation باید تمام آرگومان ها از پیش آماده باشد.برای اجرای cross validation, ObjectStream قابل مقدار دهی مجدد باشد

```
InputStreamFactory inputStreamFactory = new
MarkableFileInputStreamFactory(new File("parsing.train"));
ObjectStream<String> stringStream = new
PlainTextByLineStream(inputStreamFactory, StandardCharsets.UTF_8);
ObjectStream<Parse> sampleStream = new ParseSample(stringStream);
ParserCrossValidator evaluator = new ParserCrossValidator("en",
trainParameters, headRules, \
parserType, listeners.toArray(new
ParserEvaluationMonitor[listeners.size()]));
evaluator.evaluate(sampleStream, 10);
FMeasure result = evaluator.getFMeasure();
System.out.println(result.toString());
```

Coreference Resolution:

سیستم OpenNLP Coreference Resolution چندین مولفه های ذکر شده در سند را به هم وصل می کند.این پیاده سازی در حال حاضر پیاده سازی opennlp محدود به عبارت های اسمی می شود.دیگر مولفه ها قابل دستیابی نیست.

نتیجه‌گیری (Conclusion):

OpenNLP یک ابزار متن باز برای هدف پردازش متن در زبان طبیعی است در این مقاله سعی شد ویژگی های اساسی این ابزار مورد بررسی قرار گیرد از این ابزار و ویژگی های ذکر شده در مقاله می توان در توسعه موتور های جست وجو گر و سیستم های جست وجوی متن در data warehouse ها و.... استفاده نمود

تقدیر و تشکر (Acknowledgment):

درنهایت از خداوند متعال و استاد محبوبم مهندس بهادر که از راهنمایی های ایشان در تدوین این مقاله بهره بردم تشکر میکنم

مراجع(References) :

https://en.wikipedia.org/wiki/Language_model

https://en.wikipedia.org/wiki/Natural_language_processing

https://opennlp.apache.org/docs/1.9.2/manual/opennlp.html