Style Definition: TOC 1: Tab stops: 3.32 cm, Right + 3.96 cm, Right + 17.51 cm, Left,Leader: ...

Style Definition: TOC 2: Tab stops: 5.37 cm, Right + 17.51 cm, Left,Leader: ...


# گزارش پروژه پایانی – فاز نهایی

## استخراج رابطه در زبان فارسی با بررسی وابستگی‌های جهانی Seraji و PerDT

مبانی پردازش زبان و گفتار

استاد: دکتر بهروز مینایی بیدگلی

اعضای گروه:

دانیال بازمانده ۹۷۵۲۱۱۳۵

محمدحسین کریمیان ۹۷۵۲۱۴۶۸

## فهرست مطالب

## ۱ هدف پروژه

هدف اصلی و غایی این پروژه، پیاده‌سازی یک سیستم هوشمند با استفاده از ابزارهای پردازش زبان‌های طبیعی و یادگیری ماشین، در جهت استخراج رابطه از جملات زبان فارسی است. موضوع پروژه انتخابی در واقع، ترکیبی از دو پروژه از لیست پروژه‌های داده شده است. چرا که در این پروژه، استخراج رابطه با استفاده از هر دو وابستگی جهانی Seraji و PerDT انجام شده است.

## ۲ مفهوم استخراج رابطه

در ابتدا، به توضیح و تفصیل مفهوم استخراج رابطه (Relation Extraction) در پردازش زبان‌های طبیعی می‌پردازیم.

عملیات استخراج رابطه در آموزش مدل‌های هوش مصنوعی مبتنی بر پردازش زبان طبیعی برای یادگیری روابط موجودیت‌های مختلف در یک متن برای تجزیه و تحلیل داده‌های استخراج‌شده انجام می‌شود. تکنیک‌های مختلفی برای انجام استخراج موجودیت وجود دارد، از تطبیق رشته ساده تا رویکردهای خودکار پیچیده‌تر.

منظور از استخراج رابطه مجموعه تسک‌هایی است که به استخراج روابط معنایی موجود در متون می‌پردازد. این روابط معمولا بین دو یا چند نهاد از یک نوع خاص (مانند شخص، سازمان، مکان و...) رخ می‌دهند و در تعدادی از دسته‌بندی‌های معنایی قرار می‌گیرند. (مانند متاهل، مجرد، استخدام‌شده و...) به عبارت بهتر، استخراج رابطه وظیفه‌ی پیش‌بینی صفات و روابط را برای موجودات و انتیتی‌های درون یک جمله دارد و مولفه‌ای کلیدی برای ساخت گراف‌های دانش است.

به عنوان نمونه، در زبان انگلیسی جمله‌ی «Barack Obama was born in Honolulu, Hawaii» را در نظر بگیرید. در این جمله هدف پیدا کردن رابطه‌ی «bornInCity» است. این رابطه بین دو موجودیت (Entity) «Barak Obama» و «Honolulu» وجود دارد.

البته در این پروژه ما با جملات و دادگان فارسی کار خواهیم داشت و به پیاده‌سازی استخراج رابطه در زبان فارسی پرداخته‌ایم.

به عنوان نمونه در زبان فارسی، جمله‌ی «دانیال بازمانده در شهر مشهد به دنیا آمد.» را در نظر بگیرید. هدف از تسک استخراج رابطه پیدا کردن رابطه‌ی «به دنیا آمدن» است. ما انتظار داریم پس از انجام موفق عملیات، آرگومان‌های این رابطه به دست آیند. در واقع، «دانیال بازمانده» و «شهر مشهد» آرگومان‌ها و «به دنیا آمد.» رابطه‌ای است که بین این دو آرگومان وجود دارد.

به عنوان نمونه‌ی دیگر می‌توان جمله‌ی «پاریس در فرانسه است.» را بیان کرد. در این جمله آرگومان‌ها «پاریس» و «فرانسه» هستند و «در» رابطه‌ای است که بین این دو موجودیت وجود دارد. همانطور که واضح است، الزامی به فعل بودن و معنی فعل دادن برای رابطه وجود ندارد.

انواع روش‌های مختلفی برای انجام عملیات استخراج رابطه وجود دارد:

۱) Rule-based RE
۲) Weakly Supervised RE
۳) Supervised RE
۴) Distantly Supervised RE
۵) Unsupervised RE

هر کدام از روش‌های فوق، مزایا و معایبی دارند که تفصیل آنها از حوصله‌ی این گزارش خارج است.

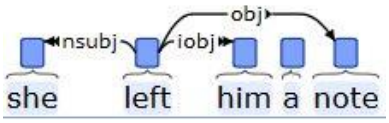
از کاربردهای مهم استخراج رابطه می‌توان به جستجوی ساختار یافته، تجزیه و تحلیل احساسات ( Sentiment Analysis)، سیستم‌های پرسش و پاسخ (Question Answering) و خلاصه‌سازی متن می‌توان نام برد.

## ۳ وابستگی‌های جهانی

وابستگی جهانی (UD) پروژه‌ای است که در حال توسعه‌ی حاشیه‌نویسی از نظر زبانی سازگار برای بسیاری از زبان‌ها، با هدف تسهیل توسعه‌ی تجزیه‌کننده چندزبانه، یادگیری بین زبانی و تجزیه‌وتحلیل تحقیقات از منظر گونه‌شناسی زبان است. فلسفه‌ی کلی این است که فهرستی جهانی از مقوله‌ها و دستورالعمل‌ها برای تسهیل حاشیه‌نویسی ثابت ساختارهای مشابه در بین زبان‌ها فراهم کند.

به عبارت بهتر، وابستگی‌های جهانی (UD) چارچوبی برای حاشیه‌نویسی ثابت دستور زبان (بخش‌هایی از گفتار، ویژگی‌های صرفی و وابستگی‌های نحوی) در زبان‌های مختلف انسانی است. UD یک تلاش با بیش از ۳۰۰ مشارکت‌کننده است که نزدیک به ۲۰۰ درخت را از بیش از ۱۰۰ زبان تولید می‌کنند.

این طرح، تجزیه و تحلیل نحوی جملات را از نظر وابستگی‌های دستور زبان مربوطه تولید می کند. هر وابستگی از یک تابع نحوی مشخص می‌شود که روی لبه‌ی وابستگی (فلش در شکل) نشان داده می‌شود.




برای موفقیت یک UD خوب، موارد زیر باید رعایت شوند:

- در زمینه‌ی تحلیل زبانی برای هر زبان رضایت‌بخش و مناسب باشد.
- برای گونه‌شناسی زبانی خوب باشد. به‌عنوان مثال، زمینه‌ای مناسب برای ایجاد موازی زبانی بین زبان‌ها و خانواده‌های زبانی فراهم کند.
- برای حاشیه‌نویسی سریع و ثابت توسط یک حاشیه‌نویس انسانی مناسب باشد.
- به راحتی توسط یک غیر زبان شناس قابل درک و استفاده باشد.
- برای تجزیه کامپیوتری با دقت بالا مناسب باشد.
- به خوبی از وظایف درک زبان پایین دستی پشتیبانی کند. (استخراج رابطه، درک مطلب، ترجمه ماشینی و...)

در زبان فارسی، وابستگی جهانی حاوی داده‌هایی از چندین بانک درختی (Treebank) است که توسط تیم‌های مختلف در زمان‌های مختلف و با ابزارهای تبدیل متفاوت ایجاد شده‌اند. در حال حاضر دو وابستگی جهانی مهم در زبان فارسی وجود دارند: Seraji و PerDT.

## ۳٫۱ وابستگی جهانی *Seraji*

بانک درختی وابستگی جهانی فارسی (Persian UD) نسخه تبدیل‌شده‌ی بانک درختی وابستگی فارسی اوپسالا (UPDT) است. (سراجی، ۲۰۱۵). این بانک درختی، طرح حاشیه‌نویسی اصلی خود را بر اساس وابستگی‌های تایپ‌شده‌ی استنفورد دارد. (دی مارنف و همکاران، ۲۰۰۶؛ د مارنف و منینگ، ۲۰۰۸)

این طرح برای زبان فارسی گسترش یافت تا روابط نحوی خاص زبان را در بر گیرد که نمی‌توانست توسط طرح اولیه توسعه‌یافته برای انگلیسی پوشش داده شود. این بانک درختی شامل ۶۰۰۰ جمله‌ی متون مکتوب با تنوع دامنه بزرگ، از نظر ژانرهای مختلف (شامل مقالات روزنامه، داستان ها، توضیحات فنی و اسناد در مورد فرهنگ و هنر) و نشانه‌گذاری است. تغییرات در نشانه‌گذاری به دلیل تغییرات املایی کلمات مرکب و عبارات ثابت در زبان است.

*UPDT* اصلی توسط مژگان سراجی و زیر نظر جواکیم نیور و کارینا جهانی در دانشگاه اوپسالا ساخته شد. جدای از طرح حاشیه‌نویسی جهانی و قوانین کلی در *UD*، *UD* فارسی و *UPDT* در نشانه گذاری تفاوت بیشتری دارند. تمام کلمات حاوی کلیدهای تقسیم‌نشده (قطعات ضمایر و کوپولا) که با برچسب‌های پیچیده در *UPDT* حاشیه نویسی شده اند، از کلیدواژه ها جدا شده‌اند و برچسب‌های متمایز در *UD* فارسی دریافت کرده‌اند.

تبدیل *UPDT* به وابستگی های جهانی به صورت نیمه خودکار انجام شده است. در این فرآیند، از یک اسکریپت تبدیل برای معکوس کردن سر و روابط وابسته در تعدیل‌کننده حرف اضافه (prep) و مفعول یک حرف اضافه (pobj) استفاده کردیم. علاوه بر این، ما از اسکریپت‌های دیگری که برای فارسی طراحی شده‌اند، برای جدا کردن انواع کلیتیک‌ها از میزبان خود استفاده کرده‌ایم. متعاقباً قوانین مختلفی را برای بازنویسی تگ‌های درشت قسمت گفتار و برچسب‌های وابستگی اضافه کردیم.

## ۳٫۲ وابستگی جهانی *PerDT*

بانک درختی وابستگی جهانی فارسی (PerUDT) حاصل پوشش خودکار بانک درخت وابستگی فارسی (PerDT) با اصلاحات دستی گسترده است که توسط محمدصادق رسولی، پگاه صفری، امیرسعید مولودی و علیرضا نوریان در سال ۲۰۱۳ پیاده‌سازی شده است.

بانک درختی اصلی شامل ۲۹ هزار جمله است که از متون فارسی معاصر در ژانرهای مختلف از جمله: اخبار، مقالات دانشگاهی، مقالات مجلات و داستان‌ها نمونه برداری شده است. این بانک درختی بر اساس یک طرح زبان خاص حاشیه‌نویسی شد و تبدیل خودکار آن شامل سه مرحله اصلی بود:

۱) بازبینی توکن‌سازی (Revising Tokenization)
۲) نگاشت POS (POS Mapping)
۳) نگاشت وابستگی (Dependency Mapping)

در مرحله‌ی نشانه‌گذاری، به منظور جداسازی عطف های چند کلمه‌ای افعال ساده که به صورت یک نشانه در PerDT گروه بندی شده اند، از دستورالعمل های موجود برای یافتن خودکار افعال اصلی پیروی کردیم. هم‌چنین به طور خودکار کلیدهای ضمیری را جدا کردیم.

در مرحله تبدیل *POS*، ما از آخرین تگر فارسی NER مبتنی بر BERT (Taher et al., 2020) با اصلاحات دستی برای گسترش فراخوان استفاده کردیم. از طریق هفت موجودیت مختلف شناسایی‌شده توسط برچسب‌گذار، از Person و Location برای علامت‌گذاری برچسب‌های PROPN استفاده کردیم. در مجموع، PerDT شامل ۴۳ رابطه نموی بدون نگاشت مستقیم است. حروف ربط از ابتدای جمله تا انتها مرتب شده اند و مهمتر از آن، حروف اضافه به عنوان سر عبارات اضافه و افعال کمکی به عنوان سر جملات قرار می‌گیرند.

بنابراین ما ترتیب حروف ربط را از انتها به ابتدا از طریق یک اسکریپت و قواعد مناسب برای تبدیل هر نوع رابطه به نسخه UD آن به درستی مرتب کردیم. در طول کل فرآیند و در پایان هر مرحله، نتایج را بررسی کردیم و در صورت نیاز اصلاحات دستی را اعمال کردیم.

## ۳٫۳ مقایسه‌ی وابستگی‌های جهانی

در تصویر زیر، جدولی از مقایسهٔ «درخت بانک وابستگی فارسی اوپسالا» با هریک از وابستگی‌های جهانی Seraji و PerDT ارائه داده است. مبنا را بر وابستگی‌های جهانی می‌گذاریم و نشان می‌دهد هر برچسب اجزای سخن در وابستگی جهانی چه معادلی در پروژه‌ی دانشگاه اوپسالا (Seraji) و «پروژه‌ی دادگان وابستگی زبان فارسی (PerDT)» دارد.

| Dad Sub-Tag Fa | Dad Sub-Tag | Dad Fa Name | Dad Tag | Seraji Fa Name | Seraji Name | Seraji Tag | Persian Name | UD Name | UD Tag | UD Num |
|---|---|---|---|---|---|---|---|---|---|---|
| صفت مطلق | AJP | صفت | ADJ | صفت | Adjective | ADJ | صفت | adjective | ADJ | 1 |
| صفت عالی | AJSUP | | | صفت عالی | Superlative adjective | ADJ_SUP | | | | |
| صفت تفضیلی | AJCM | | | | | | | | | |
| صفت شمارشی پسین | POSNUM | | | | | | | | | |
| | | | | صفت مفعولی | Participle adjective | ADJ_IND | | | | |
| | | | | صفت ندا | Vocative adjective | ADJ_SUP | | | | |
| | | | | صفت مقایسه‌ای | Comparative adjective | ADJ_CMPR | | | | |
| | ندارد | حرف اضافه پیشین | PREP | حرف اضافه پیشین | Preposition | P | حرف اضافه | adposition | ADP | 2 |
| قید مختص | SADV | قید | ADV | قید | Adverb | ADV | قید | adverb | ADV | 3 |
| | | | | قید مقایسه | Adverb of Comparison | ADV_COMP | | | | |
| | | | | قید استفهام | Adverb of Interrogation | ADV_I | | | | |
| | | | | قید مکان | Adverb of Location | ADV_LOC | | | | |
| | | | | قید نفی | Adverb of Negation | ADV_NEG | | | | |
| | | | | قید زمان | Adverb of time | ADV_TIME | | | | |
| | | شبه جمله | PSUS | | | | | | | |
| | | جزء دستوری | PART | | | | | | | |
| وجهی | MODL | فعل | V | فعل کمکی | Auxiliary Verb | V_AUX | فعل کمکی | auxiliary | AUX | 4 |
| | | نقش نمای همپایگی | CONJ | حرف ربط | Conjunction | CON | حرف ربط هم پایه ساز | coordinating conjunction | CCONJ | 5 |
| صفت تعجبی | EXAJ | پیش توصیف گر | PREM | مشخصگر | Determiner | DET | مشخصگر | determiner | DET | 6 |
| صفت پرسشی | QUAJ | | | | | | | | | |
| صفت اشاره | DEMAJ | | | | | | | | | |
| صفت مبهم | AMBAJ | | | | | | | | | |
| نقش نمای ندا پیشین | PRADR | نقش نمای ندا | ADR | صوت و حرف ندا | Interjection | INT | صوت و حرف ندا | interjection | INTJ | 7 |
| نقش نمای ندا پسین | POSADR | | | | | | | | | |
| جاندار بی جان | ANM IANM | شبه جمله | PSUS | | | | اسم | noun | NOUN | 8 |
| | | اسم | N | اسم جمع | Plural noun | N_PL | | | | |
| | | | | اسم مفرد | Singular noun | N_SING | | | | |
| | | | | اسم ندا | Vocative noun | N_VOC | | | | |
| | | شاخص | IDEN | | | | | | | |
| | | صفت شمارشی پیشین | PRENUM | عدد | Numeral | NUM | عدد | numeral | NUM | 9 |
| | | صفت شمارشی پسین | POSNUM | | | | | | | |
| | | حرف اضافه پسین | POSTP | علامت مفعولی | Accusative marker | CLITIC | ادات | particle | PART | 10 |
| ضمیر شخصی جدا | SEPER | ضمیر | PR | ضمیر | Pronoun | PRO | ضمیر | pronoun | PRON | 11 |
| ضمیر شخصی پیوسته | JOPER | | | | | | | | | |
| ضمیر اشاره | DEMON | | | | | | | | | |
| ضمیر پرسشی | INTG | | | | | | | | | |
| ضمیر بازتابی مشترک | CREFX | | | | | | | | | |
| ضمیر بازتابی غیرمشترک | UCREFX | | | | | | | | | |
| ضمیر متقابل | RECPR | | | | | | | | | |
| ندارد | | | | | ندارد | | اسم خاص | proper noun | PROPN | 12 |
| | | علامت نگارشی | PUNC | جداکننده | Delimiter | DELM | علامت نگارشی | punctuation | PUNCT | 13 |
| | | نقش نمای وابستگی | SUBR | حرف ربط | Conjunction | CON | حرف ربط وابسته‌ساز | subordinating conjunction | SCONJ | 14 |
| ندارد | | | | نماد | Symbol | SYM | نماد | symbol | SYM | 15 |
| | | فعل | V | فعل امری | Imperative verb | V_IMP | فعل | verb | VERB | 16 |
| | | | | فعل زمان گذشته | Past tense verb | V_PA | | | | |
| | | | | فعل گذشته کامل | Past participle verb | V_PP | | | | |
| | | | | فعل ربطی | Verb copula | V_COP | | | | |
| | | | | فعل زمان حال | Present tense verb | V_PRS | | | | |
| | | | | فعل شرطی (التزامی) | Subjunctive verb | V_SUB | | | | |
| معلوم | ACT | | | | | | | | | |
| مجهول | PASS | | | | | | | | | |
| وجهی | MODL | | | | | | | | | |
| | | شبه جمله | PSUS | واژه خارجی | Foreign word | FW | دیگر واژه ها | other | X | 17 |

## ۴ فریم‌ورک *Predpatt*

Predpatt مخفف عبارت Predicate-Argument Extraction from Universal Dependencies یک کتابخانه‌ی قدرتمند توسعه‌یافته است که به عنوان چارچوبی از الگوهای استخراج گزاره (Predicate)-آرگومان قابل توسعه، قابل تفسیر و زبان خنثی ارائه می‌شود. PredPatt عملیات تجزیه نحوی عمیق پروژه‌ی وابستگی جهانی را به یک لایه معنایی کم‌عمق اولیه پیوند می‌دهد. این می‌تواند پایه‌ای برای لایه‌بندی آینده‌ای از حاشیه‌نویسی‌های معنایی در بالای درخت‌های وابستگی جهانی باشد. هم‌چنین به‌طور جداگانه می‌تواند به‌عنوان یک مؤلفه زبان‌شناختی به‌خوبی از سازوکار «IE Universal» در نظر گرفته شود.

این کتابخانه در سراسر زبان‌های دنیا کار می کند. *Predpatt* با سایر سیستم های برجسته‌ی Open IE مقایسه شده است و نتایج به‌دست‌آمده نشان می‌دهد که PredPatt به بهترین دقت و یادآوری دست می یابد.

کاربرد اصلی *PredPatt* استخراج روابط از جملات ورودی به آن است. به این صورت که تلاش می‌کند با استفاده از الگوریتم‌های قوی پیاده‌سازی‌شده، آرگومان‌ها و رابطه‌های موجود در یک جمله را به دست آورد. با استفاده از این آرگومان‌ها و رابطه‌ها می‌توان گراف معنایی (*Semantic Graph*) مرتبط با جملات را ترسیم کرد.

به عنوان نمونه می‌توان کاربرد آن را در زبان انگلیسی مشاهده کرد. همانطور که مشخص است، آرگومان‌های *a* و *b* و *c* از جمله به دست آمده است که هرکدام از آرگومان‌ها نقش *POS* مخصوص به خود را دارند.

*?a extracts ?b from ?c*
*?a: PredPatt*
*?b: predicates*
*?c: text*
*?a gave ?b to ?c*
*?a: Chris*
*?b: the book*
*?c: Pat*

به عنوان نمونه‌ی دیگر در جملات زبان فارسی داریم:

علی دایی بازیکن فوتبال بازنشسته تیم ملی ایران و باشگاه پرسپولیس است.

?*a* بازیکن ?*b* است.

?*a*: علی دایی

?*b*: فوتبالِ (بازنشسته تیم ملی ایران و باشگاه پرسپولیس)

در مثال بالا، *a* و *b* آرگومان‌های ما هستند. یعنی در این جمله، علی دایی و فوتبال آرگومان‌های رابطه محسوب می‌شوند. هم‌چنین کلمه‌ی «بازیکن» به عنوان رابطه‌ی بین این دو آرگومان محسوب می‌شود.

**نکته مهم:** برای تشخیص اینکه کدام کلمه به عنوان رابطه درنظر گرفته شود، ما همیشه تمام کلماتی که بین دو آرگومان می‌آیند را به عنوان رابطه‌ی آن دو آرگومان در نظر می‌گیریم. برای استناد به درستی این کار، از مقاله‌ی Farsbase bkp نوشته دکتر مینایی بهره گرفته‌ایم.

سهراب سپهری شاعر، نویسنده و نقاش اهل ایران بود.

?*a* شاعر بود

?*a*: سهراب سپهری

?*a* نویسنده

?*a*: سهراب سپهری

?*a* نقاش ?*b*

?*a*: سهراب سپهری

?*b*: اهل ایران

مطابق توضیحات قبل، دو آرگومان «سهراب سپهری» و «اهل ایران» داریم و کلمات «شاعر»، «نویسنده» و «نقاش» نیز به عنوان روابط بین این آرگومان‌ها در نظر گرفته می‌شوند. پس سه رابطه در مجموع داریم.

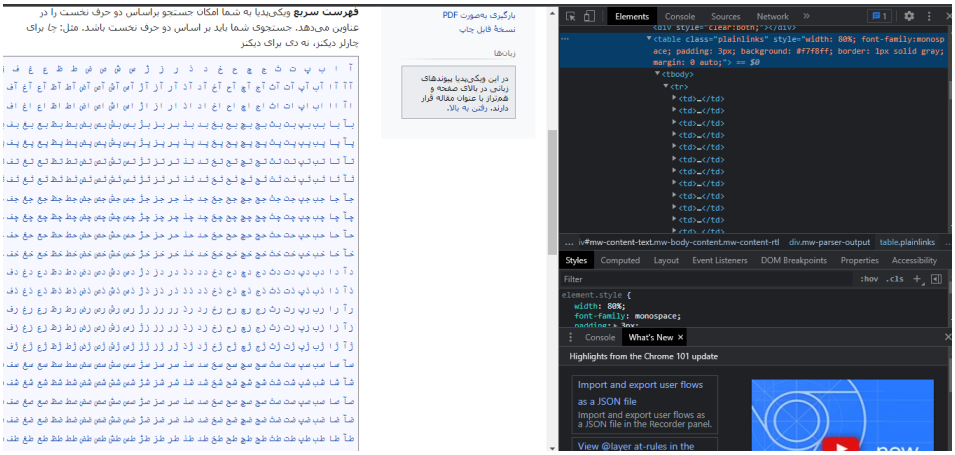
# ۵ آماده‌سازی دیتاست

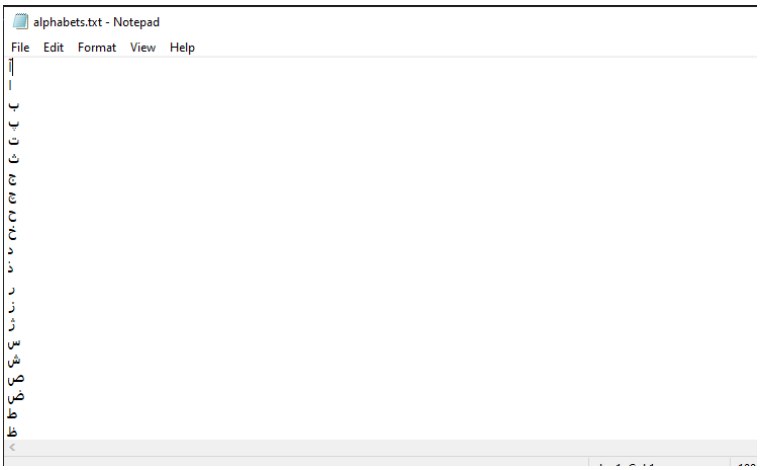
## ۵/۱ مرحله اول: جمع‌آوری داده‌ها

به عنوان اولین فاز، احتیاج به جمع‌آوری مقدار زیادی داده داریم. زیرا برای دستیابی به هدف نهایی، نیاز به تعداد جملات زیادی داریم تا عملیات آموزش و آزمون را روی آنها انجام دهیم و مدل خود را بهبود دهیم. به عنوان سورس و مرجع از دیتاست سایت جامع ویکی‌پدیا استفاده می‌کنیم.

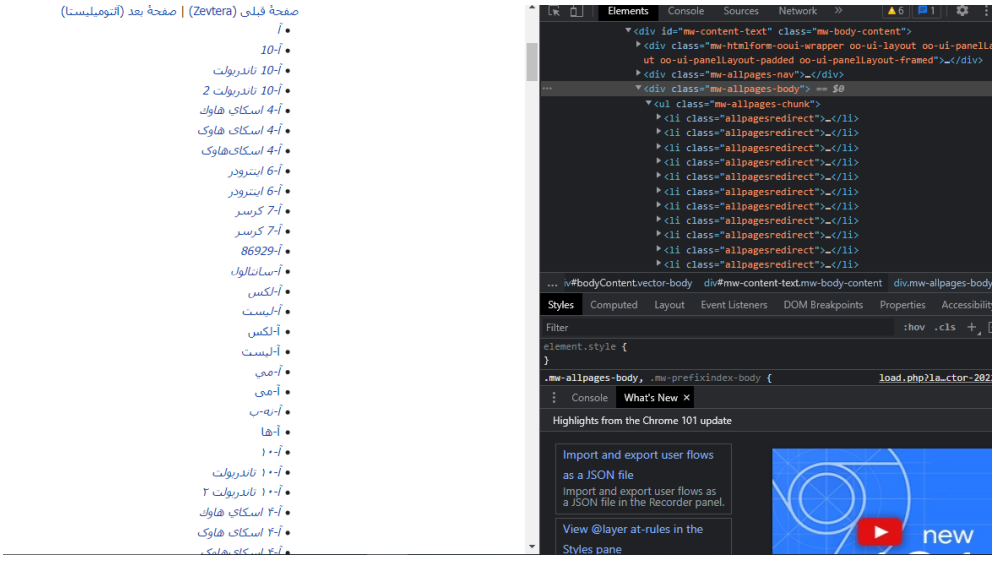
برای دستیابی به داده‌ها و جملات موجود در این وبسایت بزرگ، به Crawl کردن آنها می‌پردازیم. برای این کار، با استفاده از کتابخانه‌ی *BeautifulSoup* اقدام به خزش در فایل‌های *html* می‌کنیم و اطلاعات را مرحله به مرحله استخراج می‌کنیم. توضیحات مربوط به این بخش به طور کامل در گزارش فاز قبلی آمده است و اینجا به طور خلاصه به مراحل انجام کار اشاره می‌کنیم و به همین بسنده می‌کنیم.

در گام اول، ابتدا تمامی الفبای موجود در این وبسایت را به دست می‌آوریم:

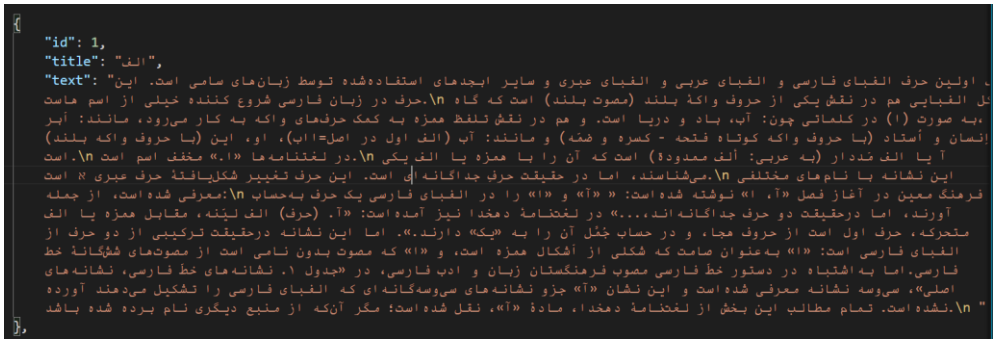





در گام بعد، اقدام به دستیابی به تمام موضوعاتی که با آن حروف شروع می‌شوند، می‌کنیم. به این خاطر که داده‌های مورد نیاز ما حداکثر سی هزار جمله می باشد، فقط موضوعاتی که در صفحه اول حروف هستند را در نظر می گیریم و همچنین حروفی از فهرست که بیش از یک حرف دارند را هم حذف می کنیم.

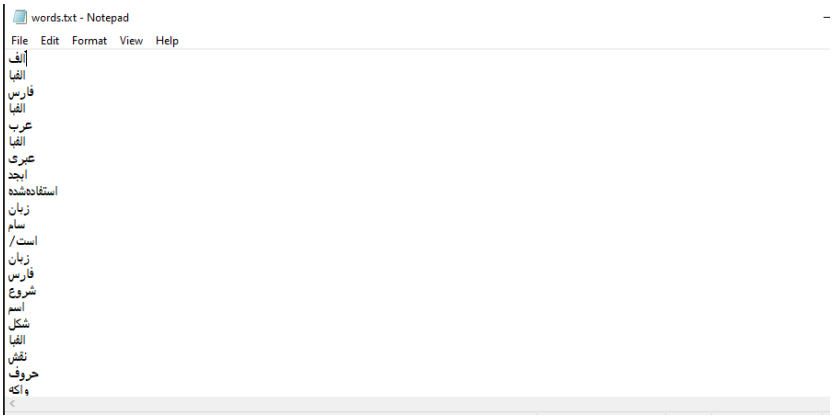



حال با ایجاد یک درخواست (*request*) به هرکدام از لینک‌های موجود، اقدام به دستیابی به جملات موجود در هر صفحه می‌کنیم. برای هر صفحه یک موجودیت *json* شامل id، title و subject درست می‌کنیم.



## ۵/۲ مرحله دوم: پاکسازی داده‌ها

در این مرحله با استفاده از کتابخانه‌ی هضم که همانند *nltk* در زبان انگلیسی است، اقدام به نرمال‌سازی هرکدام از جملات می‌کنیم. سپس برای به دست آوردن کلمات، با *stemming* ریشه اصلی آن ها را به دست می آوریم و در نهایت چک می کنیم که این کلمات بین *stop words* های زبان فارسی هستند یا خیر و اگر در فایل *stop words* نبود، آن را در لیست کلمات در فایل *words.txt* می نویسیم که در شکل زیر این فایل را می بینیم.

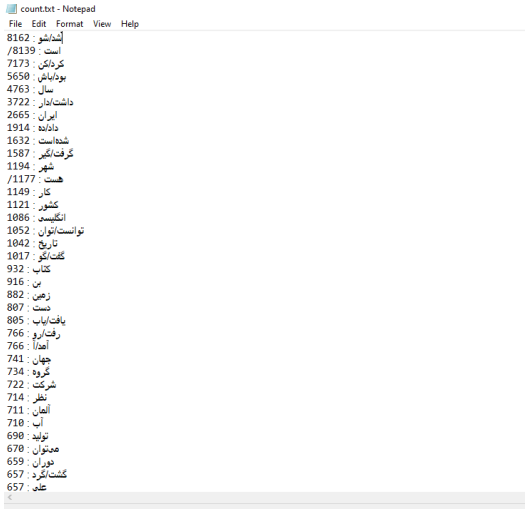



## ۵/۳ مرحله سوم: پردازش آماری داده‌ها

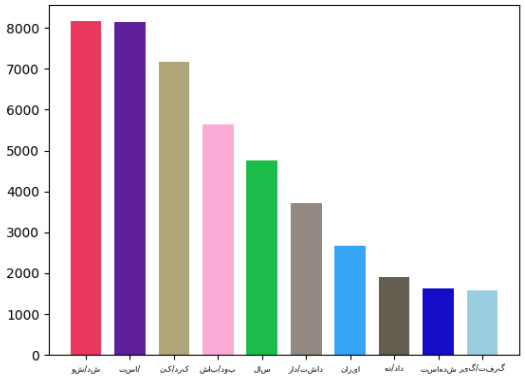
در نهایت تمامی جملات به دست آمده را جدا می‌کنیم و در فایل *sentences.json*، همراه با آیدی و موضوع آن می نویسیم. از این جملات به عنوان دادگان اصلی در جهت استفاده در بخش‌های بعدی استفاده می‌کنیم.

تعداد جملات نهایی ۳۱۳۶۳ عدد می‌باشند که برای استفاده در وابستگی‌های جهانی مورد استفاده خواهند بود.

همچنین برای این که هر کلمه را با تعداد تکرارش داشته باشیم، یک دیکشنری تعریف می کنیم و هر بار که کلمه ای تکرار شد یکی به تعداد تکرار آن اضافه می کنیم و در نهایت آن را پس از *sort* کردن بر اساس تکرار بیشتر، در فایل *count.txt* ذخیره می کنیم.




نمودار فراوانی کلمات در زیر قابل مشاهده است:

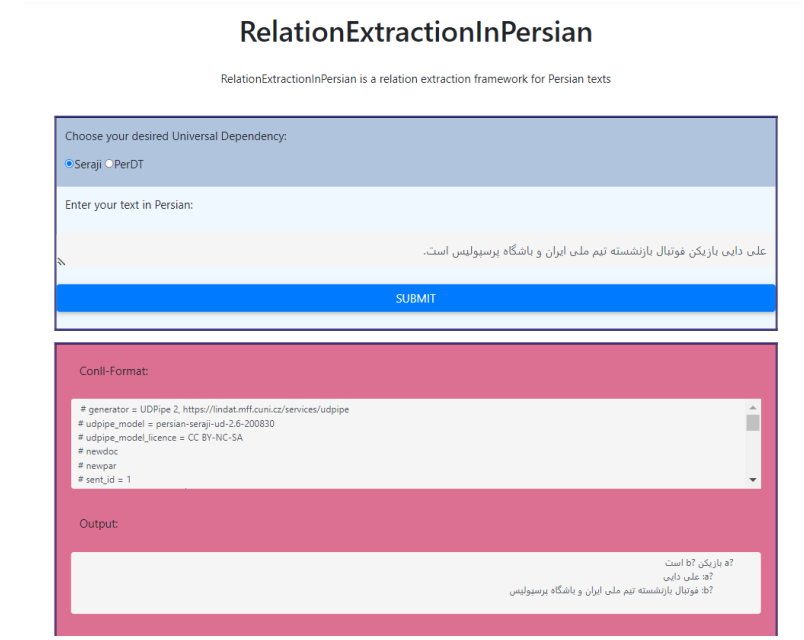


## ۶ پیاده‌سازی فرایند استخراج رابطه

در این مرحله، اقدام به پیاده‌سازی سیستمی در جهت استخراج روابط موجود در جملات می‌کنیم. به نحوی که سیستم مربوط با دریافت یک یا تعدادی از جملات و هم‌چنین انتخاب نوع وابستگی جهانی (*Seraji* یا *PerDT*)، آرگومان‌های مربوطه و رابطه‌ی بین آرگومان‌ها را استخراج کند.

**توجه:** لازم به ذکر است در جهت توسعه‌ی یک سیستم نرم‌افزاری مطلوب در جهت راحتی مشتریان و *UI/UX* بهتر، اقدام به پیاده‌سازی این سیستم در قالب یک سیستم *client/server* کردیم و تلاش کردیم صفحه‌ای به عنوان رابط کاربری برای کاربر نمایش دهیم تا به واسطه‌ی *API* های سرور بتواند به استخراج آرگومان‌های مربوطه از جملات دلخواه خود بپردازد. پیاده‌سازی این بخش نرم‌افزاری جزو قسمت‌های پروژه نبوده است.

برای بخش سرور از فریم‌ورک *Django* مربوط به زبان پایتون استفاده کرده‌ایم. با استفاده از این فریم‌ورک، یک صفحه طراحی کرده‌ایم که کاربر بتواند با ورودی متن مربوطه و انتخاب نوع وابستگی جهانی و انتخاب گزینه *Submit*، نتیجه را مشاهده کند.



هم‌چنین اقدام به پیاده‌سازی *API* های مربوطه برای هر دو وابستگی جهانی پرداختیم که بتوان از دیوایس‌ها و برنامه‌های دیگر نیز با استفاده از این *API* ها ارتباط برقرار کرد.

برای استخراج آرگومان‌های مربوطه به پیاده‌سازی هر دو وابستگی جهانی پرداختیم:

## ۶٫۱ استخراج آرگومان‌ها با استفاده از وابستگی جهانی *Seraji*

در این قسمت، پس از دریافت *ud_type* و *text* از کاربر، در صورتی که کاربر وابستگی جهانی سراجی را انتخاب کرده باشد، اقدام به تولید یک فایل موقتی به نام *file.txt* می‌کنیم. تمامی جملات ورودی را در این فایل می‌ریزیم. سپس اقدام به اجرا کردن یک کامند از طریق *bash* می‌کنیم.

دلیل استفاده از این فایل موقتی به دلیل نیاز این کامند به خواندن اطلاعات از فایل دارد. (*data=@file.txt*) مقدار پارامتر *model* را برابر *persian* قرار دادیم تا از مدل‌های مربوط به زبان فارسی برای استخراج رابطه استفاده شود. از *normalized_space* به عنوان *tokenizer* استفاده می‌کنیم و هم‌چنین برای تجزیه‌گر (پارسر) از *api* ای که برای *udpipe* تعبیه شده است، استفاده می‌کنیم.

در صورتی که اجرای کامند فوق دچار خطا نشود، از کلید *result* در نتایج این دستور می‌توانیم فرمت *conllu* موردنظر را به دست آوریم.

منظور از *Conllu Format* چیست؟

*CoNLL* نامی معمولی برای قالب‌های *TSV* (مقادیر جدا شده با تب مانند *csv*) در *NLP* است و در واقع فرمتی خاص برای نمایش مقادیر است که در این حوزه بسیار پر کاربرد است. بسیاری از فرمت های مختلف *CoNLL* وجود دارد زیرا *CoNLL* هر سال یک کار مشترک متفاوت است که در کنفرانس سالانه *NLP* تصمیم‌گیری می‌شود. فرمت مشهور آن به این صورت است که:

- هر توکن در یک خط نشان داده می شود.
- هر جمله با یک خط خالی (*NewLine*) از جمله بعدی جدا می شود.
- هر ستون نشان‌دهنده یک *Annotation* است.

- هر کلمه در یک جمله دارای تعداد ستون های یکسانی است. (در برخی قالب ها: هر کلمه در مجموعه تعداد ستون های یکسانی دارد.)
- و... که در این گزارش نمی‌گنجد.

در ادامه، با استفاده از تابع *load_conllu* اقدام به خواندن فرمت *Conll* تولید شده می‌کنیم و تمامی تگ‌های *POS* به‌دست‌آمده و اطلاعات لازم را در یک لیست نگه‌داری می‌کنیم.

به عنوان مرحله نهایی و در قدم آخر، با ورودی دادن این لیست به کلاس *PredPatt* روابط مورد نظر و درخت رابطه ایجاد می‌شود. برای اینکه آرگومان‌های مربوطه را ببینیم، متود *pprint()* را از این کلاس صدا می‌زنیم.

مواردی که در بالا توضیح داده شد، در تصویر زیر قابل مشاهده است:

```
if request.method == "POST":
    ud_type, text = request.POST.get("ud_type"), request.POST.get("input_text")

    if ud_type == 'seraji':
        with open('file.txt', 'w', encoding='utf-8') as f:
            f.write(text.replace('\r', '').strip())
            f.close()

        bash_command = """curl -F data=@file.txt -F model=persian -F tokenizer=normalized_spaces -F tagger= -F
        parser= http://lindat.mff.cuni.cz/services/udpipe/api/process """
        try:
            send_order = subprocess.check_output(bash_command.split(), shell=True)
        except subprocess.CalledProcessError as e:
            os.remove("file.txt")
            raise RuntimeError("command '{}' return with error (code {}): {}".format(e.cmd, e.returncode, e.output))

        os.remove("file.txt")
        conll = json.loads(send_order)['result']
        ppatt = PredPatt([ud_parse for sent_id, ud_parse in load_conllu(conll)][0])
        return render(request, 'service.html',
                      {'input_text': text, 'ud_type': ud_type, 'conll_text': conll, 'output_text': ppatt.pprint()})
```

## ۶٫۲ استخراج آرگومان‌ها با استفاده از وابستگی جهانی *PerDT*

پیاده‌سازی این قسمت در بعضی بخش‌ها مشابه وابستگی جهانی سراجی است. در ابتدا اقدام به دانلود مدل‌های شبکه‌های عصبی موازی برای زبان فارسی می‌کنیم. برای دانلود این مدل‌ها از پکیج *stanza* استفاده می‌کنیم. با استفاده از متود *Pipeline* اقدام به انتخاب شبکه عصبی پیش‌فرض در زبان فارسی می‌کنیم. حال که مدل مربوطه را به دست آوردیم، جمله ورودی را به عنوان ورودی به مدل می‌دهیم.

خروجی مدل تعدادی جملات برای ما خواهد بود که این جملات شامل تعدادی کلمات هستند. در این مرحله بر خلاف روش سراجی که خودش همه نقش‌ها را به ما می‌داد، خودمان باید یکی یکی ویژگی‌های مورد نیاز را به ازای هر کلمه به دست آوریم. ویژگی‌هایی از جمله شناسه، متن کلمه، لم، جایگاه آن در متن، *xpos*، حرف شروع و پایانی و...

تمامی ویژگی‌های مورد نیاز را خودمان دستی تولید می‌کنیم و در قالب یک رشته به همدیگر می‌چسبانیم تا مانند سراجی به PredPatt دهیم تا خروجی را برایمان تولید کند. در واقع، فرمت conll را در اینجا خودمان تولید کردیم و به Predpatt دادیم. بر خلاف روش سراجی که با ران کردن کامند مربوطه این کار را انجام می‌داد.

مواردی که در بالا توضیح داده شد، در تصویر زیر قابل مشاهده است:

```
elif ud_type == "perdt":
    stanza.download('fa')  # This downloads the Persian models for the neural pipeline
    nlp = stanza.Pipeline('fa')  # This sets up a default neural pipeline in Persian
    doc = nlp(text)
    for sentence in doc.sentences:
        res = "# text = " + text.strip()
        res = res.strip() + '\n'
        for word in sentence.words:
            res = res + str(word.id) + "\t" + str(word.text) + "\t" + str(word.lemma) + "\t" + \
                  str(word.pos) + "\t" + str(word.xpos) + "\t" + str(word.feats) + "\t" + \
                  str(word.head) + "\t" + str(word.deprel) + "\t" + str(word.start_char) + "\t" +\
                  str(word.end_char)
            res = res + '\n'

    ppatt = PredPatt([ud_parse for sent_id, ud_parse in load_conllu(res)][0])
    return render(request, 'service.html',
                  {'input_text': text, 'ud_type': ud_type, 'conll_text': res, 'output_text': ppatt.pprint()})
```

## ۶٫۳ به دست آوردن روابط و اجرای وابستگی *Seraji* روی تمامی جملات

در این مرحله، از *API* هایی که در مرحله‌ی قبل برای استخراج آرگومان‌ها با استفاده از وابستگی جهانی سراجی پیاده‌سازی کردیم، استفاده می‌کنیم تا هر یک از آرگومان‌ها را در تمامی جملاتی که داریم، به همراه رابطه‌ی بین این آرگومان‌ها به دست آوریم. در نهایت، دیتاست محدودتری به دست می‌آوریم که شامل تمامی جملاتی است که حداقل یک رابطه در درون خود دارند. (ممکن است تعدادی از جملات طبق مدل *Seraji* اصلا شامل رابطه نباشند.)

ابتدا اقدام به باز کردن فایل جملات و همچنین فایل جملات بعلاوه روابط آنها (برای نوشتن) می‌کنیم. سپس، با استفاده از آدرسی که برای *API* مدل سراجی تعیین کردیم، در حالتی که سرور در حال اجرا باشد، اقدام به ریکوئست زدن می‌کنیم.

```
sentences_file = open('../data/sentence_broken/sentences.json', 'r', encoding='utf8')
results = open('../data/sentence_broken/sentences_with_relations.json', 'w', encoding='utf8')

seraji_request_url = 'http://127.0.0.1:8000/api/seraji'

sentences_json = json.load(sentences_file)

counter = 29000
subject_counter = 2159
error_counter = 0
dataset = []
```

در واقع، روی تک تک جملات دیتاست خود، آنها را به عنوان ورودی *input_text* به *API* مربوطه تحت ریکوئست می‌دهیم. سپس، از خروجی به‌دست‌آمده با استفاده از عملیات مربوط به کار با رشته‌ها در پایتون، ابتدا اطمینان پیدا می‌کنیم که حتما دو آرگومان وجود داشته باشند. (در صورت وجود تک آرگومان، رابطه تعریف نمی‌شود.)

سپس در صورت وجود دو آرگومان، چک می‌کنیم که *subject* صفحه ویکی‌پدیا در آرگومان اول جمله باشد و بالعکس. در صورت وجود، آرگومان دوم را *object* می‌نامیم و تمامی کلمات بین این دو آرگومان را به عنوان رابطه‌ی بین این دو آرگومان در نظر می‌گیریم. (پیش‌تر توضیح داده شد که با استناد به مقاله‌ی *Farsbase pbk* دکتر مینایی این فرض را کرده‌ایم.)

```python
for sentence_dict in sentences_json[29000:]:
    try:
        response = requests.post(seraji_request_url, json={'input_text': sentence_dict['text']})
        output_text = response.json()['output_text']

        subj_index = output_text.find('?a:')
        obj_index = output_text.find('?b:')
        if subj_index != -1 and obj_index != -1:
            subject = output_text[subj_index + 3:obj_index].strip()
            if sentence_dict['subject'] in subject or subject in sentence_dict['subject']:
                object = output_text[obj_index + 3:output_text.find('\n', obj_index + 3)]
                if 'SOMETHING' not in object and len(subject) >= 2:
                    relation = output_text[output_text.find('?a') + 2: output_text.find('?b')]

                    subject = remove_extra_whitespaces(subject)
                    object = remove_extra_whitespaces(object)
                    relation = remove_extra_whitespaces(relation)
                    dataset.append(
                        {
                            'id': subject_counter,
                            'text': sentence_dict['text'],
                            'subject': subject,
                            'object': object,
                            'relation': relation
                        }
                    )
                    subject_counter += 1
                    # if subject_counter >= 6:
                    #     break

                    print('Subject counter: {}'.format(subject_counter))

        counter += 1
        print(str(counter) + " from 31363 (", end='')
        print("%4.2f" % ((counter / 31363) * 100), end='')
        print("%)")

        # if counter >= 29000:
        #     break
    except:
        error_counter += 1
        print('Error #{} occurred!'.format(error_counter))

results.write(json.dumps(dataset, ensure_ascii=False))
results.close()
```

در انتها، با استفاده از تابعی که خودمان تعریف کرده‌ایم (*remove_extra_whitespaces*)، اقدام به حذف فواصل اضافی در ابتدا و انتهای رشته‌ها می‌کنیم و تمامی روابط به‌دست‌آمده را در قالب یک انتیتی داخل لیست *dataset* ذخیره می‌کنیم.

```python
def remove_extra_whitespaces(word):
    if len(word) >= 1:
        if word[0] == ' ':
            word = word[1:]
        if word[-1] == ' ':
            word = word[:-1]

    return word
```

در انتها اقدام به ذخیره‌ی تمامی انتیتی‌های دیتاست در درون یک جیسون واحد در فایل *sentences_with_relations.json* می‌کنیم.

فرمت هریک از موجودیت‌های خروجی به صورت زیر خواهد بود:

```
    {
        "id": 327,
        "text": "شنا باعث کاهش آثار زیان‌بار استرس می‌شود.",
        "subject": "شنا",
        "object": "کاهش آثار زیان‌بار استر",
        "relation": "باعث"
    },
```

```
{
    "id": 82,
    "text": "میلان در اولین دهۀ قرن، یکی از قدرتمندترین باشگاه‌های اروپا و جهان بود.",
    "subject": "میلان",
    "object": "اولین دهۀ قرن",
    "relation": "در"
},
```

Formatted: Centered

Formatted: Centered

```
{
    "id": 67,
    "text": "مجموعه شعرهای احمد شاملو (دوجلدی) در آلمان غربی منتشر می‌شود.",
    "subject": "مجموعه شعرهای احمد شاملو ( دوجلدی)",
    "object": "آلمان غرب",
    "relation": "در"
},
```


همانطور که مشاهده می‌شود، توضیحات پیشین داده‌شده در مورد نتایج به‌دست‌آمده صدق می‌کنند و می‌بینیم که اکثر جملات به درستی آرگومان‌هایشان و روابط بین آن آرگومان‌ها استخراج شده است.

**توجه:** لازم به ذکر است که برای اجرا کردن این وابستگی جهانی روی جملات، با توجه به تعداد زیاد جملات و طولانی‌شدن فرایند، این کار در ۵ مرحله مجزا انجام گرفته و در نهایت، خروجی‌ها با همدیگر ادغام گشته‌اند و در قالب یک فایل جیسون درآمده‌اند. فایل نهایی شامل مجموعا ۲۳۹۲ روابط استخراج شده از جملات می‌باشد. طبیعتا تعداد زیادی از جملات به دلایلی از جمله عدم وجود آرگومان، تک آرگومانی، نبودن موضوع در آرگومان اول و... حذف شده‌اند و مقادیری باقی مانده اند که بتوانیم در بخش‌های بعدی از آنها استفاده کنیم.

تصاویر زیر، نمایانگر مراحلی از اجرای کد مربوطه روی جملات هستند. تصویر اول پیشروی ریکوئست‌ها به ازای هر جمله را نشان می‌دهد و تصویر دوم نیز اطلاعاتی در مورد سرعت آپلود و دانلود و... در ازای هر بار دستیابی به مدل *Conllu* با استفاده از دستور اجرایی در *bash* می‌باشد. (این عملیات با صدا زدن *API* مربوط به سرویس *udpipe* انجام می‌گیرد.)

Formatted: Centered