برای شروع مواردی که باید انجام شوند را فهرست می کنیم:

1. پیدا کردن فایل متنی کتب و واژه نامه مغولی
2. تبدیل فایل های متنی به فرمت خوانا برای برنامه (مانند txt)
3. نوشتن برنامه با قابلیت های زیر:
   * باز کردن و دریافت متن از فایل و واژه نامه
   * جدا کردن کلمات از یکدیگر (Word Tokenizing)
   * بررسی ریشه کلمات فایل و مقایسه با واژه نامه
     + تبدیل کلمات به ریشه ها (کتابهایش ← کتاب، می خوردیم ← خور)
     + مقایسه لغات و شمارش آنها
     + خروجی نتیجه خام
4. دادن ورودی ها (فایل های کتب، واژه نامه مغولی و کلمات ایست) به برنامه و دریافت خروجی
5. تحلیل خروجی ها

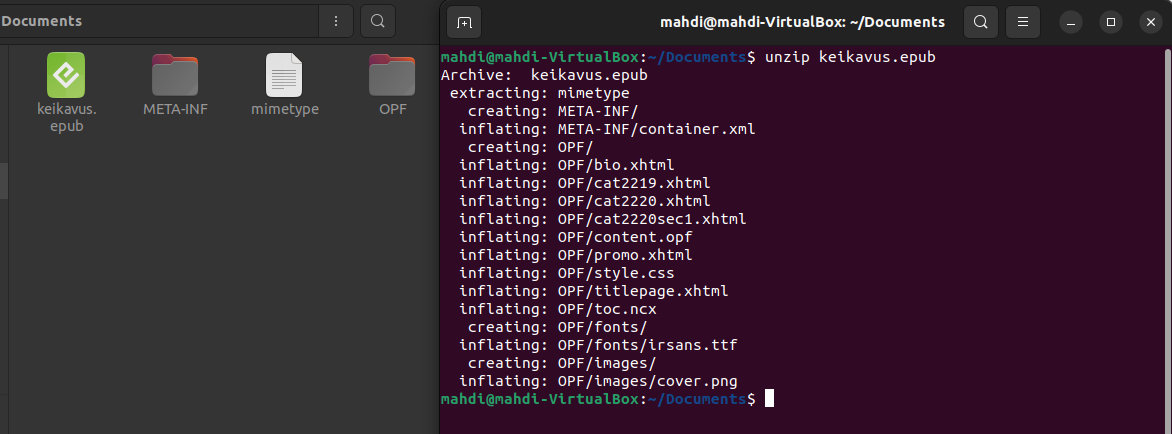
**پیدا کردن فایل کتب و نوشته های شاعران**

سایت منبع باز (Open source) [گنجور1](https://ganjoor.net/) که به عنوان مرجع انتخاب شده وارد می شویم. در بخش [کتابخانه گنجور](http://epub.ganjoor.net/) می توانیم کتاب ها را با فرمت epub دانلود کنیم (تصویر 1).

تصویر 1-نمایی از سایت کتابخانه ی گنجور

1.«گنجور» مجموعه‌ای تحت وب از آثار سخنسرایان پارسی‌گو است که با استفاده از آن می‌توان ضمن مرور به تفکیک نام شاعر و نام آثار او، در بین آثار جستجو کرد. این مجموعه در ابتدا به کمک نرم‌افزار مدیریت محتوای وردپرس راه‌اندازی شده بود و از سال ۱۴۰۰ به بعد به پشتوانه نرم‌افزار اختصاصی بازمتن خود کار می‌کند.

سپس فایل را از حالت کتاب الکترونیک (epub) خارج می کنیم.



تصویر2 – نمای متحوای باز شده کتاب نمونه (کیکاووس) با فرمت epub

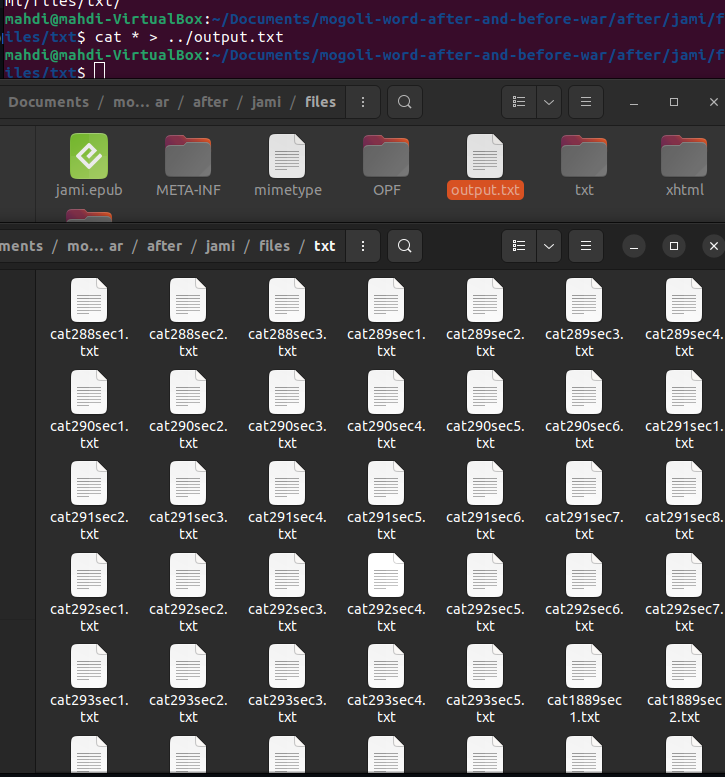
با بررسی فایل های باز شده ی epub متوجه می شویم، فایل های در پوشه OPF به شکل \*sec\*.xhtml حاوی محتوای کتاب ما هستند.

حال باید فایل های xhtml را به فرمت خوانا تری برای بررسی (همچون txt) در برنامه تبدیل کنیم.

سایت ها و برنامه های مختلفی برای تبدیل xhtml به txt وجود دارد. ما از [aspose](https://products.aspose.app/words/conversion/xhtml-to-html) استفاده کردیم. هر فایل xhtml به یک فایل txt تبدیل می شود.

سپس با ادغام فایل های txt، همه را به یک فایل تبدیل کردیم. با دستور زیر می توانیم این کار را انجام دهیم. (فرض شده که تمامی فایل ها در یک پوشه هستند و هیچ فایل دیگری در آن پوشه نیست.) (تصویر 3)

|  |
| --- |
| cat \* > your\_address/file\_name.txt |



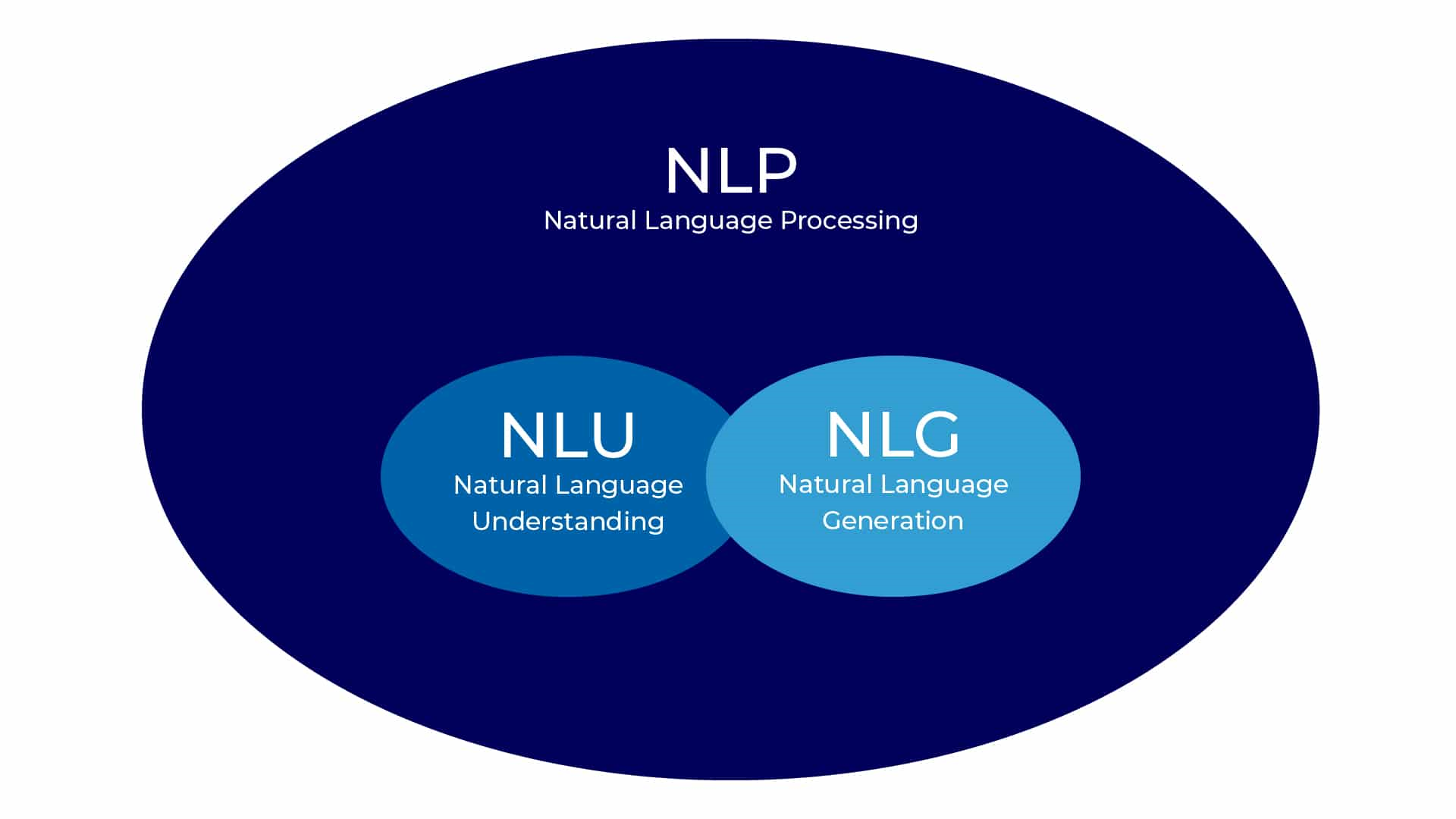
تصویر3 – نمای تبدیل فایل های txt به یک فایل txt

درنهایت یک فایل را برای دادن به برنامه آماده می شود.

**نوشتن برنامه**

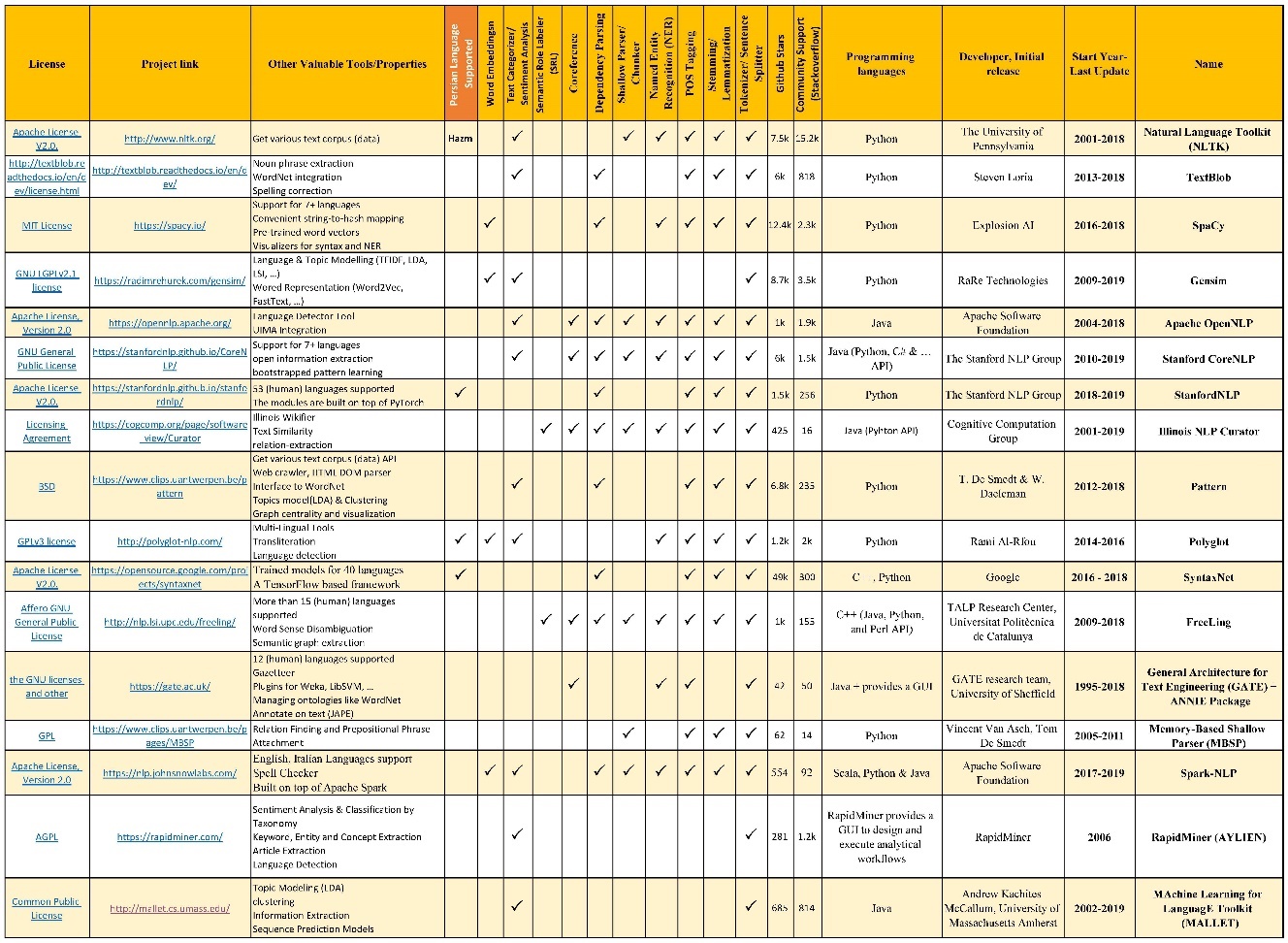
برای نوشتن برنامه از زبان پایتون و کتابخانه ی wordcloud\_fa که متشکل از کتابخانه های wordcloud، hazm و nltk است استفاده کردیم. ابتدا توضیحات مختصری درباره ی کتابخانه ها می دهیم.

کتابخانه های hazm و nltk به ترتیب برای پردازش زبان های فارسی و انگلیسی کاربرد دارند.

پردازش زبان طبیعی (Natural Language Processing) که به اختصار NLP نیز نامیده می‌شود، روشی است برای درک زبان انسانی برای رایانه؛ این علم یکی شاخه‌های دانش هوش مصنوعی محسوب می‌شود و به رایانه‌ها کمک می‌کند تا با آگاهی از چگونگی استفاده بشر از زبان، زبان انسانی را درک کند. پردازش زبان طبیعی یک دانش پیچیده و دشوار است. این شاخه از هوش مصنوعی خود به دو شاخه دیگر یعنی NLG و NLU تقسیم می شوند. Natural Language Understanding یا به اختصار NLU به فارسی درک زبان طبیعی بر درک خواندن ماشینی از طریق قواعد دستور زبانی و موضوع تمرکز دارد و ماشین‌ها را قادر می‌سازد تا معنای مورد نظر یک جمله را تعیین کنند. Natural Language Generation یا به اختصار NLG به فارسی تولید زبان طبیعی بر تولید متن یا ساخت متن به زبان انگلیسی یا زبان‌های دیگر توسط یک ماشین و بر اساس یک مجموعه داده معین تمرکز می‌کند (تصویر 4).

تصویر4 – ارتباط بین NLP، NLG و NLU با یکدیگر

در جدول زیر کتابخانه های معروف NLP آورده شده اند (جدول 1).



جدول 1 – جدول مقایسه کتابخانه های مشهور NLP

از کتابخانه های فارسی معروف در این حوزه می توان به Hazm، Stanford، farsiNLPTools و parsivar اشاره کرد.

کتابخانه‌ NLTK (Natural Language Toolkit) یکی از جامع‌ترین و قدیمی‌ترین کتابخانه‌های پردازش زبان طبیعی در پایتون است. این کتابخانه پایه و استانداردی برای کتابخانه‌های پردازش متن محسوب شده و برای کاربردهای پژوهشی فوق‌العاده است. یکی از ویژگی‌های خوب این کتابخانه امکان اتصال به پیکره‌های مختلف متنی است.

کتابخانه هضم با استفاده از کتابخانه NLTK در سال ۱۳۹۲ توسط دانشجویان دانشگاه علم و صنعت برای پردازش زبان فارسی توسعه داده‌شده است. در ابتدا هضم تنها برای زبان پایتون و سیستم‌عامل لینوکس طراحی‌شده بود، اما اکنون برای زبان‌های جاوا و #C نیز قابل‌استفاده است. نسخه جاوایی این کتابخانه با عنوان JHazm منتشرشده است. ازجمله ویژگی‌های این کتابخانه مرتب کردن متون، بخش‌بندی، ریشه‌یابی کلمات، تحلیل صرفی و تجزیه نحوی جملات، سازگاری با NLTK و پشتیبانی از نسخه‌های ۲ و ۳ پایتون هست.

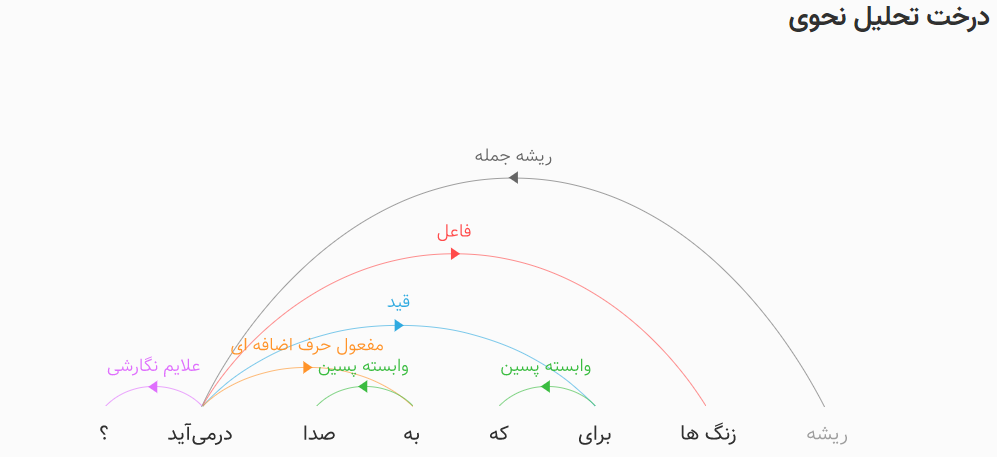
همچنین میزان دقت این کتابخانه در تحلیل متون در جدول زیر آمده است (جدول 2).

|  |  |
| --- | --- |
| **Module name** | **Precision** |
| **Lemmatizer (ریشه یابی)** | 89.9% |
| **Chunker (تشخیص گروه های اسمی و فعلی)** | 89.9% |
| **POSTagger (تحلیل نقش کلمات)** | 97.1% |
| **DependencyParser (درخت وابستگی)** | 97.1% |

جدول 2 – میزان دقت کتابخانه هضم

در ادامه تعدادی دستورات کاربردی هضم آورده شده:

|  |
| --- |
| >>> normalizer = Normalizer()  >>> normalizer.normalize ( 'اصلاح نویسه ها و استفاده از نیم فاصله پردازش را آسان می کند' )  'اصلاح نویسه ها و استفاده از نیم فاصله پردازش را آسان می کند'  >>> sent\_tokenize ( 'ما هم برای وصل کردن آمدیم! ولی برای پردازش، جدا بهتر نیست؟' )  [ 'ما هم برای وصل کردن آمدیم!' , 'ولی برای پردازش، جدا بهتر نیست؟' ]  >>> word\_tokenize ( 'ولی برای پردازش، جدا بهتر نیست؟' )  [ 'ولی', 'برای', 'پردازش'، '،'، 'جدا', 'بهتر', 'نیست', '؟' ]  >>> stemmer = Stemmer()  >>> stemmer.stem ( 'کتاب ها' )  'کتاب'  >>> lemmatizer = Lemmatizer()  >>> lemmatizer.lemmatize ( 'می روم' )  'رفت#رو'  >>> tagger = POSTagger ( model = 'resources/postagger.model' )  >>> tagger.tag ( word\_tokenize( 'ما بسیار کتاب می‌خوانیم' ))  [ ( 'ما' , 'PRO' ), ( 'بسیار' , 'ADV' ), ( 'کتاب' , 'N' ), ( 'می خوانیم' , 'V' ) ]  >>> chunker = Chunker( model= 'resources/chunker.model' )  >>> tagged = tagger.tag ( word\_tokenize( 'کتاب خواندن را دوست داریم' ))  >>> tree2brackets( chunker.parse (tagged))  '[کتاب خواندن NP] [را POSTP] [دوست داریم VP]'  >>> parser = DependencyParser (tagger = tagger, lemmatizer = lemmatizer ) |

همچنین این کتابخانه حتی توانایی رسم درخت تحلیل نحوی جملات را دارد. به عنوان مثال دستور زیر درخت تحلیل نحوی جمله " زنگ‌ها برای که به صدا درمی‌آید؟" را رسم می کند (تصویر 5)

|  |
| --- |
| >>> parser.parse ( word\_tokenize( 'زنگ‌ها برای که به صدا درمی‌آید؟' ))  <DependencyGraph with 8 nodes> |

تصویر 5 – نمودار تحلیل نحوی

کتابخانه wordcloud\_fa برای تولید ابر کلمات در زبان فارسی کاربرد دارد. ابر کلمات به تصویری می گویند که در آن لغات متنی به نمایش در می آید به صورتی که در آن کلمه ای که بزرگ تر است بیشتر تکرار شده اند. در تصویر زیر می توانید ابر کلمات مجموعه آثار مولانا مشاهده کنید (تصویر 6).



تصویر 6 – ابر کلمات مجموعه آثار مولانا

از آنجایی که در این کتابخانه واژه ها از هم جدا و شمرده می شوند ما می توانیم با تغییراتی در کد های آن، خروجی مد نظر خود را ستخراج کنیم و با ایجاد شرطی مغولی بودن آن را نیز بررسی کنیم.

موضوع دیگری که در این برنامه اهمیت دارد داشتن فهرستی از کلمات ایست (stop word) است. کلمه ایست به کلماتی از جمله از، به، است، گفت، یا، ای و... می گویند که مفهوم خاصی ندارند ویا صرفا برای تکمیل جمله به کار می روند و اهمیت معنایی زیادی ندارند. به همین علت و برای بهتر شدن تحلیل، لازم است این لغات از متن حدف شوند. با جست و جویی در اینترنت به [فهرستی](https://github.com/kharazi/persian-stopwords) از این کلمات رسیدیم. این فهرست شامل 1900 کلمه بود. درهنگام بررسی و تحلیل تعدادی لغت نیز به آن اضافه کردیم. لازم به ذکر است که این فهرست کاربرد عمومی دارد. بنابراین برای تحلیل متون مختلف نیاز به تغییر و اختصاصی کردن این فهرست می باشد. به عنوان مثال فهرست کلمات ایست برای متون عادی، مقاله، نظرات مشتریان، موجود در شبکه های اجتماعی و... متفاوت می باشد. بنابراین برای تحلیل و بررسی دقیق تر اشعار و نوشته های ادیبان لازم است که این لغات تک به تک بررسی شوند و در صورت نیاز کم ویا اضافه شوند اما به دلیل کمبود وقت این بررسی انجام نشده است.

درنهایت بعد از جدا کردن و شمارش لغات برای خروجی خام برنامه، فرمت CSV انتخاب شد. «CSV» مخفف عبارت «Comma Separated Values» و به معنای «مقادیر جدا شده با ویرگول» است. اینگونه فایل‌ها در واقع نوعی فایل خام متنی هستند که شامل لیستی از داده‌ها می‌شوند. به صورت رایج فایل‌های یاد شده برای جابه‌جایی داده‌ها بین نرم‌افزارهای متفاوت مورد استفاده قرار می‌گیرند. برای مثال، پایگاه‌های داده و نرم‌افزارهایی از جمله اکسل معمولا از فایل‌های CSV پشتیبانی می‌کنند. همچین این فایل ها سازگاری بسیار خوبی با زبان پایتون دارند و به آسانی نیز قابل تولید هستند.

با توجه به توضیحات بالا، در ارتباط با پروژه "بررسی نفوذ زبان مغولی قبل و بعد از حمله مغول به ایران" به صورت زیر برنامه نویسی و پیاده سازی شد.

ابتدا برای نصب wordcloud\_fa (که شامل تمامی کتابخانه های مورد نیازمان است) دستور زیر را در خط فرمان وارد می کنیم:

|  |
| --- |
| pip install wordcloud-fa |

سپس طبق توضیحات موجود در [اسناد کتابخانه](https://github.com/alihoseiny/word_cloud_fa) کد های اولیه را می نویسیم. حاصل کد main.py، تصویر ابر کلمات است. که مشابه آن را در تصویر 6 مشاهده شد.

main.py

from wordcloud\_fa import WordCloudFa

wodcloud = WordCloudFa(no\_reshape=True, persian\_normalize=True, include\_numbers=False, collocations=False, width=1600, height=800, background\_color="white")

text = ""

with open('book.txt', 'r') as file:

    text = file.read()

wc = wodcloud.generate(text)

image = wc.to\_image()

image.show()

image.save('output.png')

حال لازم است فایل های کتابخانه ی wordcloud\_fa را طبق نیاز خود تغییر بدهیم. کتابخانه ی wordcloud\_fa برای codespase github که سیستم عامل اوبونتو1 است در آدرس زیرقابل دستیابی است:

|  |
| --- |
| /usr/local/python/3.10.4/lib/python3.10/site-packages/wordcloud\_fa/WordCloudFa.py |

بعد از باز کردن فایل مذکور برای اینکه با استفاده از تمامی ابزار های هضم، Header File را به صورت زیر تغییر می دهیم:

# from hazm import Normalizer, word\_tokenize

from hazm import \*

با بررسی ساختار فایل متوجه می شویم لغات در شئی (Class) و تابع (def) زیر جدا می شوند (tokenize).

class WordCloudFa(WordCloud):

    def process\_text(self, text: str) -> Dict[str, int]

1. اوبونتو (به انگلیسی: Ubuntu) یک توزیع گنو/لینوکس بر مبنای دبیان است.

با ایجاد دو ویژگی برنامه را به صورتی که می خواهیم تغییر میدهیم.

الف) تبدیل کلمات به ریشه ی آنها

ب) فیلتر کردن واژگان و جدا سازی واژگان مغولی

برای ایجاد ویژگی های ذکر شده تکه کد زیر را اضافه می کنیم.

        if self.regexp:

            words = re.findall(self.regexp, text, flags)

        else:

            words = word\_tokenize(text)

        #############added

        #####Stemmer

        root = False

        if root:

            buffer = list()

            for i in words:

                buffer.append(Stemmer().stem(i))

            words = buffer

        #####tokeniz and add mogol dic

        mogoli = True

        if mogoli:

            with open("../../mogholi\_dic.txt") as file\_dic:

                text = file\_dic.read()

                mogoli\_words = word\_tokenize(text)

            find\_mogoli = True

            if find\_mogoli:

                buffer = list()

                for i in words:

                    for j in mogoli\_words:

                        if i == j :

                            buffer.append(i)

                words = buffer

        ##############

با تغییر درستی و نادرستی متغیر های root و mogoli در اول دستورات می توان آنها را فعال یا غیر فعال کرد. سپس برای تنظیم قالب خروجی، در شئی و تابع

class WordCloudFa(WordCloud):

    def generate\_from\_frequencies(self, frequencies: Dict[str, float], max\_font\_size=None)

قطعه کد زیر را اضافه می کنیم:

        if self.persian\_normalize:

            words = WordCloudFa.normalize\_words(words)

        new\_frequencies = dict(zip(words, values))

        ############ added

        sort\_dic = sorted(new\_frequencies.items(), key=lambda x:x[1], reverse=True)

        for i in sort\_dic:

            coma = True

            for j in i:

                print(j, end="")

                if coma:

                    print(",", end="")

                    coma = False

            print()

        ############

حاصل کد بالا خروجی ای به شکل فایل CSV است که هم با پایتون و هم با اکسل قابل بررسی است.

سپس فهرست لغات ایست اختصاصی مورد نظر خود را اضافه کنیم برای این منظور به فایل زیر رفته و آن را با فهرست خود جایگزینش می کنیم.

|  |
| --- |
| /usr/local/python/3.10.4/lib/python3.10/site-packages/wordcloud\_fa/stopwords |

با اجرای برنامه به صورت زیر، خروجی در فایلی با فرمت CSV (مثلا log.csv) ذخیره می شود.

|  |
| --- |
| Python main.py > log.csv |

خروجی تفکیک و شمارش کلمات به 4 حالت (همه کلمات، همه کلمات مغولی، ریشه ی همه کلمات، همه ی کلمات با ریشه مغولی) گرفته شد، اما در نهایت فقط خروجی "همه ی کلمات مغولی" و "همه ی کلمات" مورد ارزیابی قرار گرفت. لازم است بررسی شود که چرا ریشه های کلمات به درستی تحلیل نمی شوند.

دو خروجی CSV هر ادیب که شامل "همه کلمات مغولی" و "همه ی کلمات" می شود را باهم ادغام کرده و به صورت یک فایل اکسل در می آوریم که برای تحلیل آماده شوند. (جدول 3)



جدول 3 – نمایی از ادغام دو فایل خروجی حاصل از کتب مولانا که به صورت یک جدول اکسل درآمده است

تمامی کد ها، ورودی ها و خروجی ها در [مخزن گیت هاب مربوطه](https://github.com/mvajhi/mogoli-word-after-and-before-war) بارگزاری شده است.

منابع:

1. پردازش زبان طبیعی (NLP) چیست؟ - مهرداد توکلی – همیار آی تی ([لینک](https://www.hamyarit.com/blog/nlp/#:~:text=%D9%BE%D8%B1%D8%AF%D8%A7%D8%B2%D8%B4%20%D8%B2%D8%A8%D8%A7%D9%86%20%D8%B7%D8%A8%DB%8C%D8%B9%DB%8C%20(Natural%20Language,%D8%B2%D8%A8%D8%A7%D9%86%20%D8%A7%D9%86%D8%B3%D8%A7%D9%86%DB%8C%20%D8%B1%D8%A7%20%D8%AF%D8%B1%DA%A9%20%DA%A9%D9%86%D8%AF.))
2. معرفی بهترین کتابخانه‌های پردازش متن - بخش اول – textmining – ویرگول ([لینک](https://blog.text-mining.ir/nlp-lib1-nltk-spacy-corenlp-jrgpcjfyawfx))
3. پردازش زبان طبیعی چیست؟ - نویسنده ناشناس – هوشینو ([لینک](https://hooshio.com/%D9%BE%D8%B1%D8%AF%D8%A7%D8%B2%D8%B4-%D8%B2%D8%A8%D8%A7%D9%86-%D8%B7%D8%A8%DB%8C%D8%B9%DB%8C-%DA%86%DB%8C%D8%B3%D8%AA/))
4. هضم ([لینک](https://www.roshan-ai.ir/hazm/))
5. فایل CSV چیست و چه کاربردی دارد؟ - نویسنده ناشناس – فرادرس ([لینک](https://blog.faradars.org/csv-%DA%86%DB%8C%D8%B3%D8%AA%D8%9F/))