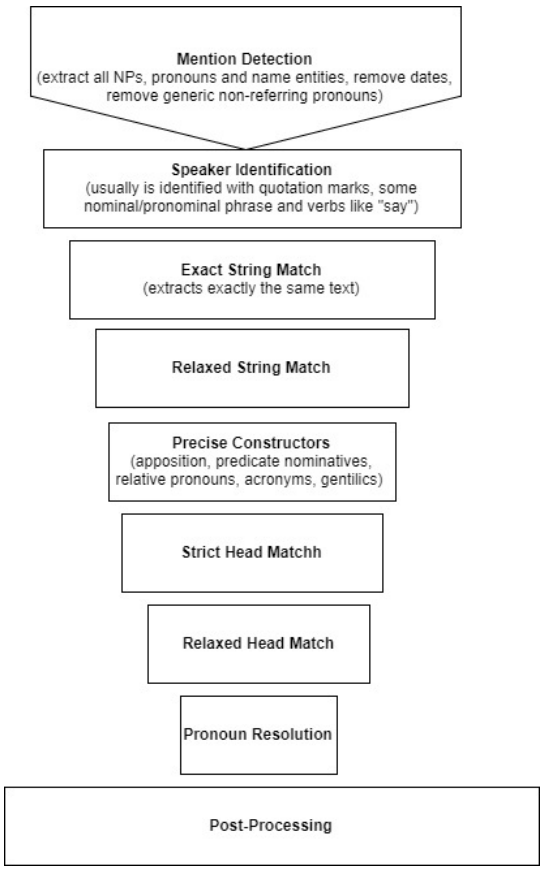
تمرین شماره ۱

۱.

اولین موضوع بررسی‌شده پیدا کردن مرجع اسمی[1]است در متن است. پیداکردن مرجع اسمی در واقع خوشه‌بندی همه گروه‌های اسمی (مرجع‌ها) است که به یک واحد بازمی‌گردند. این جملات را در نظر بگیرید:

[حسن]R1 پشت ستون ایستاده بود. فقط یک [[چشم]R2ش]R3 پیدا بود. [شازده]R4 گفت: «خب؟» [مراد]R5 دست کشید به هردو [[پای]R6ش]R7 و [[سیگار]R8ش]R9 را پیچید و گفت: «[شازده]R10 جون، خبر داری که [حاج تقی]R11 [[عمر]R12ش]R13 را داد به [شما]R14؟»[2]

برای مثال، در این متن کوتاه، سه خوشه‌ی {R1, R3}، {R5, R7, R9} و {R11, R13} به یک واحد بازمی‌گردند.



در پیداکردن مرجع ضمیر، هدف نهایی ساختن مدلی است که بتواند گروه‌های اسمی هم‌مرجع را دسته‌بندی کند. چندین مدل برای حل این مساله پیشنهاد شده است. منینگ، ژورافسکی و سوردیانو (2010) مدلی قاعده-بنیاد طراحی کرده‌اند که با استفاده از غربال‌های مختلف، مراجع مختلف متن را پیدا و دسته‌بندی می‌کند. آن‌ها برای آموزش از پیکره-ACE2004-ROTH DEVکه حاوی ۶۸ متن و ۴۵۳۶ مرجع است، استفاده کردند و برای آزمون نیز از سه پیکره دیگر استفاده کردند و معیار ارزیابی خود را نیز سه معیار B^3، MUC و F1 در نظر گرفتند. این مدل قاعده-بنیاد را به طور خلاصه می‌توان چنین توصیف کرد: برای پیدا کردن مرجع m_i، یک لیست از تمام گروه‌های هم‌مرجع خواهیم داشت: $m_1, m_2, \ldots m_{i-1}$. با استفاده از تعدادی قاعده، می‌توان گزینه‌های مختلف را بررسی کرد و بهترین مرجع را بین آن‌ها برای m_i انتخاب کرد.

این مدل حاوی چند غربال است که به ترتیب روی متن اجرا می شوند، و با اعمال هر غربال روی متن، مراجع خوشه بندی می‌شوند. هم‌چنین با اعمال هر غربال، فراخوانی[3]افزایش می‌یابد و دقت درستی[4]کاهش پیدا می‌کند. این مدل در عین سادگی، هم‌تراز آخرین و پیشرفته ترین مدل‌های پیشنهادشده عمل می‌کند. یک مزیت قابل‌توجه آن، ماژولار بودن آن است. در شکل رو به رو یک شِمای کلی از غربال‌های مختلف این مدل را مشاهده می‌کنید.

---

1. Co-reference resolution
2. متن از کتاب شازده احتجاب، نوشته هوشنگ گلشیری، صفحه ۱۸ برداشته شده است.
3. Recall
4. Precision

موضوع دوم، که موضوع منتخب برای پروژه این درس است، استفاده از الگوریتم word2vec و پیاده‌سازی آن روی پیکره‌ای از کلمات فارسی محاوره و فارسی رسمی است. الگوریتم word2vec از الگوریتم‌های شبکه عصبی مصنوعی است که در چند سال گذشته با اقبال گسترده‌ای مواجه شده است و لغات را به صورت بردار بازنمایی می‌کند. این الگوریتم برای نمایش کلمات به صورت بردار از دو شیوه skipgram وCBOW استفاده می‌کند. CBOW با استفاده از یک بافت معین‌شده یک کلمه را پیش‌بینی می‌کند؛ به عبارت دیگر، با داشتن w_{i-2}، w_{i-1}، w_{i+1}، w_{i+2} (بافت) w_i (واژه) را حدس می‌زند. Skipgram برعکس این شیوه عمل می‌کند؛ به عبارت دیگر، در روش skipgram با داشتن یک واژه (w_i) بافت آن واژه (w_{i-2}، w_{i-1}، w_{i+1}، w_{i+2}) تخمین زده می‌شود. در این باره تنها یک مقاله به اجمال بررسی شده است و در نتیجه موضوع هنوز نیازمند مداقه و مطالعه بیشتر است.

منابع:

Christopher Manning, Dan Jurafsky, Mihai Surdeanu. (2010). A Multi-Pass Sieve for Coreference Resolution. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 492-501.

Surdeanu, M. (2013, April 19). Co-Reference Resolution Revisited. University of Arizona, The United States. Retrieved from https://www.youtube.com/watch?v=xYfkPVbrae0

Hinton, G. (2017). *Neural Networks for Machine Learning*. Retrieved from coursera.com: https://www.coursera.org/learn/neural-networks/home/welcome

Rong, X. (2014). word2vec Parameter Learning Explained. *arXiv.org*.

۲.

الف)

اولین بحران در سنتز گفتار در زبان فارسی تولید تلفظ درست کلمات است. با توجه به این که اعراب‌گذاری در فارسی مکتوب وجود ندارد،کلماتی که اعراب‌گذاری نشده‌اند، مثل واژه «سر» می‌توانند به شیوه‌های مختلف (سِر، سَر و سِّر) خوانده شوند. به طور کلی در مواردی که تلفظ واژه و صورت نوشتاری آن متفاوتند، تولید تلفظ صحیح می‌تواند چالش برانگیز باشد؛ برای مثال، در «می‌خوابد» و «می‌خواهد» نیز این دوگانگی مشاهده می‌شود.

مساله دیگری که در سنتز گفتار زبان فارسی وجود دارد، تشخیص درست کسره اضافه است. کسره اضافه، مانند اعراب، عموما مکتوب نمی‌شود و این امر می‌تواند به دو خوانش مختلف از یک عبارت منجر شود؛ برای مثال، در دو عبارت «در کلاس نشستم» و «در کلاس را بست»، در صورتی که تشخیص داده نشود کسره اضافه کجا باید قرار بگیرد، تلفظ نادرست تولید می‌شود.

مساله چالش برانگیز دیگر (که البته تماما منحصر به زبان فارسی نیست) در نظر گرفتن علائم سجاوندی و تولید تلفظ صحیح با توجه به آن‌هاست. همه آن مثال معروف را شنیده‌ایم: «بخشش لازم نیست، اعدامش کنید» و «بخشش، لازم نیست اعدامش کنید». آن چه این دو جمله را از هم متمایز می‌کند، محل یک ویرگول است. در سنتز گفتار، در صورتی که نتوان سازوکاری یافت که محل‌های متفاوت این ویرگول به تلفظ‌های متفاوت بینجامند، به مشکل برخواهیم خورد.

ب)

از چالش‌های بازشناسی گفتار در زبان فارسی، همان طور که در بخش الف نیز اشاره شد، می‌توان تمام مواردی را نام برد که صورت نوشتاری و تلفظ یک واژه/عبارت/جمله یکسان نباشند؛ مانند «می‌خواهم» و «می‌خوابد». هم‌چنین، برای مثال در خواندن اعداد «و» به صورت ـً خوانده می شود. یا این که در ارقام تلفنی، دو عدد «دو» و «نه» به علت فرآیندهای واجی محل ابهام هستند و تشخیص آن ها مستلزم چیزی بیش از سیگنال ورودی است. هم چنین، کلمات هم‌آوا نیز می وانند محل ابهام قرار گیرند؛ برای مثال تشخیص «صد» از «سد»، «حیات» از «حیاط»، «ارز» از «عرض» یا «ارض».

ج)

یکی از چالش‌های ترجمه ماشینی چندمعنایی و برطرف کردن ابهام است. برای مثال «شیر»، «آل» و «روان» از جمله کلماتی هستند که چند معنا دارند و تشخیص این که در جمله یا عبارت فعلی کدام معنا منظور شده است، مستلزم بررسی بافت و شیوه همنشینی کلمات است. این فرآیند ابهام زدایی واژگانی (WSD) نام دارد. چندمعنایی می تواند نحوی نیز باشد؛ برای مثال در جمله «من تو را بیشتر از او دوست دارم» دو معنای متفاوت وجود دارد که برای ترجمه باید برطرف شود.

چالش دیگر در ترجمه ماشینی ترتیب واژه هاست؛ برای مثال در انگلیسی و فرانسه ترتیب SVO مشاهده می‌شود در حالی که ترتیب واژه‌ها در فارسی SOV است.

هم‌چنین، برخی واژه‌ها معادل دقیقی در زبان‌های دیگر ندارند و برای ترجمه آن‌ها ناچار به ساخت عبارت یا جمله می‌شویم.

ضرب المثل‌ها و اصطلاحات (که همگی باید یک واحد در نظر گرفته شوند) می‌توانند در ترجمه ماشینی نیز چالش‌برانگیز باشند.

عبارات کنایی و طعن‌آمیز نیز می‌توانند مشکل‌زا باشند.

۳.

الف)

برای حل قسمت الف، ابتدا یک تابع نرمال‌ساز نوشتیم که متون ورودی تابع را پیش از هر چیز نرمال کند و سپس با در نظر گرفتن فاصله به عنوان مرز بین کلمات، تعداد کلمات را محاسبه کند. با توجه به هدف نهایی ما در این تمرین، تابع نرمال‌ساز طوری نوشته شد که مرز کلمات را تا حد ممکن از هم مشخص کند؛ به همین دلیل، مواردی چون صورت‌های نوشتاری مختلف «ک» یا «ی» که نقشی در تعیین مرز کلمات نداشتند، نرمال نشدند (به استثنای نیم فاصله).

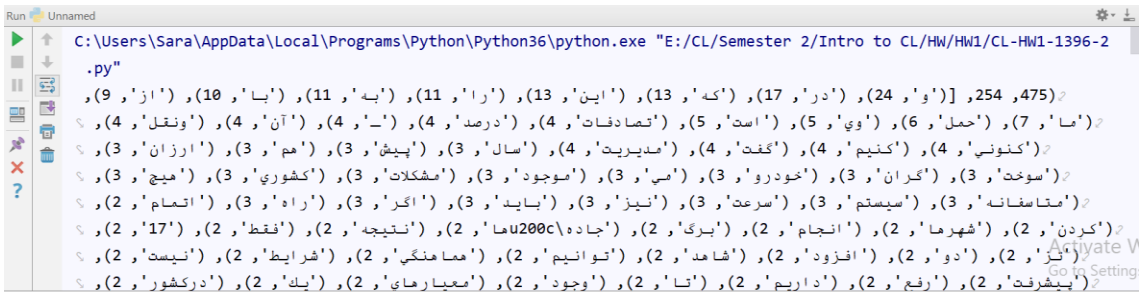
ابتدا تمام حروف لاتین از متن حذف شدند. از آن جا که سوال چیزی در مورد اعداد نگفته است، ما اعداد را حذف نکردیم و تنها با استفاده از عبارت‌های قاعده مند در دو طرفشان فاصله قرار دادیم که هر عدد یک واژه شمرده شود. برای پاک‌سازی علائم نگارشی، یک راه این بود که لیستی از تمام علائم نگارشی موجود تهیه کنیم و با یک عبارت قاعده مند آن‌ها را با فاصله جایگزین کنیم. با این حال، چون در زمان نوشتن تمرین، موفق به پیدا کردن یک لیست تام و تمام نشدیم، برای این که احتمال خطا را پایین بیاوریم، از ماژول استرینگ پایتون استفاده کردیم. به عبارت دیگر، پاک‌سازی علائم نگارشی برای ما دو مرحله داشت: با استفاده از

ماژول استرینگ تمام علائم نگارشی متن پاکسازی می‌شد. در این مرحله، هنوز علائم نگارشی ای که مختص زبان فارسی بودند و در ماژول استرینگ وجود نداشتند، در متن باقی می‌ماندند. در مرحله دوم، علائم نگارشی ویژه زبان فارسی از فایل حذف شدند. در نهایت، فاصله‌های بیشتر از یک عدد همگی با تنها یک فاصله جایگزین شدند.

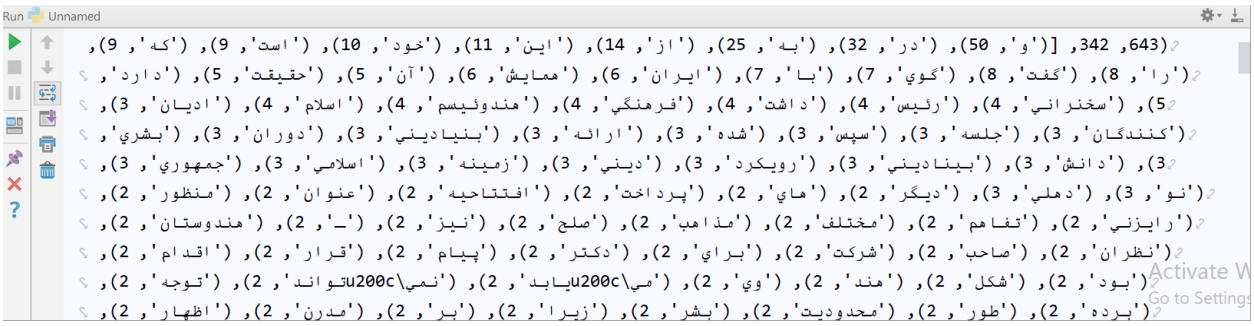
تابع CountWordsFile که ورودی آن مسیر فایل مورد نظر است، پس از دریافت فایل متنی مورد نظر، تابع نرمال ساز را روی آن اعمال می کند (و بعد محتوای نرمال شده را روی فایل جدیدی می نویسد) و بعد با متد split(). کلمات را می شمارد. شمارش تعداد کلمات یکتا با یک دیکشنری انجام شد و در نهایت، دیکشنری موردنظر از مقادیر زیاد به مقادیر کم sort شد.

```
def CountWordsFile(TextFilename):

    ''' Returns the number of words in a text.
    Input : name of the file (str) It's the address.
    Output : number of all words, all unique words and the repetition
of each unique word (int)
    '''
    with open(TextFilename,"r", encoding="utf-8") as file:
        text = file.read()
    t = Normalizer(text)


    #Find the number of words.
    no_words = len(t.split(' '))
    #Find the frequency of each unique word.
    freq = {}
    for word in t.split(' '):
        freq[word] = freq.get(word, 0) + 1

    no_unique_words = len(freq)
    #Sort the frequencies in a descending order.
    sorted_freq = sorted(freq.items(), key=operator.itemgetter(1))
    sorted_freq.reverse()
    return (no_words, no_unique_words, sorted_freq)
```

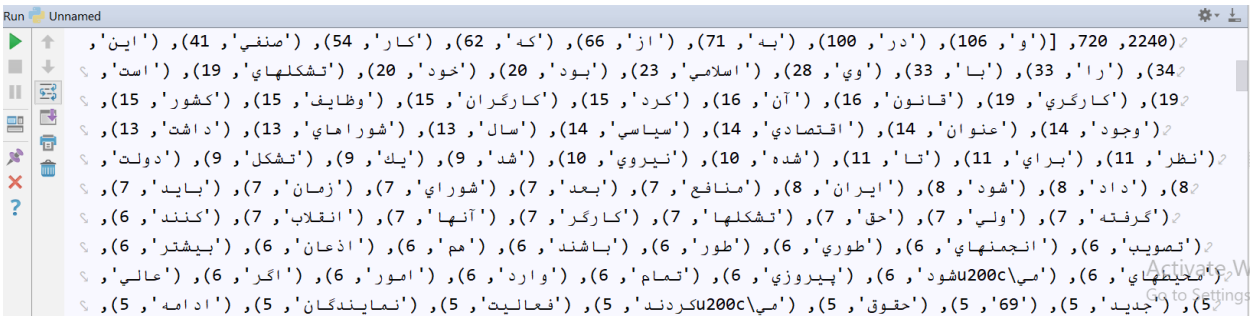
اولین فایل فولدر اجتماعی حاوی ۴۷۵ واژه و ۲۵۴ واژه یکتا بود و پربسامدترین واژه آن «و» با ۲۴ بار تکرار بود.



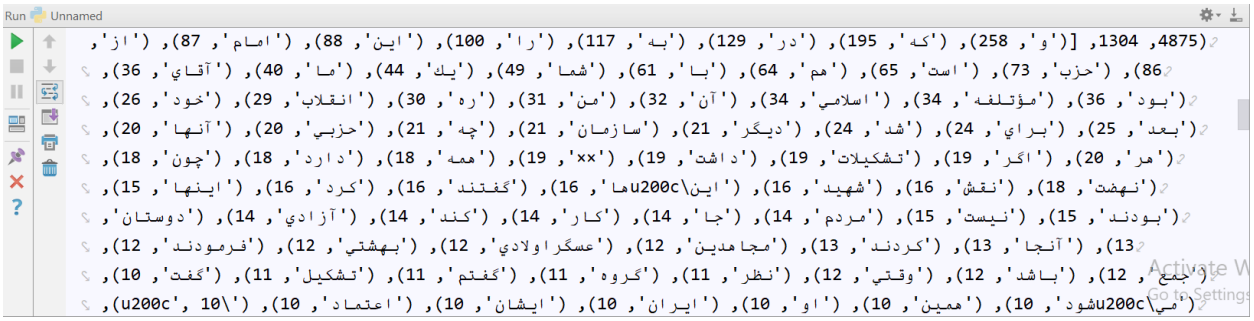
اولین فایل فولدر ادیان حاوی۶۴۳ واژه و ۳۴۲ واژه یکتا بود و پربسامدترین واژه آن «و» با ۵۰ بار تکرار است.



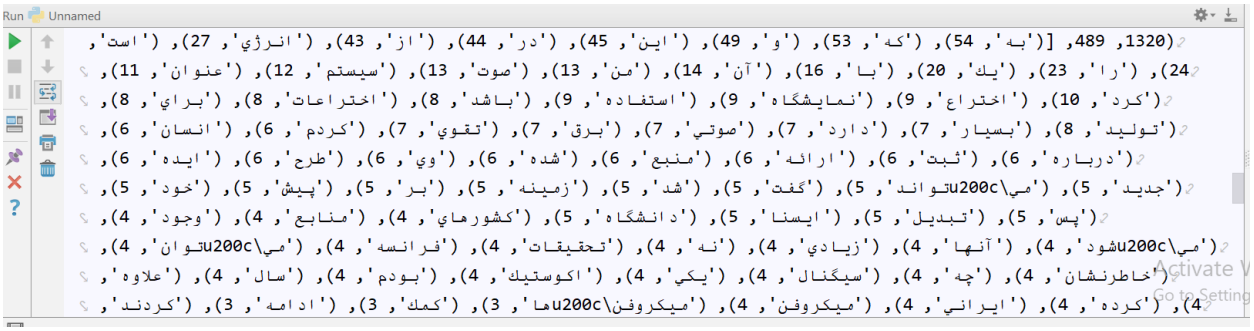
اولین فایل فولدر اقتصادی حاوی ۲۲۴۰ واژه و ۷۲۰ واژه یکتاست و پربسامدترین واژه آن «و» با ۱۰۶ بار تکرار است.



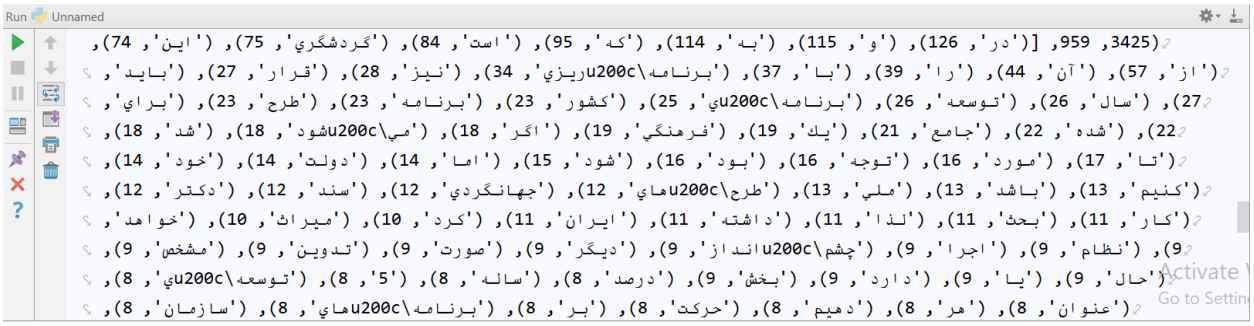
اولین فایل فولدر سیاسی حاوی ۴۸۷۵ واژه و ۱۳۰۴ واژه یکتاست و پربسامدترین واژه آن «و» با ۲۵۸ بار تکرار است.



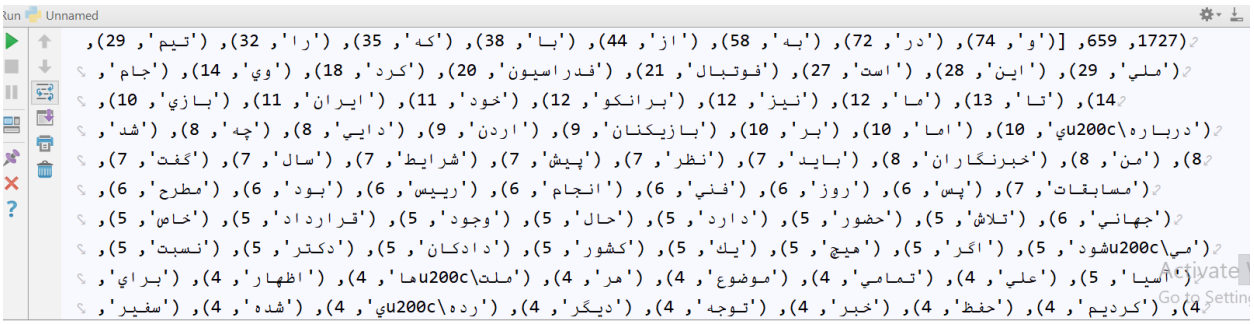
اولین فایل فولدر فناوری حاوی ۱۳۲۰ واژه و ۴۸۹ واژه یکتاست و پربسامدترین واژه آن «به» با ۵۴ بار تکرار است.



اولین فایل فولدر مسائل راهبردی ایران حاوی ۳۴۲۵ واژه و ۹۵۹ واژه یکتاست و پربسامدترین واژه آن «در» با ۱۲۶ بار تکرار است.



و در نهایت، اولین فایل فولدر ورزشی حاوی ۱۷۲۷ واژه و ۶۵۹ واژه یکتاست و پربسامدترین واژه آن «و» با ۷۴ بار تکرار است.



ب)

تابع CountWordsFolder مسیر فولدر زبرا (که خود حاوی چند فولدر دیگر است) را می‌گیرد و تعداد کل کلمات هر پوشه، تعداد کلمات یکتای هر پوشه، تعداد کل کلمات پیکره زبرا و تعداد کلمات یکتای پیکره زبرا را محاسبه می‌کند.

برای نوشتن این تابع، ابتدا تابعی به نام FindFiles نوشتیم که تمام فایل های متنی یک پوشه را برمی گرداند. در تابع CountWordsFolder پس از گرفتن آدرس پوشه زبرا، تمام پوشه‌های درون این پوشه را با استفاده از متد listdir() به دست می آوریم و سپس مسیر ورودی به هر پوشه را با استفاده از ماژول os می‌سازیم. سپس در هر پوشه، تمام فایل‌های متنی را با استفاده از تابع FindFiles استخراج می‌کنیم و در نهایت تعداد واژگان و واژگان یکتای هر پوشه را محاسبه کرده و ذخیره می‌کنیم و در نهایت تعداد واژگان و واژگان یکتا را برای کل پیکره به دست می‌آوریم.

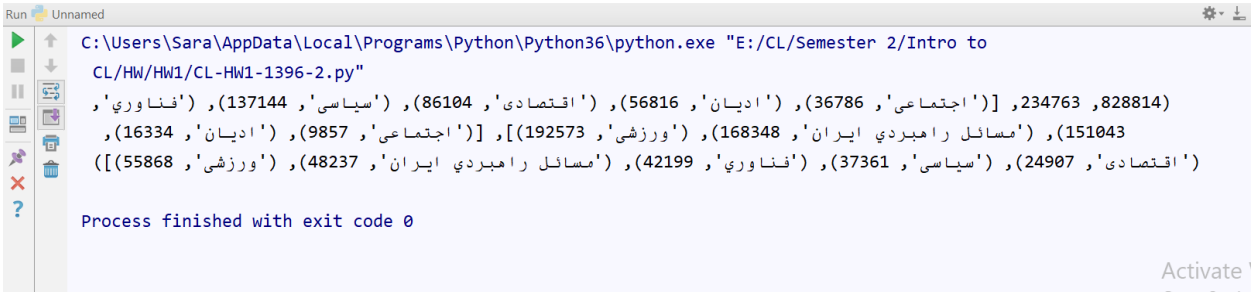
```
def CountWordsFolder(Path):
    '''Returns the result of CountWordsFile over a folder path, for
each folder and for the Zebra corpus.
    Input : path (str)
    Output: Number of all words and unique words for the given folder
and for Zebra.
    '''
    p = []
    Folders = os.listdir(Path)
    SumAll, SumUnique = 0, 0
```

```
    FolderAll, FolderUnique = [], []
    #Obtain the path for each subfolder and put it in a list.
    for Folder in Folders:
        p.append(os.path.join(Path + "\\" + str(Folder)))
    #For each subfolder, find all the .txt files, get the result and
put it in a list.
    for i in range(len(p)):
        f = FindFiles(p[i])
        for j in range(len(f)):
            SumAll += CountWordsFile(f[j])[0]
            SumUnique += CountWordsFile(f[j])[1]
        FolderAll.append((Folders[i], SumAll))
        FolderUnique.append((Folders[i], SumUnique))
    #Calculate the result for the whole corpus.
    ZebraAll = sum([x[1] for x in FolderAll])
    ZebraUnique = sum([x[1] for x in FolderUnique])
    return ZebraAll, ZebraUnique, FolderAll, FolderUnique
print(CountWordsFolder("E:\CL\Semester 2\Intro to CL\HW\HW1\Zebra"))
```

همان‌طور که در تصویر زیر مشاهده می‌کنیم، تعداد کل واژه‌های پوشه اجتماعی ۳۶۷۸۶، ادیان ۵۶۸۱۶، اقتصادی ۸۶۱۰۴، سیاسی ۱۳۷۱۴۴، فناوری ۱۵۱۰۴۳، مسائل راهبردی ایران ۱۶۸۳۴۸ و ورزشی ۱۹۲۵۷۳ واژه است.

تعداد واژگان یکتای پوشه اجتماعی ۹۸۵۷، ادیان ۱۶۳۳۴، اقتصادی ۲۴۹۰۷، سیاسی ۳۷۳۶۱، فناوری ۴۲۱۹۹، مسائل راهبردی ایران ۴۸۲۳۷ و ورزشی ۵۵۸۶۸ واژه است.

تعداد کل واژگان پیکره زبرا ۸۲۸۸۱۴ واژه و تعداد واژگان یکتای آن ۲۳۴۷۶۳ واژه است.



ج)

برای نوشتن این تابع، ابتدا با استفاده از تابع FindFiles مسیر رسیدن به تمامی فایل‌های پوشه ورودی را استخراج کردیم. سپس محتویات هر فایل را خوانده، \n را حذف کرده، متن را نرمال کرده و سپس متن نهایی را در یک لیست قرار دادیم. بدین ترتیب، محتوای نهایی هر فایل به این لیست پیوست می‌شود. به عبارت دیگر، هر آرایه لیست نشانگر محتوای یک فایل درون پوشه است. حال برای خروجی نهایی (outfile) آرایه‌های این لیست را در متغیر outfile می‌ریزیم و پس از هر آرایه \n اضافه می‌کنیم که محتویات فایل‌های متفاوت جدا از هم قرار گیرند. در انتها، خروجی نهایی را روی یک فایل متنی می‌نویسیم.

فایل‌های خروجی این تابع به فایل زیپ پیوست شده‌اند.

```
def CombineFiles(Path):
    '''Returns the content of files in a folder each in one line. Each
line represents the conent of one .txt file in
    the folder.
    Input: path (str)
    Output: text(str)
    '''
    outfile = ""
    Paths = FindFiles(Path)
    P = []
    #Delete \n for the content of each file, normalize it and append
it to a list.
    for path in Paths:
        with open(path, "r", encoding="utf-8") as h:
            t = Normalizer(h.read().replace("\n", ""))
        P.append(t)
    #Put all the files together in "outfile" and add "\n" after each
file.
    for file in range(len(P)):
        outfile = outfile + P[file] + "\n"

    #Write the result to a new file.
    with open("Merged.txt", "w", encoding="utf-8") as g:
        g.write(outfile)
    return outfile
```