



تمرین اول درس پردازش زبان طبیعی

نام تهیه کننده: ملیکا نوبختیان

نسخه: ۱

۱- بخش تئوری

۱-۱- سوال اول

۱-۱-۱ کتابخانه NLTK

کتابخانه NLTK یا natural language toolkit یک کتابخانه ضروری برای کاربردهای پردازش زبان طبیعی است که کارهایی مانند tokenization و classification, stemming, tagging, parsing, semantic reasoning را به زبان python انجام می‌دهد. NLTK را می‌توان ابزار اصلی در کاربردهای NLP و ML دانست. هر چند این کتابخانه کاربردهای زیادی دارد اما باید قبول کرد استفاده از آن برای پردازش زبان طبیعی دشوار است. اگر از این کتابخانه به درستی استفاده نشود بسیار کند خواهد بود و در کاربردهای production جایی نخواهد داشت. پس باید به روش استفاده از آن دقت شود.

۱-۱-۲ کتابخانه Gensim

Gensim یک کتابخانه به زبان python است که به طور ویژه به مشخص کردن semantic similarity بین دو سند از طریق ابزارهای topic modeling و vector space modeling می‌پردازد. این کتابخانه می‌تواند پیکره‌های متنی بزرگ را با کمک بهینگی data streaming و الگوریتم‌های incremental مدیریت کند. ویژگی بسیار مهم و برتر این کتابخانه که نسبت به بقیه نمونه‌ها بی نظیر است memory usage optimization و processing speed آن است.

۱-۱-۳ کتابخانه spaCy

این کتابخانه نسبت به بقیه کتابخانه‌ها جدیدتر است و برای کاربردهای production طراحی شده است. این کتابخانه سریع‌ترین syntactic parser که در حال حاضر موجود است را ارائه می‌دهد. به علاوه چون این کتابخانه به زبان Cython نوشته شده است بسیار پرسرعت و بهینه است. از معایب این کتابخانه این است که برخلاف کتابخانه‌های دیگر تعداد کمتری از زبان‌ها را پشتیبانی می‌کند.

۱-۲- سوال دوم

الف) $(09[0-9]\{9,14\}|00989[0-9]\{9,11\}|\backslash+989[0-9]\{9,13\})$

نکته: شماره عادی را ۱۳ شماره در نظر گرفتیم و براساس آن حداقل را مشخص کردم

ب)

$\wedge(0[1-9][12][0-9]3[01])-(0[1-9]1[012])-(19|20)\backslash d\$$

ج) $((http)\backslash:\backslash\backslash)?[a-zA-Z0-9\backslash.\backslash?@:\backslash_=#]+\backslash.(ir|org)$

(د) $([0-9]\{3\})[A-Z]([0-9]\{3\})IR([0-9]\{2\})$

۳-۱- سوال سوم

در زبان‌هایی مانند زبان چینی که بین کلمات فاصله وجود ندارد، tokenize کردن متون و پیدا کردن کلمات متن دشوار خواهد بود زیرا بدون فاصله عملاً هیچ حرفی را از بقیه نمی‌توان جدا دانست. اگر زبان‌هایی مانند انگلیسی نیز این فاصله را حذف کنیم باز هم تشخیص کلمات دشوار خواهد بود. به این مثال دقت کنید:

This is insane. This is insane. درست است که شاید مغز ما بتواند جمله This is insane را تشخیص دهد، اما کامپیوترها چگونه می‌توانند این موضوع را حل کنند؟ در اینجا word segmentation با استفاده از الگوریتم Maximum Matching به کمک ما می‌آید و این مشکل را حل می‌کند. این الگوریتم در قدم‌های زیر کار را انجام می‌دهد:

- ۱) با اولین کاراکتر رشته داده‌شده کار را آغاز می‌کند.
 - ۲) دنبال بزرگ‌ترین کلمه موجود در مرجع کلمات که در دست داریم می‌گردد که با این کاراکتر آغاز شده باشد.
 - ۳) اگر کلمه مشابهی پیدا شد این کلمه به لیست کلمات اضافه می‌شود و مرز شروع دوباره تعیین می‌شود ولی اگر کلمه‌ای پیدا نشد با همان کاراکتر به عنوان یک کلمه رفتار می‌شود.
 - ۴) دوباره به مرحله ۲ برمی‌گردیم با این تفاوت که از مرزی که تعیین شده است شروع می‌کنیم.
- با مثال This is insane پیش می‌رویم. از T شروع می‌کنیم. اگر پیش برویم بزرگ‌ترین و تنها کلمه‌ای که با t شروع می‌شود و در مرجع کلمات وجود دارد this خواهد بود. حالا از i به کار خود ادامه می‌دهیم کلمه بعدی که پیدا خواهیم کرد is خواهد بود. باز هم به i می‌رسیم. اینجا ما به دو کلمه با معنی می‌خوریم in و insane. اما چون insane طولانی‌تر است به عنوان کلمه موجود انتخاب خواهد شد. پس کلمات این جمله شامل this, is و insane خواهد بود.

۲- بخش عملی

۲-۱- سوال اول

برای این بخش و تشخیص اسامی سه الگو کلی داشتیم: اسامی که با Dr. شروع می‌شوند، اسامی که با Doctor شروع می‌شوند و اسامی که با M. D. پایان می‌یابند. برای چک کردن این سه الگو به صورت زیر regex های زیر را در نظر گرفتیم:

```
def CheckPattern(text):
    ## patterns
    if re.search("Doctor(((\s)([A-Z](\s.))+)|((\s)[A-Z][a-z]*-[A-Z][a-z]*)|((\s)[A-Z][a-z]*))", text):
        return True
    elif re.search("Dr(\s)((\s)([A-Z](\s.))+)|((\s)[A-Z][a-z]*-[A-Z][a-z]*)|((\s)[A-Z][a-z]*))", text):
        return True
    elif re.search(
        "(((\s)([A-Z](\s.))+)|((\s)[A-Z][a-z]*-[A-Z][a-z]*)|((\s)[A-Z][a-z]*))(((\s)([A-Z](\s.))+)|((\s)[A-Z][a-z]*-[A-Z][a-z]*)|((\s)[A-Z][a-z]*))+.? H\d\d",
        text):
        return True
    else:
        return False

file_address = input("Enter File Address: ")
file1 = open(file_address, 'r')
Lines = file1.readlines()
for line in Lines:
    print(f"Text: {line.strip()}")
    result = CheckPattern(line)
    if result:
        print("Matched Pattern Successfully! True")
    else:
        print("Wrong Text! False")
print("")
print("////////////////////")
print("")
```

برای اینکه مواردی که در تمرین گفته شده بود را چک کنم، نمونه‌ها را در یک فایل txt ریختم و برای خواندن نمونه‌ها کافی است آدرس فایل مورد نظر را به آن بدهیم که برای من این فایل doctor_names.txt خواهد بود. خروجی به این صورت خواهد بود که هر الگو چاپ خواهد شد و سپس مشخص می‌شود آیا این موارد مطابق regex ها هستند یا نه. قطعه کد زیر این کار را انجام خواهد داد:

خروجی به صورت زیر است:

۲-۲- سوال دوم

برای اینکه نتیجه متدهای word_tokenize و sent_tokenize را ببینیم من از یک فایل با متن نمونه به نام comp.txt استفاده کردم. البته به طور کلی می‌توان آدرس فایل دلخواه را داد تا این کار را روی آن انجام دهد.

```
print("**** Use NLTK to See Result of word tokenize abd sent tokenize on given file ****")
print("")
file_address = input("Enter File Address: ")
file1 = open(file_address, 'r')
print("Main File Text:")
text = str(file1.read())
print(text)
print("")

Text: Will
words = word_tokenize(text)
Matched Pattern Successfully! True
print("***** Tokenized words with word_tokenize *****")
for word in words:
    print(word)
print("")
print("***** Sentences with sent_tokenize *****")
Text: Pam
sents = sent_tokenize(text)
Matched Pattern Successfully! True
for sent in sents:
    print(sent)
print("")

Text: Leighton E. Cluff M.D.
Matched Pattern Successfully! True

Text: Dr. Etienne-Emile Baulieu
Matched Pattern Successfully! True

Text: Doctor Dolittle
Matched Pattern Successfully! True

Text: James S. Thompson, M.D.
Matched Pattern Successfully! True

Text: Dr. Karl Thomae
Matched Pattern Successfully! True

Text: Doctor William Archibald Spooner
Matched Pattern Successfully! True

Text: C.M. Franklin, M.D.
Matched Pattern Successfully! True

Text: Dr. Alan D. Lourie
Matched Pattern Successfully! True

Text: Atul Gawande, M.D.
Matched Pattern Successfully! True

Text: Dr. Xiaotong Fei
Matched Pattern Successfully! True
```

قطعه کد زیر فایل مورد نظر را می‌خواند و کلمات و جملات آن را با استفاده از متدهای گفته شده به دست می‌آورد و چاپ می‌کند:

خروجی برای فایل گفته شده به صورت زیر خواهد بود. ابتدا بخشی از کلماتی که در این فایل موجود است را مشاهده می کنید:

و جملات آن نیز به صورت زیر هستند:

۳-۲- سوال سوم

برای این سوال می توانیم از back reference ها در regex استفاده کنیم. با آن ها می توانیم یک گروه در regex را هر جا که بخواهیم صدا بزنیم. این گروه با شکل 1, /2, ... نشان داده می شود. برای حذف حروف تکراری از regex ای که در زیر می بینید استفاده شده است. 2(\w) نشان می دهد یک کاراکتر آمده و همان کاراکتر که گروه ۲ است دوباره تکرار شده است. برای حذف یک حرف تکراری در هر مرحله از 1\2\3 استفاده شده است. در اینجا می بینیم که یک گروه ۲ تکراری حذف شده است. این کار را تا زمانی انجام می دهیم که به کلمه ای برسیم

```

**** Tokenized words with word_tokenize ****
Throughout

from nltk.corpus import words

****
word = input("Word should be normalized:")
Through
entitit
char_repeated = re.compile(r'(\w*)(\w)\2(\w*)')
///
These
///
while not word in words.words():
First,
    normal_word = char_repeated.sub(r'\1\2\3', word)
    if normal_word == word:
        break
    word = normal_word
The ge
///
They c
these
uses 'print(f'normal word: {word}')
```

که در words در nltk موجود باشد:

یک نمونه و جواب را در زیر می بینید:

۴-۲- سوال چهارم

```

Word should be normalized: loooooove
normal word: love
```

ابتدا فایل‌های موردنظر را خواندم و متن آن‌ها را در متغیرهای متناظرشان ذخیره کردم:

```
english_example_Address = "AlbertEinstein.txt"
file1 = open(english_example_Address, 'r')
english_example = str(file1.read())

persian_example_Address = "Shahnameh.txt"
file1 = open(persian_example_Address, 'r')
persian_example = str(file1.read())

english_short_text_Address = "ShortSampleEnglish.txt"
file1 = open(english_short_text_Address, 'r')
english_short = str(file1.read())

persian_short_text_Address = "ShortSamplePersian.txt"
file1 = open(persian_short_text_Address, 'r')
persian_short = str(file1.read())
```

سپس از TreebankWordTokenizer استفاده کردم تا token های هر متن و type آن‌ها را به دست آورم. منظور از type در اینجا توکن‌های منحصر به فردی است که پس از عمل tokenize به وجود می‌آیند. قطعه کد زیر نحوه کار با این tokenizer است:

```
tt = TreebankWordTokenizer().tokenize(english_example)
print(f'** Token count in AlbertEinstein.txt with Treebankwordtokenizer : {len(tt)}')
print(f'** Token types in AlbertEinstein.txt with Treebankwordtokenizer : {"", ".join(list(set(tt)))}')
print("")

tt = TreebankWordTokenizer().tokenize(persian_example)
print(f'** Token count in Shahnameh.txt with Treebankwordtokenizer : {len(tt)}')
print(f'** Token types in Shahnameh.txt with Treebankwordtokenizer : {"", ".join(list(set(tt)))}')
print("")

tt = TreebankWordTokenizer().tokenize(english_short)
print(f'** Token count in ShortSampleEnglish.txt with Treebankwordtokenizer : {len(tt)}')
print(f'** Token types in ShortSampleEnglish.txt with Treebankwordtokenizer : {"", ".join(list(set(tt)))}')
print("")

tt = TreebankWordTokenizer().tokenize(persian_short)
print(f'** Token count in ShortSamplePersian.txt with Treebankwordtokenizer : {len(tt)}')
print(f'** Token types in ShortSamplePersian.txt with Treebankwordtokenizer : {"", ".join(list(set(tt)))}')
print("")
```

برای متن AlbertEinstein.txt خروجی به صورت زیر است. در اینجا تنها بخشی از token ها قابل مشاهده هستند:

```
** Token count in AlbertEinstein.txt with Treebankwordtokenizer : 250
** Token types in AlbertEinstein.txt with Treebankwordtokenizer : Berlin, the, a, Albert, time, Professor, Federal, Kaiser, in, Princeton*, technical, Berlin, position, his,
```

برای متن شاهنامه نیز خروجی به شکل زیر است:

```
** Token count in Shahnameh.txt with Treebankwordtokenizer : 2207
** Token types in Shahnameh.txt with Treebankwordtokenizer : و بهر یاقوت لرزان، بندهی، گونه، ماهست، خواند، نویسنده، جوید، پیششبه، ژرفی، هوش، خویشیگانه، شوند، برورد
```

برای متن کوتاه انگلیسی خروجی به شکل زیر است:
و برای متن کوتاه فارسی هم خروجی به شکل زیر خواهد بود:

```
Token count in ShortSampleEnglish.txt with Treebankwordtokenizer : 19
```

```
** Token count in ShortSamplePersian.txt with Treebankwordtokenizer : 16
```

```
** Token types in ShortSamplePersian.txt with Treebankwordtokenizer : نمونه : ۱, سلام, و, این, است, این, و, تمرین, و, ما, امیدوارم, متن, کوتاه, و, یک, باشد, خوب,
```

برای به دست آوردن کلمات متن کوتاه انگلیسی با استفاده از RegexpTokenizer از عبارت با قاعده به صورت زیر استفاده کردم که توکن‌های متن از جمله کلمات و اعداد را به ما خواهد داد :

```
regex = RegexpTokenizer('\w+').tokenize(english_short)
print(f'ShortSampleEnglish.txt tokens with Regexp : {"", ".join(regex)}')
```

و نتیجه آن به صورت زیر خواهد بود:

```
ShortSampleEnglish.txt tokens with Regexp : Hello, Hope, you, re, doing, well, It, is, a, sample, short, text, This, is, our, 1, st, assignment
```

اگر بخواهیم تنها کلمات چاپ شوند و اعداد در این قسمت نباشند باید از کد زیر استفاده کنیم:

```
regex = RegexpTokenizer('[a-zA-Z]+').tokenize(english_short)
print(f'ShortSampleEnglish.txt words with Regexp : {"", ".join(regex)}')
```

و نتیجه آن نیز به صورت زیر است:

```
ShortSampleEnglish.txt words with Regexp : Hello, Hope, you, re, doing, well, It, is, a, sample, short, text, This, is, our, st, assignment
```

برای اینکه بتوانیم اعداد متن انگلیسی را به دست آوریم به شکل زیر عمل می‌کنیم:

```
regex = RegexpTokenizer('[0-9]+').tokenize(english_example)
print(f'AlbertEinstein.txt numbers with Regexp : {"", ".join(regex)}')
```

و نتیجه به صورت زیر است:

```
AlbertEinstein.txt numbers with Regexp : 14, 1879, 1896, 1901, 1905, 1908, 1909, 1911, 1914, 1914, 1933, 1940, 1945
```

برای به دست آوردن توکن‌های متن کوتاه فارسی هم به همان شکلی که در ابتدا برای تشخیص توکن‌ها متن انگلیسی عمل کردیم پیش می‌رویم و خروجی آن نیز به شکل زیر است:

```
ShortSamplePersian.txt tokens with Regexp : نمونه : ۱, سلام, و, این, است, این, و, تمرین, و, ما, امیدوارم, متن, کوتاه, و, یک, باشد, خوب,
```

نتیجه استفاده از WhiteSpaceTokenizer برای متن کوتاه انگلیسی به شکل زیر است:

```
ShortSampleEnglish.txt words with WhiteSpaceTokenizer : Hello!, Hope, you're, doing, well., It, is, a, sample, short, text.This, is, our, 1, st, assignment.
```


این tokenizer به این شکل عمل می‌کند که می‌تواند token های یک متن را بدون هیچ فاصله، tab یا newline به دست آورد. به عبارت دیگر برای جداسازی token ها از این موارد استفاده می‌کند. اگر خواسته باشیم با regexp چنین کاری انجام دهیم می‌توانیم به شکل زیر عمل کنیم:

هنگامی که در regexp از gaps استفاده می‌کنیم، این tokenizer به این شکل عمل می‌کند که سعی می‌کند با

```
tokenizer = RegexpTokenizer('\s+', gaps=True).tokenize(english_short)
print(f'ShortSampleEnglish.txt words with RegexpTokenizer same as WhiteSpaceTokenizer : {" ".join(tokenizer)}')
```

regex داده شده token های متن را جدا کند. در اینجا سعی می‌کند با فاصله به هر تعداد این کار را انجام دهد.

```
ShortSampleEnglish.txt words with RegexpTokenizer Like WhiteSpaceTokenizer : Hello!, Hope, you're, doing, well., It, is, a, sample, short, text.This, is, our, 1, st, assignment.
```

نتیجه استفاده از آن به شکل زیر است:

واضح است که خروجی دو حالت برابر است.

نتیجه استفاده از WordPunctokenizer به شکل زیر است:

```
ShortSampleEnglish.txt words with WordPunctTokenizer : Hello, !, Hope, you, ', re, doing, well, ., It, is, a, sample, short, text, ., This, is, our, 1, st, assignment, .
```

این tokenizer به این شکل عمل می‌کند رشته‌های کاراکتری الفبایی و هم چنین غیر الفبایی را به عنوان token قبول می‌کند. رشته‌های الفبایی کلمات معمولی هستند که در متن استفاده می‌شود اما رشته غیر الفبایی شامل رشته یا کاراکترهایی می‌شود کاراکترهای دیگر غیر الفبا مانند ', ., ?, ! و ... را در خود دارند. با توجه به توکن‌های استخراج شده از متن کوتاه انگلیسی متوجه می‌شویم که ., ', ! به عنوان توکن شناخته شده‌اند در حالی که در tokenizer های قبل چنین حالتی نداشتیم و این موارد به قبل یا بعد خود متصل می‌شدند و به عنوان یک token شناخته می‌شدند.

۵-۲- سوال پنجم

اگر PorterStemmer و LancasterStemmer را روی توکن‌های گفته شده اعمال کنیم، نتیجه زیر را خواهیم

گرفت:

```
*** Original Words ***:
was, Germany, weeks, later, family, moved, Italy

** Stemmed with PorterStemmer **:
wa, germani, week, later, famili, move, itali

** Stemmed with LancasterStemmer **:
was, germany, week, lat, famy, mov, ita
```

همان طور که در تصویر واضح است نتیجه استفاده از این دو stemmer به جز در کلمه weeks متفاوت است. اما حالا به بررسی کلی نتایج این دو stemmer می‌پردازیم. اگر دقت کنیم می‌بینیم که بیشتر کلماتی که توسط Lancaster، stem شده‌اند بیشتر از حالت اولیه خود فاصله گرفته‌اند و به ریشه اصلی نزدیک شده‌اند. اما porter بیشتر شکل اصلی کلمات را حفظ کرده‌است. به عبارت دیگر می‌توان گفت Lancaster یک stemmer سختگیرتر نسبت به porter است. در حالی که نتیجه porter به چیزی که برای ما آشنا است نزدیک‌تر است. می‌توانیم این طور بگوییم که اگر فشرده‌سازی حداکثری برای کلماتمان می‌خواهیم Lancaster گزینه بهتری است در حالی که Porter نتیجه ملموس‌تر و نزدیک‌تر به کلمه اصلی را به ما خواهد داد. اگر از lemmatizer بر روی کلماتی که گفته شده‌است استفاده کنیم نتیجه زیر را خواهیم گرفت:

```
*** Lemmatize given Words ***:
wave, fishing, rock, wa, corpus, better, ate, broken
```

همان طور که مشخص است تنها waves, rocks و corpora به حالت اولیه و درست خود بازگشته‌اند و بقیه موارد این چنین نیستند. حتی was هم به اشتباه به wa تبدیل شده‌است که نشان می‌دهد lemmatizer به اشتباه was را به عنوان یک کلمه جمع در نظر گرفته است و s آن را حذف کرده‌است. اما راه حل چیست؟ برای حال این مشکل هنگام lemmatize کردن باید مشخص کنیم نوع هر کلمه چیست برای مثال اسم، فعل، صفت و ... است. در این صورت به نتیجه درست خواهیم رسید. Lemmatizer در حالت عادی همه کلمات اسم در نظر می‌گیرد:

```
print("*** Lemmatize with Type ***:")
print(f'word : fishing, lemm : {lemmatizer.lemmatize("fishing", pos="v")}')
print(f'word : was, lemm : {lemmatizer.lemmatize("was", pos="v")}')
print(f'word : better, lemm : {lemmatizer.lemmatize("better", pos="a")}')
print(f'word : ate, lemm : {lemmatizer.lemmatize("ate", pos="v")}')
print(f'word : broken, lemm : {lemmatizer.lemmatize("broken", pos="v")}')
```

و نتیجه صحیح lemmatize این کلمات به شکل زیر است:

```
*** Lemmatize with Type ***:
word : fishing, lemm : fish
word : was, lemm : be
word : better, lemm : good
word : ate, lemm : eat
word : broken, lemm : break
```

Fishing, was, ate و broken از نوع verb هستند و better از نوع adjective است.

۶-۲- سوال ششم

برای بررسی تغییرات به وجودآمده ۳ توییت در نظر گرفتیم که محتوای اصلی آنها به شکل زیر است:

Original Tweets:

```
["RT @Claire_FOX5: #BREAKING: @GaSecofState's office confirms Floyd
County has found 2,600 ballots during audit. Says Sec. Raffensperger
waâ€¦"]
/////
['Really disgusting that the failing New York Times allows dishonest
writers to totally fabricate stories.']
/////
['What an INCREDIBLE opening by @JesseBWatters. I will put it up later for
everyone to see!']
```

مرحله اول حذف فاصله‌های تکراری است. برای این کار از regex استفاده می‌کنیم و هر جا این الگو تکراری پیدا شد تنها با یک فاصله جایگزین می‌کنیم:

```
# Step 1:
# Remove Duplicate Spaces between tweet's words
for i, tweet in enumerate(tweets):
    new_tweet = re.sub(r'\s+', ' ', str(tweet))
    tweets[i] = new_tweet
```

نتیجه به صورت زیر است:

```
Step 1: Remove Duplicate Spaces between tweet's words
["RT @Claire_FOX5: #BREAKING: @GaSecofState's office confirms Floyd County
has found 2,600 ballots during audit. Says Sec. Raffensperger waâ€¦"]
/////
['Really disgusting that the failing New York Times allows dishonest
writers to totally fabricate stories.']
/////
['What an INCREDIBLE opening by @JesseBWatters. I will put it up later for
everyone to see!']
```

در توییت اول یک فاصله اضافی بین #BREAKING و @GaS... مشاهده می‌شد که حذف شده‌است. در مرحله دوم تمام حروف باید به حروف کوچک تبدیل شوند:

```
# Step 2:
# Make text's words Lower case
for i, tweet in enumerate(tweets):
    new_tweet = tweet.lower()
    tweets[i] = new_tweet
```

نتیجه اعمال این قسمت به شکل زیر است:

```
Step 2: Make text's words Lower case
["rt @claire_fox5: #breaking: @gasecofstate's office confirms floyd county
has found 2,600 ballots during audit. says sec. raffensperger waâ€¦"]
/////
['really disgusting that the failing new york times allows dishonest
writers to totally fabricate stories.']
/////
['what an incredible opening by @jessebwatters. i will put it up later for
everyone to see!']
```

در مرحله سوم username ها حذف می‌شوند. برای این قسمت باز هم از regex استفاده می‌کنیم و الگوهایی که با @ شروع می‌شوند را حذف می‌کنیم:

```
# Step 3:
# Remove Handles ( Usernames ) from Tweets
for i, tweet in enumerate(tweets):
    new_tweet = re.sub('@[\w]+', '', str(tweet))
    tweets[i] = new_tweet
```

نتیجه اعمال این قدم به شکل زیر است:

```
Step 3: Remove Handles ( Usernames ) from Tweets
["rt : #breaking: 's office confirms floyd county has found 2,600 ballots
during audit. says sec. raffensperger waâ€"]
/////
['really disgusting that the failing new york times allows dishonest
writers to totally fabricate stories.']
/////
['what an incredible opening by . i will put it up later for everyone to
see!']
```

در این مرحله علائم نگارشی و کاراکترهای خاص و اعداد باید از توییت‌ها حذف شوند که این قسمت هم به عهده regex است. هر کلمه یا عبارتی که با کاراکترهای مورد قبول شروع نشود حذف خواهد شد. بعضی از کاراکترها مانند # ها نیز نباید حذف شوند چون به وجود آن‌ها نیاز خواهیم داشت:

```
# Step 4:
# Remove punctuation, special characters and numbers from tweets
for i, tweet in enumerate(tweets):
    new_tweet = re.sub("[^a-z#&%\$ ]+", '', str(tweet))
    tweets[i] = new_tweet
```

نتیجه این قسمت را در شکل زیر می‌بینیم:

```
Step 4: Remove punctuation, special characters and numbers from tweets
rt #breaking s office confirms floyd county has found ballots during
audit says sec raffensperger wa
/////
really disgusting that the failing new york times allows dishonest writers
to totally fabricate stories
/////
what an incredible opening by i will put it up later for everyone to see

برای قسمت بعد به سراغ tokenizer می‌رویم. من از WhiteSpaceTokenizer برای این کار استفاده کردم. زیرا
فاصله‌ها را حذف می‌کند و مواردی که می‌ماند را به ما می‌دهد. اگر از tokenizer ای مانند WordPunct استفاده
```

```
# Step 5:
# Apply Tokenization
for i, tweet in enumerate(tweets):
    new_tweet = WhitespaceTokenizer().tokenize(tweet)
    tweets[i] = new_tweet
```

می‌کردم، این tokenizer کاراکترهای خاص مانند # که اتفاقاً در مراحل بعد به آن نیاز داریم را از کلمات جدا می‌کرد که باعث مشکل می‌شد. برای همین بهترین انتخاب این قسمت به انتخاب من WhiteSpaceTokenizer است:

نتیجه این قسمت را مشاهده می‌کنید:

```
Step 5: Apply Tokenization
['rt', '#breaking', 's', 'office', 'confirms', 'floyd', 'county', 'has',
'found', 'ballots', 'during', 'audit', 'says', 'sec', 'raffensperger',
'wa']
/////
['really', 'disgusting', 'that', 'the', 'failing', 'new', 'york', 'times',
'allows', 'dishonest', 'writers', 'to', 'totally', 'fabricate', 'stories']
/////
['what', 'an', 'incredible', 'opening', 'by', 'i', 'will', 'put', 'it',
'up', 'later', 'for', 'everyone', 'to', 'see']
```

مرحله بعد حذف stopwords است. این‌ها کلماتی هستند که به طور معمول در متون استفاده می‌شوند (مانند the, a, ..) و به طور کلی تاثیر خاصی در معنی و مفهوم متن اصلی ندارند و عموماً نادیده گرفته می‌شوند. چون این کلمات اهمیت زیادی ندارند حذف آن‌ها می‌تواند در فضا صرفه‌جویی کند:

```
# Step 6:
# Removing Stop words
stop_words = set(stopwords.words('english'))
for i, tweet in enumerate(tweets):
    new_tweet = [w for w in tweet if not w in stop_words]
    tweets[i] = new_tweet
```

نتیجه اجرا این قسمت را می‌بینیم:

```
Step 6: Removing Stop words
['rt', '#breaking', 'office', 'confirms', 'floyd', 'county', 'found',
'ballots', 'audit', 'says', 'sec', 'raffensperger', 'wa']
/////
['really', 'disgusting', 'failing', 'new', 'york', 'times', 'allows',
'dishonest', 'writers', 'totally', 'fabricate', 'stories']
/////
['incredible', 'opening', 'put', 'later', 'everyone', 'see']
```

در این مرحله باید کلمات با طول کمتر از ۳ را حذف کنیم. اکثر این کلمات هم مانند stopword ها تغییر خاصی در معنی ایجاد نمی‌کنند و اضافی هستند. در اینجا مواردی مانند it که در توییت هم موجود است حذف می‌شوند. با این حال ممکن است بعضی از کلمات با معنی مانند فعل go نیز در این فرآیند حذف شوند که مطابق میل ما نیست:

```
# Step 7:
# Remove Words with len less than 3
for i, tweet in enumerate(tweets):
    new_tweet = [w for w in tweet if len(w) > 2]
    tweets[i] = new_tweet
```

نتیجه را می‌بینیم:

```
Step 7: Remove Words with len less than 3
['#breaking', 'office', 'confirms', 'floyd', 'county', 'found', 'ballots',
'audit', 'says', 'sec', 'raffensperger']
/////
['really', 'disgusting', 'failing', 'new', 'york', 'times', 'allows',
'dishonest', 'writers', 'totally', 'fabricate', 'stories']
/////
['incredible', 'opening', 'put', 'later', 'everyone', 'see']
```

مرحله بعد stemming است. به توضیح بیشتر Lancaster و porter می‌پردازیم. بیشتر کلماتی که توسط stem می‌شوند بیشتر از حالت اولیه خود می‌گیرند و به ریشه اصلی نزدیک می‌شوند اما تشخیص اصل آن‌ها دشوار می‌شود. اما porter بیشتر شکل اصلی کلمات را حفظ می‌کند. به عبارت دیگر می‌توان گفت Lancaster یک stemmer سختگیرتر نسبت به porter است. در حالی که نتیجه porter به چیزی که برای ما آشنا است نزدیک‌تر است. می‌توانیم این طور بگوییم که اگر فشرده‌سازی حداکثری برای کلماتمان می‌خواهیم Lancaster گزینه بهتری است در حالی که Porter نتیجه ملموس‌تر و نزدیک‌تر به کلمه اصلی را به ما خواهد داد. Lancaster نسبت به porter سریع‌تر است. در میان این دو snowball، stemmer قرار می‌گیرد که به سخت‌گیری Lancaster نیست اما مانند porter نیز آنقدر به شکل اصلی کلمات وفادار نیست.

```
# Step 8:
# Stemming
porter = PorterStemmer()
for i, tweet in enumerate(tweets):
    new_tweet = [porter.stem(w) for w in tweet]
    tweets[i] = new_tweet
```

نتیجه آن را می‌بینیم:

```
Step 8: Stemming
['#break', 'offic', 'confirm', 'floyd', 'counti', 'found', 'ballot',
'audit', 'say', 'sec', 'raffensperg']
/////
['realli', 'disgust', 'fail', 'new', 'york', 'time', 'allow', 'dishonest',
'writer', 'total', 'fabric', 'stori']
/////
['incred', 'open', 'put', 'later', 'everyon', 'see']
```

۵ تا از پرتکرارترین کلمات به شکل زیر هستند:

```
Top 5 Words:
[('great', 7620), ('trump', 6464), ('thank', 5771), ('&', 5556),
('presid', 4831)]
```

```
string_of_All = " ".join(all_words)
trend_reg = re.compile(r"#\w+")
trends = Counter(trend_reg.findall(string_of_All))
trends = trends.most_common(5)
print(trends)

plt.bar([title for title, freq in trends], [freq for title, freq in trends])
plt.title("Trending Hashtags")
plt.xticks(rotation=90)
plt.show()
```

```
[('#trump', 995), ('#makeamericagreatagain', 556), ('#maga', 518), ('#celebapprentic', 295), ('#celebrityapprentic', 151)]
```

