تمرین اول NLP

یاسمن لطف‌اللهی - ۹۷۵۲۱۴۸۶

سوالات تئوری

۱. ابزارهای پردازش زبان طبیعی

NLTK، SpaCy و Stanford CoreNLP سه ابزار مهم و پرکاربرد در زمینه پردازش زبان طبیعی هستند.

۱. NLTK یکی از پراستفاده ترین کتابخانه های پایتون در زمینه NLP است. این کتابخانه منابع زیادی مانند WordNet را در بر دارد. همچنین دارای ابزارهای بسیاری برای تسک های مختلف است که با استفاده از آن ها می توانیم کارهای مانند پیش پردازش را سریع تر و راحت تر جلو ببریم. از جمله فیچرهای این لایبرری، توکن سازی، Stemming، طبقه بندی متن هستند.

۲. SpaCy یکی از جدیدترین کتابخانه های پایتون در زمینه NLP است. این کتابخانه برای هر تسک، فقط بهترین روش ها را ارائه می دهد و مانند NLTK تنوع زیادی ندارد. چون با Cython نوشته شده است، از Performance بالایی برخوردار است و سریع تر است. همچنین دارای مدل های مختلفی از جمله ترنسفورمرها است.

۳. Stanford CoreNLP یک کتابخانه پراستفاده به زبان جاوا ست. این کتابخانه نیز تسک های مختلفی از جمله ساخت توکن و Parse کردن جملات را می تواند انجام دهد با این کتابخانه می توانیم به راحتی تسک های پیچیده با حجم بالا را انجام دهیم.

۴. عبارات منظم

آ) ^(09|((00|\+)989))\d{0,16}

ب) ^(0[1-9]|[12][0-9]|3[01])-(0[1-9]|1[0-2])-\d{4}$

ج) ^((http:\/\/)?(www.)?)\w+(\.org|\.ir)$

د) ^(\d{2})[A-Z]\d{3}IR(\d{2})$

سوالات عملی

۱. regex

بررسی این کلمات با regex زیر صورت گرفت. کلمات در فایل doctors_names قرار دارند. برای تست کلماتی که نباید قبول شوند، چند کلمه به پایان این فایل اضافه شد.

```
regex = re.compile(r"((Doctor|Dr\.) ([A-Z][a-z-\.]+) ?)+)|(([A-Z][a-z\.]+,?( )?)+M\.D\.)")
```

با دستور fullmatch، بررسی شد که کلمات توسط regex قبول می شوند یا خیر.

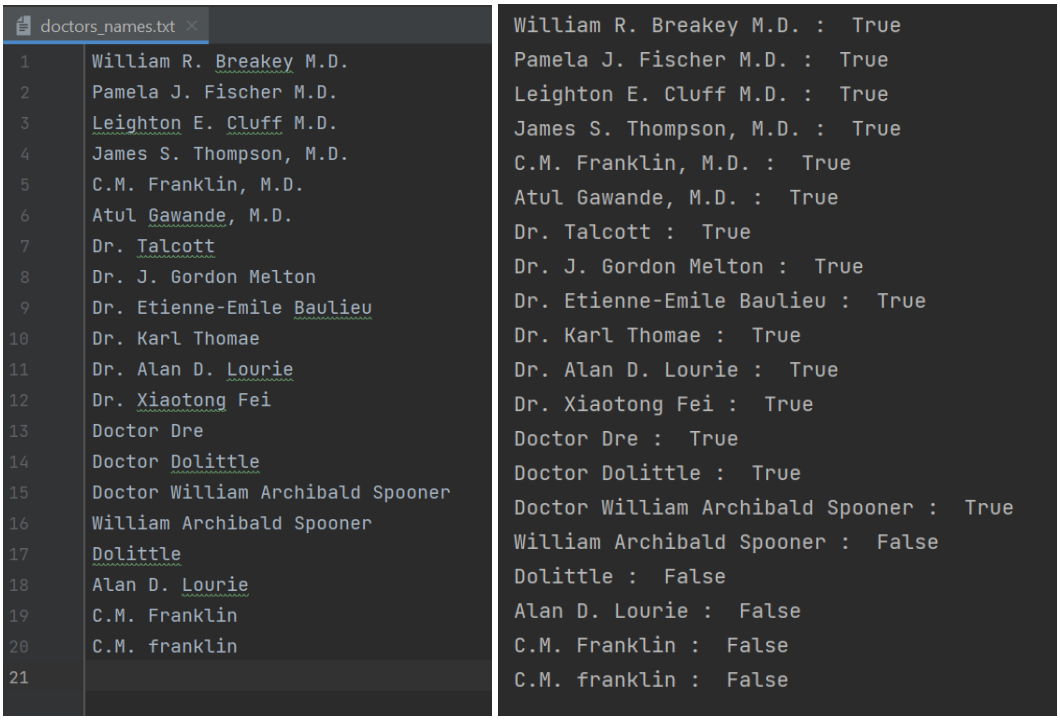
- **کد**

```
import re
regex = re.compile(r"((Doctor|Dr\.) ([A-Z][a-z-\.]+) ?)+)|(([A-Z][a-z\.]+,?( )?)+M\.D\.)")
def read_names():
    names = []
    with open('doctors_names.txt') as f:
        lines = f.readlines()
        for line in lines:
            names.append(line.strip())
    return names

def match_regex(text):
    match = regex.fullmatch(text)
    if match is None:
        return "False"
    return "True"


if __name__ == '__main__':
    names = read_names()
    for name in names:
        print(name, ": ", match_regex(name))
```

- **محتویات فایل تکست و نتیجه اجرا**



doctors_names.txt

```
William R. Breakey M.D.
Pamela J. Fischer M.D.
Leighton E. Cluff M.D.
James S. Thompson, M.D.
C.M. Franklin, M.D.
Atul Gawande, M.D.
Dr. Talcott
Dr. J. Gordon Melton
Dr. Etienne-Emile Baulieu
Dr. Karl Thomae
Dr. Alan D. Lourie
Dr. Xiaotong Fei
Doctor Dre
Doctor Dolittle
Doctor William Archibald Spooner
William Archibald Spooner
Dolittle
Alan D. Lourie
C.M. Franklin
C.M. franklin
```

```
William R. Breakey M.D. :  True
Pamela J. Fischer M.D. :  True
Leighton E. Cluff M.D. :  True
James S. Thompson, M.D. :  True
C.M. Franklin, M.D. :  True
Atul Gawande, M.D. :  True
Dr. Talcott :  True
Dr. J. Gordon Melton :  True
Dr. Etienne-Emile Baulieu :  True
Dr. Karl Thomae :  True
Dr. Alan D. Lourie :  True
Dr. Xiaotong Fei :  True
Doctor Dre :  True
Doctor Dolittle :  True
Doctor William Archibald Spooner :  True
William Archibald Spooner :  False
Dolittle :  False
Alan D. Lourie :  False
C.M. Franklin :  False
C.M. franklin :  False
```

۲. آشنایی با nltk

برای بررسی تو متود word_tokenize و sent_tokenize از متن زیر استفاده شد که بخشی از کتاب Speech and Language Processing است. متن در فایل q2_sample_text.txt ذخیزه شده است.

```
Of course modern conversational agents are much more than a diversion; they can answer questions, book flights, or
find restaurants, functions for which they rely on a much more sophisticated understanding of the user's intent, as
we will see in Chapter 24. Nonetheless, the simple pattern-based methods that powered ELIZA and other chatbots play
a crucial role in natural language processing.
We'll begin with the most important tool for describing text patterns: the regular expression. Regular expressions
can be used to specify strings we might want to extract from a document, from transforming "I need X" in Eliza
above, to defining strings like $199 or $24.99 for extracting tables of prices from a document.
```

قبل از توکن سازی باید پکیج punkt را دانلود کنیم. سپس می توانیم از متود ها استفاده کنیم.

- **کد**

```python
from nltk.tokenize import word_tokenize, sent_tokenize
import nltk

nltk.download('punkt')

def read_text():
    with open('sample_text.txt', encoding="utf-8") as f:
        txt = f.read()
    return txt

if __name__ == '__main__':
    text = read_text()
    print("text: ", text)
    print("work_tokenize: ", word_tokenize(text))
    print("sent_tokenize: ", sent_tokenize(text))
```

- **نتیجه اجرا**

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
text:  Of course modern conversational agents are much more than a diversion; they can answer questions,
We'll begin with the most important tool for describing text patterns: the regular expression. Regular ex
work_tokenize:  ['Of', 'course', 'modern', 'conversational', 'agents', 'are', 'much', 'more', 'than', 'a'
sent_tokenize:  ['Of course modern conversational agents are much more than a diversion; they can answer
```

نتیجه در فایل q2_result.txt ذخیره شده است.

همانطور که مشاهده می شود، word_tokenize متن را به کلمات و توکن ها تبدیل می کند. sent_tokenize متن را به جملات کامل تبدیل می کند.

۳. نرمالسازی

برای حذف حروف تکراری، به کمک regex حروف تکراری را پیدا می کنیم. یکی از آن ها را حدف می کنیم و چک می کنیم که این کلمه وجود دارد. برای چک کردن کلمه، از words پکیج nltk استفاده می کنیم.

- در صورتی فرم صحیح کلمه وارد شده، حرف تکراری داشته باشد (مثال: hello)، تنها در صورتی فرم صحیح یافت می شود که تکرار حروف اضافه همان حرف تکراری باشد(مثال: hello).

- **کد**

```python
import nltk
from nltk.corpus import words
import re

nltk.download('words')
words = words.words()

repeat_regex = re.compile(r"(\w)\1+")


def remove_at_idx(string, idx):
    return string[0: idx:] + string[idx + 1::]


def fix_word(word):
    fixed_word = word
    should_break = 0
    while fixed_word not in words:
        repeated_char = (repeat_regex.search(fixed_word))
        if repeated_char is None:
            break
        fixed_word = remove_at_idx(fixed_word, repeated_char.span()[0])
        should_break += 1
        if should_break > len(word):
            break
    return fixed_word


if __name__ == '__main__':
    for i in range(5):
        input_word = input("please enter a word: ")
        print("correct word: ", fix_word(input_word.strip()))
```

- **نتیجه اجرا**

```
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
please enter a word: looove
correct word:  love
please enter a word: hellllo
correct word:  hello
please enter a word: phhhonnee
correct word:  phone
please enter a word: laptop
correct word:  laptop
please enter a word: aalreadyyy
correct word:  already
```

۴. Word Tokenization

الف - فایل ها را می خوانیم.

ب - پکیج های مورد نظر را ایمپورت می کنیم.

```
from nltk.tokenize import TreebankWordTokenizer, RegexpTokenizer,\
    WhitespaceTokenizer, WordPunctTokenizer
import nltk


def read_text(path):
    with open(path, encoding="utf-8") as f:
        txt = f.read()
    return txt


if __name__ == '__main__':
    # الف
    english_sample = read_text("AlbertEinstein.txt")
    english_short = read_text("ShortSampleEnglish.txt")
    persian_sample = read_text("Shahnameh.txt")
    persian_short = read_text("ShortSamplePersian.txt")
```

پ - تجزیه توسط TreebankWordTokenizer و بدست آوردن تعداد و انواع توکن ها

- **کد**

```
    print("---TreeBankTokenizer-----------------------")
    treebank_tokenizer = TreebankWordTokenizer()
    tree_bank_english_sample_res = treebank_tokenizer.tokenize(english_sample)
    print(
        f"English sample:\nTokens num: {len(tree_bank_english_sample_res)}, Types num:
{len(set(tree_bank_english_sample_res))}")
    tree_bank_persian_sample_res = treebank_tokenizer.tokenize(persian_sample)
    print(
        f"Persian sample:\nTokens num: {len(tree_bank_persian_sample_res)}, Types num:
{len(set(tree_bank_persian_sample_res))}")
    tree_bank_english_short_res = treebank_tokenizer.tokenize(english_short)
    print(
        f"English short:\nTokens num: {len(tree_bank_english_short_res)}, Types num:
{len(set(tree_bank_english_short_res))}")
    tree_bank_persian_short_res = treebank_tokenizer.tokenize(persian_short)
    print(
        f"Persian short:\nTokens num: {len(tree_bank_persian_short_res)}, Types num:
{len(set(tree_bank_persian_short_res))}")
```

- **نتیجه اجرا**

```
---TreeBankTokenizer-----------------------
English sample:
Tokens num: 250, Types num: 131
Persian sample:
Tokens num: 2207, Types num: 921
English short:
Tokens num: 19, Types num: 18
Persian short:
Tokens num: 16, Types num: 16
```

ت - برای این Tokenizer باید یک regex تعریف کنیم. یک Tokenizer برای جداسازی کلمات و یکی برای جداسازی اعداد تعریف می کنیم.

برای تعریف Tokenizer جداسازی کلمات از \s و gaps=True استفاده می کنیم. در این صورت Tokenizer فاصله ها (\s+) را به عنوان جداکننده در نظر می گیرد و کلمات بینشان را به عنوان توکن برمی گرداند.

- **کد**

```
print("---RegexpTokenizer-----------------------")
word_re_tokenizer = RegexpTokenizer('\s+', gaps=True)
num_re_tokenizer = RegexpTokenizer('\d+')
num_re_english_sample_res = num_re_tokenizer.tokenize(english_sample)
print(f"English sample numbers:\n{', '.join(num_re_english_sample_res)}")
word_re_english_short_res = word_re_tokenizer.tokenize(english_short)
print(f"English short words:\n{', '.join(word_re_english_short_res)}")
word_re_persian_short_res = word_re_tokenizer.tokenize(persian_short)
print(f"Persian short words:\n{', '.join(word_re_persian_short_res)}")
```

- **نتیجه اجرا**

```
---RegexpTokenizer-----------------------
English sample numbers:
14, 1879, 1896, 1901, 1905, 1908, 1909, 1911, 1914, 1914, 1933, 1940, 1945
English short words:
Hello!, Hope, you're, doing, well., It, is, a, sample, short, text.This, is, our, 1, st, assignment.
Persian short words:
سلام!, امیدوارم, خوب, باشید., این, یک, متن, کوتاه, نمونه, است.این, تمرین, ۱, ما, است.
```

ث - از WhitespaceTokenizer استفاده می کنیم تا متن کوتاه انگلیسی را tokenize کنیم. این tokenizer توکن ها را براساس فاصله های بینشان جدا می کند. برای عملکرد مشابه، می توانیم از RegexpTokenizer با regex \s+ استفاده کنیم.

- **کد**

```
whitespace_tokenizer = WhitespaceTokenizer()
whitespace_regex_tokenizer = RegexpTokenizer('\s+', gaps=True)
whitespace_english_short_res = whitespace_tokenizer.tokenize(english_short)
whitespace_re_english_short_res = whitespace_regex_tokenizer.tokenize(english_short)
print(f"English short words with whitespace tokenizer:\n{', '.join(whitespace_english_short_res)}")
print(f"English short words with whitespace regex tokenizer:\n{', '.join(whitespace_re_english_short_res)}")
```

- **نتیجه اجرا**

```
---WhitespaceTokenizer-----------------------
English short words with whitespace tokenizer:
Hello!, Hope, you're, doing, well., It, is, a, sample, short, text.This, is, our, 1, st, assignment.
English short words with whitespace regex tokenizer:
Hello!, Hope, you're, doing, well., It, is, a, sample, short, text.This, is, our, 1, st, assignment.
```

ج -

WordPunctTokenizer متن را براساس regex \w+|[^\w\s]+، tokenize می کند. با این کار متن به مجموعه ای از عبارات الفبایی و غیرالفبایی تبدیل می کند. نتیجه اجرا روی متن کوتاه، توکن هایی تشکیل شده از کلمات و علائم نگارشی است.

- **کد**

```
print("---WordPunctTokenizer-------------------------")
    wordpunct_tokenizer = WordPunctTokenizer()
    wordpunct_english_short_res = wordpunct_tokenizer.tokenize(english_short)
    print(f"English short words:\n{', '.join(wordpunct_english_short_res)}")
```

- **نتیجه اجرا**

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

۵. Stemming

آ- دو کلاس LancasterStemmer و PorterStemmer را از nltk.stem می توانیم ایمپورت کنیم. می توانیم از هرکدام یک آبجکت بسازیم و با تابع stem، به ریشه کلمات برسیم. این تابع متن را نمی تواند بررسی کند و باید کلمات را تکی به آن بدهیم.

ب -

- **کد**

همانند قبل، توکن ها را بدست می آوریم. سپس با توجه به توضیحات آ، از دو Stemmer استفاده می کنیم تا تفاوت ها را مشاهده کنیم.

```
english_sample = read_text("AlbertEinstein.txt")
indices = [2, 10, 18, 19, 21, 22, 42]

porter_stemmer = PorterStemmer()
lanc_stemmer = LancasterStemmer()

treebank_tokenizer = TreebankWordTokenizer()
tokens = treebank_tokenizer.tokenize(english_sample)

for idx in indices:
    word = tokens[idx]
    print(f"word: {word}, porter: {porter_stemmer.stem(word)}, lank: {lanc_stemmer.stem(word)}")
```

- **نتیجه اجرا**

```
word: was, porter: wa, lank: was
word: Germany, porter: germani, lank: germany
word: weeks, porter: week, lank: week
word: later, porter: later, lank: lat
word: family, porter: famili, lank: famy
word: moved, porter: move, lank: mov
word: Italy, porter: itali, lank: ita
```

به طور کلی، Lancaster به عبارتی قوی تر از Porter است و بیشتر کلمات را کوتاه می کند. همین ویژگی ممکن است باعث ایجاد خطا شود.

پ-

- **کد**

```
words = ["Waves", "fishing", "rocks", "was", "corpora", "better", "ate", "broken"]
lemmatizer = WordNetLemmatizer()
for w in words:
    print(f"word: {w}, lemmatized (default): {lemmatizer.lemmatize(w)}")
```

- **نتیجه اجرا**

```
word: waves, lemmatized (default): wave
word: fishing, lemmatized (default): fishing
word: rocks, lemmatized (default): rock
word: was, lemmatized (default): wa
word: corpora, lemmatized (default): corpus
word: better, lemmatized (default): better
word: ate, lemmatized (default): ate
word: broken, lemmatized (default): broken
```

ت - همانطور که مشاهده می شود، ریشه بعضی کلمات اشتباه برگردانده شده است. این ممکن است بخاطر این باشد که تابع lemmatize براساس نقش کلمه در جمله (Part of speech - POS) عمل می کند. در صورتی که این پارامتر پاس داده نشود، مقدار default آن، "اسم" قرار می گیرد که برای تمام کلمات بالا درست نیست.

برای تصحیح، می توانیم در تابع lemmatize، یک پارامتر pos پاس دهیم. برای پیدا کردن pos می توانیم از ابزارهای مختلف استفاده کنیم. چون در اینجا تعداد کلمات کم است و جمله ای هم نداریم، دستی پاس دادم. براساس داکیومنت nltk، برای اسم "n"، برای صفت "a" و برای فعل "v" باید پاس داده شود.

- **کد**

```
poss = ["n", "v", "n", "v", "n", "a", "v", "a"]
for i in range(len(words)):
    print(f"word: {w}, lemmatized (with pos): {lemmatizer.lemmatize(words[i], poss[i])}")
```

- **نتیجه اجرا**

```
word: waves, lemmatized (with pos): wave
word: fishing, lemmatized (with pos): fish
word: rocks, lemmatized (with pos): rock
word: was, lemmatized (with pos): be
word: corpora, lemmatized (with pos): corpus
word: better, lemmatized (with pos): good
word: ate, lemmatized (with pos): eat
word: broken, lemmatized (with pos): break
```

۶. پیش پردازش داده ها

هر مرحله با کامنت در کد مشخص شده است.

- **کد**

```python
def preprocess(text: str, print_steps=False):
    # 1 - remove redundant spaces
    prepro_text = re.sub(r"\s+", " ", text)
    if print_steps:
        print("remove redundant spaces: ", prepro_text)
    # 2 - lowercase
    prepro_text = prepro_text.lower()
    if print_steps:
        print("lowercase: ", prepro_text)
    # 3 - remove handles
    prepro_text = re.sub(r"@\S+ ", "", prepro_text)
    if print_steps:
        print("remove handles: ", prepro_text)
    # 4 - remove numbers, special characters and punctuation
    prepro_text = re.sub(r"[^A-Za-z\s]+", " ", prepro_text)
    if print_steps:
        print("remove special characters and punc: ", prepro_text)
    # 5 - tokenize with nltk recommended tokenizer
    tokens = word_tokenize(prepro_text)
    if print_steps:
        print("tokens: ", tokens)
    # 6 - remove stop words
    filtered_tokens = [token for token in tokens if not token in stop_words]
    if print_steps:
        print("removed stopwords: ", filtered_tokens)
    # 7 - remove short tokens
    filtered_tokens = [token for token in filtered_tokens if len(token) > 3]
    if print_steps:
        print("remove tokens shorter than 3: ", filtered_tokens)
    # 8 - stem with porter stemmer
    porter_stemmer = PorterStemmer()
    stemmed_tokens = [porter_stemmer.stem(token) for token in filtered_tokens]
    if print_steps:
        print("stemmed tokens: ", stemmed_tokens)
    all_tokens.extend(stemmed_tokens)
    full_tweet = " ".join(stemmed_tokens)
    return full_tweet
```

- **نتیجه اجرا برای توییت اول**

```
tweet:  Republicans and Democrats have both created our economic problems.
remove redundant spaces:  Republicans and Democrats have both created our economic problems.
lowercase:  republicans and democrats have both created our economic problems.
remove handles:  republicans and democrats have both created our economic problems.
remove special characters and punc:  republicans and democrats have both created our economic problems
tokens:  ['republicans', 'and', 'democrats', 'have', 'both', 'created', 'our', 'economic', 'problems']
removed stopwords:  ['republicans', 'democrats', 'created', 'economic', 'problems']
remove tokens shorter than 3:  ['republicans', 'democrats', 'created', 'economic', 'problems']
stemmed tokens:  ['republican', 'democrat', 'creat', 'econom', 'problem']
```

- **نتیجه اجرا برای توییت دوم**

```
tweet:  I was thrilled to be back in the Great city of Charlotte, North Carolina with thousands of hardworking
American Patriots who love our Country, cherish our values, respect our laws, and always put AMERICA FIRST! Thank
you for a wonderful evening!! #KAG2020 https://t.co/dNJZfRsl9y
remove redundant spaces:  I was thrilled to be back in the Great city of Charlotte, North Carolina with thousands of
hardworking American Patriots who love our Country, cherish our values, respect our laws, and always put AMERICA
FIRST! Thank you for a wonderful evening!! #KAG2020 https://t.co/dNJZfRsl9y
lowercase:  i was thrilled to be back in the great city of charlotte, north carolina with thousands of hardworking
american patriots who love our country, cherish our values, respect our laws, and always put america first! thank
you for a wonderful evening!! #kag2020 https://t.co/dnjzfrsl9y
remove handles:  i was thrilled to be back in the great city of charlotte, north carolina with thousands of
hardworking american patriots who love our country, cherish our values, respect our laws, and always put america
first! thank you for a wonderful evening!! #kag2020 https://t.co/dnjzfrsl9y
remove special characters and punc:  i was thrilled to be back in the great city of charlotte  north carolina with
thousands of hardworking american patriots who love our country  cherish our values  respect our laws  and always
put america first  thank you for a wonderful evening   kag  https t co dnjzfrsl y
tokens:  ['i', 'was', 'thrilled', 'to', 'be', 'back', 'in', 'the', 'great', 'city', 'of', 'charlotte', 'north',
'carolina', 'with', 'thousands', 'of', 'hardworking', 'american', 'patriots', 'who', 'love', 'our', 'country',
'cherish', 'our', 'values', 'respect', 'our', 'laws', 'and', 'always', 'put', 'america', 'first', 'thank', 'you',
'for', 'a', 'wonderful', 'evening', 'kag', 'https', 't', 'co', 'dnjzfrsl', 'y']
removed stopwords:  ['thrilled', 'back', 'great', 'city', 'charlotte', 'north', 'carolina', 'thousands',
'hardworking', 'american', 'patriots', 'love', 'country', 'cherish', 'values', 'respect', 'laws', 'always', 'put',
'america', 'first', 'thank', 'wonderful', 'evening', 'kag', 'https', 'co', 'dnjzfrsl']
remove tokens shorter than 3:  ['thrilled', 'back', 'great', 'city', 'charlotte', 'north', 'carolina', 'thousands',
'hardworking', 'american', 'patriots', 'love', 'country', 'cherish', 'values', 'respect', 'laws', 'always',
'america', 'first', 'thank', 'wonderful', 'evening', 'https', 'dnjzfrsl']
stemmed tokens:  ['thrill', 'back', 'great', 'citi', 'charlott', 'north', 'carolina', 'thousand', 'hardwork',
'american', 'patriot', 'love', 'countri', 'cherish', 'valu', 'respect', 'law', 'alway', 'america', 'first', 'thank',
'wonder', 'even', 'http', 'dnjzfrsl']
```

- **نتیجه اجرا برای توییت سوم**

```
tweet:  RT @CBS_Herridge: READ: Letter to surveillance court obtained by CBS News questions where there will be
further disciplinary action and choâ€¦
remove redundant spaces:  RT @CBS_Herridge: READ: Letter to surveillance court obtained by CBS News questions where
there will be further disciplinary action and choâ€¦
lowercase:  rt @cbs_herridge: read: letter to surveillance court obtained by cbs news questions where there will be
further disciplinary action and choâ€¦
remove handles:  rt read: letter to surveillance court obtained by cbs news questions where there will be further
disciplinary action and choâ€¦
remove special characters and punc:  rt read  letter to surveillance court obtained by cbs news questions where
there will be further disciplinary action and cho
tokens:  ['rt', 'read', 'letter', 'to', 'surveillance', 'court', 'obtained', 'by', 'cbs', 'news', 'questions',
'where', 'there', 'will', 'be', 'further', 'disciplinary', 'action', 'and', 'cho']
removed stopwords:  ['rt', 'read', 'letter', 'surveillance', 'court', 'obtained', 'cbs', 'news', 'questions',
'disciplinary', 'action', 'cho']
remove tokens shorter than 3:  ['read', 'letter', 'surveillance', 'court', 'obtained', 'news', 'questions',
'disciplinary', 'action']
stemmed tokens:  ['read', 'letter', 'surveil', 'court', 'obtain', 'news', 'question', 'disciplinari', 'action']
```

میزان تغییر توییت وابسته به خود توییت است. چون توییت ها توسط افراد مختلفی نوشته شده اند، ادبیات مختلفی دارند. برخی توییت ها رسمی هستند اما برخی به صورت عامیانه. پیش پردازش به ما کمک می کند که در آخر، به توییت های یکدستی برسیم که کار پردازش را برایمان راحت تر کند.

- **سوالات مطرح شده**

۵- برای توکن سازی از تابع word_tokenize استفاده شد. این تابع، با Tokenizerپیشنهادی nltk توکن ها را می سازد. در حال حاضر این tokenizer، نسخه بهبود یافته TreebankWordTokenizer به همراه PunktSentenceTokenizer برای برخی زبان ها است.

۶- stop words کلمات تکراری یک زبان هستند که اطلاعات زیادی به متن اضافه نمی کنند. برای همین هنگام پیش پردازش برای تسک های nlp، از متن حذف می شوند.

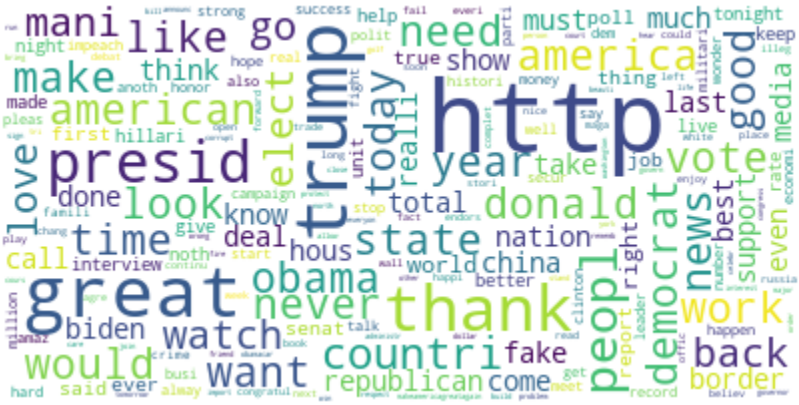
۷- کلمات کوتاه پرتکرار نیستند و ممکن است مخفف یک کلمه یا اشتباه تایپی باشند.

۸- LancasterStemmer و PorterStemmer در سوالات قبل استفاده شدند. PorterStemmer نتیجه نسبتا خوبی دارد و سریع است و برای همین پراستفاده است. LancasterStemmer معمولا قسمت بیشتری از کلمات را حذف می کند. این می تواند با تولید کلمات بی معنی، برای ما مشکل ساز شود.

- **پرتکرار ترین کلمات**

```
50 most common words:  [('http', 16968), ('trump', 8044), ('great', 7752), ('thank', 5974), ('presid', 5020),
('peopl', 3540), ('countri', 2799), ('vote', 2607), ('make', 2568), ('democrat', 2522), ('time', 2493), ('america',
2480), ('would', 2410), ('state', 2298), ('like', 2209), ('year', 2119), ('american', 2117), ('want', 2106),
('obama', 2021), ('work', 1965), ('good', 1938), ('donald', 1924), ('need', 1912), ('never', 1893), ('news', 1865),
('today', 1861), ('look', 1859), ('mani', 1755), ('love', 1698), ('elect', 1629), ('back', 1570), ('go', 1555),
('watch', 1524), ('republican', 1502), ('even', 1488), ('much', 1434), ('know', 1431), ('nation', 1416), ('think',
1393), ('total', 1376), ('show', 1365), ('fake', 1303), ('best', 1279), ('must', 1275), ('come', 1265), ('border',
1260), ('deal', 1258), ('china', 1247), ('media', 1225), ('call', 1218)]
```

- **WordCloud**



- **ترندها**

```
10 trends:  [('Trump2016', 745), ('MAGA', 472), ('MakeAmericaGreatAgain', 468), ('CelebApprentice', 270), ('1', 140), ('CelebrityApprentice', 128), ('TimeToGetTough', 92), ('AmericaFirst', 90), ('Trump', 80), ('DrainTheSwamp', 74)]
```