



## گزارش

گروه مهندسی کامپیوتر

## آشنایی با فرایند اندیس گذاری و مدل های بازیابی

نگارندگان:

عارف طالب زاده

سحر قصابی

مرضیه یاوری

استاد:

جناب آقای دکتر کاهانی

دستیار استاد:

جناب آقای مهندس سلطانی

آذر 1400

## چکیده

اولین گام جهت طراحی سیستم بازیابی اطلاعات این است که مدلی برای توصیف و تعیین مشابهت‌های موجود میان اطلاعاتی که در اختیار دارد با نیازهای اطلاعاتی کاربر تعریف کند. در این بخش مدل یا مدل‌های مورد استفاده‌ی موتور جستجوگر، برای بازیابی اطلاعات و رتبه‌بندی آنها بیان می‌شود. یکی از نکات اصلی که برای کاربر اهمیت زیادی دارد نحوه‌ی رتبه‌بندی نتایج بدست آمده توسط موتور جستجوگر است. تفاوت در کارایی موتورهای جستجو ناشی از الگوریتم‌ها و مدل‌های مختلفی است که در این قسمت از موتور جستجو پیاده‌سازی شده‌اند. یکی دیگر از نکات این مدل‌ها رفتار متفاوت آن‌ها در زبان‌های مختلف و مجموعه اسناد مختلف است. به این معنی که مدل‌های بازیابی اطلاعات که در موتورهای جستجو به منظور یافتن مشابه‌ترین سند به پرسش کاربر از میان اسناد موجود استفاده می‌شود، باید برای زبان‌های متفاوت (انگلیسی، فارسی و ...) پیاده‌سازی و ارزیابی شوند تا بتوان برای زبان مقصد بهترین مدل را انتخاب و استفاده کرد. حاصل تحقیقات گسترده در بازیابی اطلاعات، طراحی و معرفی مدل‌های مختلفی برای سیستم‌های بازیابی اطلاعات است. برخی از مهم‌ترین آنها، مدل فضای برداری (Space-Vector)، دودویی (Binary)، احتمالی-آماري، شبکه عصبی، فازی و غیره هستند. این مدل‌ها با توجه به مجموعه داده‌های مورد استفاده و زبان مقصد کارایی متفاوتی دارند. در این گزارش مراحل و فرآیند اندیس‌گذاری و پیاده‌سازی مدل بازیابی اطلاعات بولین و برداری (tf-idf) به ترتیب توضیح و بررسی شده است.

## فهرست مطالب

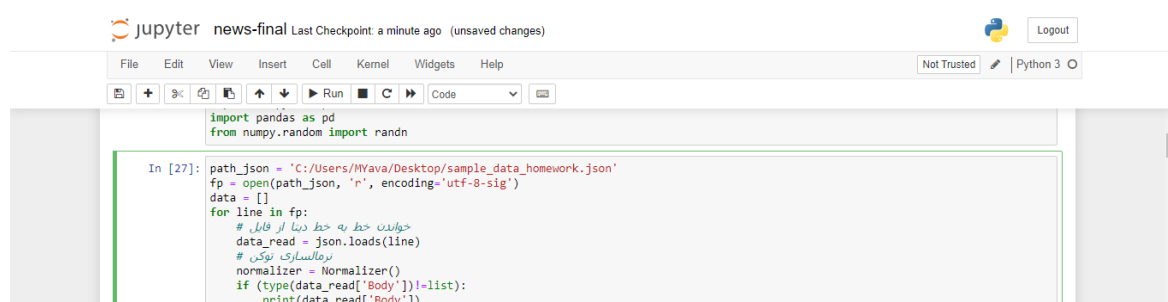
1- مرحله اول : دریافت متن خبر.....	4
2- مرحله دوم : انجام پیش پردازش .....	4
3- مرحله سوم : استخراج تعداد رخداد کلمات در اسناد .....	6
4- مرحله چهارم : ایجاد لغت نامه و لیست پست (PostingList).....	7
5- پیاده سازی مدل بولی .....	8
6- پیاده سازی مدل برداری (tf-idf) .....	9
7- پرس و جو.....	10
8- منابع .....	13

## فهرست اشکال

- شکل 1-1 : جداسازی خبر ..... 4
- شکل 1-2 : نرمال سازی متن خبر ..... 5
- شکل 2-2 : توکن سازی متن خبر ..... 5
- شکل 3-2 : ریشه یابی متن خبر ..... 6
- شکل 4-2 : چاپ متن خبر پیش پردازش شده ..... 6
- شکل 1-3 : شمارش تعداد رخداد کلمه ..... 7
- شکل 1-4 : ساخت posting list ..... 7
- شکل 2-4 : چاپ posting list ..... 8
- شکل 1-5 : ساخت مدل بولی ..... 8
- شکل 1-6 : ساخت مدل برداری ..... 9
- شکل 1-7 : نتایج پرس و جوی مدل بولی ..... 11
- شکل 2-7 : نتایج پرس و جوی مدل برداری ..... 12

## 1- مرحله اول : دریافت متن خبر

امروز بخش وسیعی از اطلاعات به صورت متن و مستندات و دیگر صورت های رسانه ای نگهداری می شود. برای دریافت دانش از اطلاعات یک متن، لازم است که ابتدا آن را پردازش کرد تا متوجه شد چه معانی در آن موجود می باشد بنابراین در مرحله اول ما نیاز داریم که متن خبر را از سایر اجزای خبر جدا کنیم تا بتوانیم در مرحله بعد آن اطلاعات را مورد پردازش قرار دهیم. جداسازی متن خبر در شکل 1-1 به صورت کد نشان داده شده است.



```
import pandas as pd
from numpy.random import randn

In [27]: path_json = 'C:/Users/Myava/Desktop/sample_data_homework.json'
fp = open(path_json, 'r', encoding='utf-8-sig')
data = []
for line in fp:
    # خواندن خط به خط دیتا از فایل
    data_read = json.loads(line)
    # نرمالسازی توکن
    normalizer = Normalizer()
    if (type(data_read['Body']) != list):
        print(data_read['Body'])
```

شکل 1-1 : جداسازی خبر

## 2- مرحله دوم : انجام پیش پردازش

پیش پردازش داده ها، یکی از مراحل مهم در فرایند بازیابی اطلاعات می باشد. در پیش پردازش سعی می شود صورت های غیر استاندارد به شکل استاندارد تبدیل شوند. اگر حروف، نشانه های نگارشی و کلمات فارسی به شکل یکسانی نوشته نشوند، متون مورد نظر قابل استفاده توسط رایانه نخواهد بود. بنابراین در این بخش روی متن های خبر پردازش های متنی نرمال سازی، توکن سازی و ریشه یابی انجام می شود.

نرمال سازی :

در نرمال سازی به دنبال تمیز و مرتب کردن متن و یکسان سازی کاراکترها با جایگزین کردن کاراکترهای استاندارد در متن ورودی می باشیم. در این مرحله باید همه حروف متن با جایگزینی با معادل استاندارد آن ها، یکسازی سازی گردند. علاوه بر این نیم فاصله و فاصله باید اصلاح و یکسان سازی شوند. در این بخش برای اعمال اصلاحات اولیه از تابع Normalizer از کلاس Normalizer در کتابخانه hazm برای نرمال سازی حروف متن خبر استفاده شده است.

```

import pandas as pd
from numpy.random import randn

In [27]: path_json = 'C:/Users/MyJava/Desktop/sample_data_homework.json'
fp = open(path_json, 'r', encoding='utf-8-sig')
data = []
for line in fp:
    # خواندن خط به خط دنیا از فایل
    data_read = json.loads(line)
    # نرمالسازی توکن
    normalizer = Normalizer()
    if (type(data_read['Body']) != list):
        print(data_read['Body'])
        norm = normalizer.normalize(data_read['Body'])

```

شکل 2-1: نرمال سازی متن خبر

توکن سازی :

در توکن سازی متن به قسمت های کوچک شکسته می شود. شکستن یک متن بر اساس واحدهای بامعنی مانند کلمه، پاراگراف، جمله و نمادهای معنا دار از توکن ساز استفاده می شود. در این بخش توکن ساز کلمات را در متن تشخیص داده و متن خبر را به دنباله ای از کلمات تبدیل می کند و به عنوان ورودی مراحل بعدی مانند ریشه یاب استفاده می شود.

```

import pandas as pd
from numpy.random import randn

In [27]: path_json = 'C:/Users/MyJava/Desktop/sample_data_homework.json'
fp = open(path_json, 'r', encoding='utf-8-sig')
data = []
for line in fp:
    # خواندن خط به خط دنیا از فایل
    data_read = json.loads(line)
    # نرمالسازی توکن
    normalizer = Normalizer()
    if (type(data_read['Body']) != list):
        print(data_read['Body'])
        norm = normalizer.normalize(data_read['Body'])
    # تبدیل متن خبر نرمال شده به توکن
    tokenizer = WordTokenizer()
    token = tokenizer.tokenize(norm)

```

شکل 2-2: توکن سازی متن خبر

ریشه یابی و بن یابی:

ریشه یابی اصطلاحی است که برای تعریف فرآیند کاهش دادن تعداد حروف یک کلمه و رسیدن به ریشه آن به کار می رود. منظور از ریشه در این تعریف، ریشه زبانی نیست و هدف این است که فرمت های گوناگون یک کلمه دارای ریشه های یکسان باشند. معمولاً ریشه یابی لغات براساس قواعد ساخت واژه ای و سپس حذف پسوند ها و پیشوندها می باشد. در بن یابی صورت های گوناگون صرفی یک واژه ساده می شود. در واقع بن واژه انتزاعی از صورت های مختلف یک واژه می باشد. هدف ریشه یابی و بن یابی کاهش اشکال فراوان و بعضی از اشکال متداول مشتق شده از یک کلمه به یک شکل پایه مشترک و اغلب شامل حذف عبارات مشتق شده است. برای انجام این کار ها از تابع های Stemmer از کتابخانه hazm استفاده شده است.

```

import pandas as pd
from numpy.random import randn

In [27]: path_json = 'C:/Users/MyJava/Desktop/sample_data_homework.json'
fp = open(path_json, 'r', encoding='utf-8-sig')
data = []
for line in fp:
    # خواندن خط به خط دنیا از فایل.
    data_read = json.loads(line)
    # نرمالسازی توکن.
    normalizer = Normalizer()
    if (type(data_read['Body']) != list):
        print(data_read['Body'])
        norm = normalizer.normalize(data_read['Body'])
        # تبدیل متن خبر نرمال شده به توکن.
        tokenizer = WordTokenizer()
        token = tokenizer.tokenize(norm)
        # ریشه یابی مبتنی بر قانون.
        stem = []
        stemmer = Stemmer()
        for i in range(0, len(token)):
            stem.append(stemmer.stem(token[i]))
        # ذخیره متن خبر بهمنی پردازش شده.
        data.append(stem)
    # تصمیم خود برای ساخت و ارسال یک
    # ماه نورد.
    # به کره ماه در سال ۲۰۲۳ خبر داده‌اند و پس از آن در سال ۲۰۲۸. فناوری‌ها دنیا دوباره قدم به کره ماه خواهد گذاشت.
    # هدف کارشناسان از

```

شکل 3-2: ریشه یابی متن خبر

```

In [5]: # چاپ دیدگاه پیش پردازش شده.
for i in range(0, len(data)):
    print('list data %i: %s' % (i, data[i]))

list data 0: ['آخرین روز و سال و ...']
list data 1: ['آخرین روز و سال و ...']

```

شکل 4-2: چاپ متن خبر پیش پردازش شده

### 3- مرحله سوم : استخراج تعداد رخداد کلمات در اسناد

زمانی که مراحل پیش پردازش انجام شد. متن خبرها تبدیل به دنباله ای از توکن شده است. در این مرحله باید مشخص کنیم از هر توکن چند بار در کل متن خبرها تکرار شده است و آن را ذخیره کنیم. با استفاده از تابع counter از کتابخانه collections تعداد رخداد هر توکن در خبر شمارش می‌شود سپس توکن به همراه تعداد تکرارش در دیکشنری ذخیره می‌شود.





```

In [7]: # چاپ تعداد رخداد هر توکن
print('token : times')
for token in dic_count:
    print('(%s : %i)' % (token, dic_count.get(token)))

print('=====')

f = open("posting_word.txt", "w", encoding='utf-8')
f.write(str(posting_word))
f.close()

# نمایش posting کلمات و
print('word: posting')
for word in posting_word:
    print('(%s : %s)' % (word, posting_word.get(word)))

print('=====')
(70 : 2)
(2 : 2)
(2 : 2)
(2 : 2)
(1 : 1)
(1 : 1)
(1 : 1)
(1 : 1)
(1 : 1)
(1 : 1)
(1 : 1)
(1 : 1)

```

شکل 4-2: چاپ posting list

## 5- پیاده سازی مدل بولی

مدل بولین بازیابی اطلاعات، یکی از اولین و ساده ترین روش های بازیابی، از تطبیق دقیق اسناد با پرسش کاربر یا درخواست اطلاعات با یافتن اسنادی که از نظر تطبیق کلمات در پرس و جو مرتبط هستند، استفاده می شود. مدل بازیابی اطلاعات بولی در نظر می گیرد که کدام کلمات کلیدی در یک سند یا عنوان وجود دارد یا وجود ندارد. بنابراین یک سند به عنوان مرتبط یا نامرتبط ارزیابی می شود. در مدل بازیابی بولی یک ماتریس براساس کلمات و متن خبر درست می شود. اگر کلمه در آن متن خبر وجود داشته باشد مقدار یک و اگر وجود نداشته باشد مقدار صفر می گیرد.

```

In [9]: np.random.seed(110)
t = list(posting_word.values())
arr = []
arr = [i for i in range(len(data))]
A = []

A = arr
df = pd.DataFrame(0, posting_word.keys(), A)
# in this "for" we check if a token exists in a document we put true otherwise false
for i in range(0, len(t)):
    df.loc[list(posting_word.keys())[i], A] = [a in list(posting_word.values())[i] for a in A]

pd.set_option('display.max_rows', 1269)
pd.set_option('display.max_columns', 10)
df

```

False	False	False	True	False	True	False	False	False	True	حوزه
False	False	False	False	False	False	False	False	False	True	فران
True	True	True	True	True	True	True	True	True	True	و
False	False	False	False	False	False	False	False	False	True	عشر
True	True	True	True	True	True	True	True	True	True	نگرود
False	False	False	False	True	False	False	False	False	True	فرهنگ
True	True	True	True	True	True	True	True	True	True	پشتگاه
True	True	True	True	True	True	True	True	True	True	خبرنگار

شکل 5-1: ساخت مدل بولی

پرس و جوهایی که در مدل بولی پرسیده میشود به شکل عبارت بولی بوده و از عملگرهای AND، OR و NOT برای اتصال کلمات پرس و جو استفاده می کند. برای مثال، AND بولی دو عبارت منطقی x

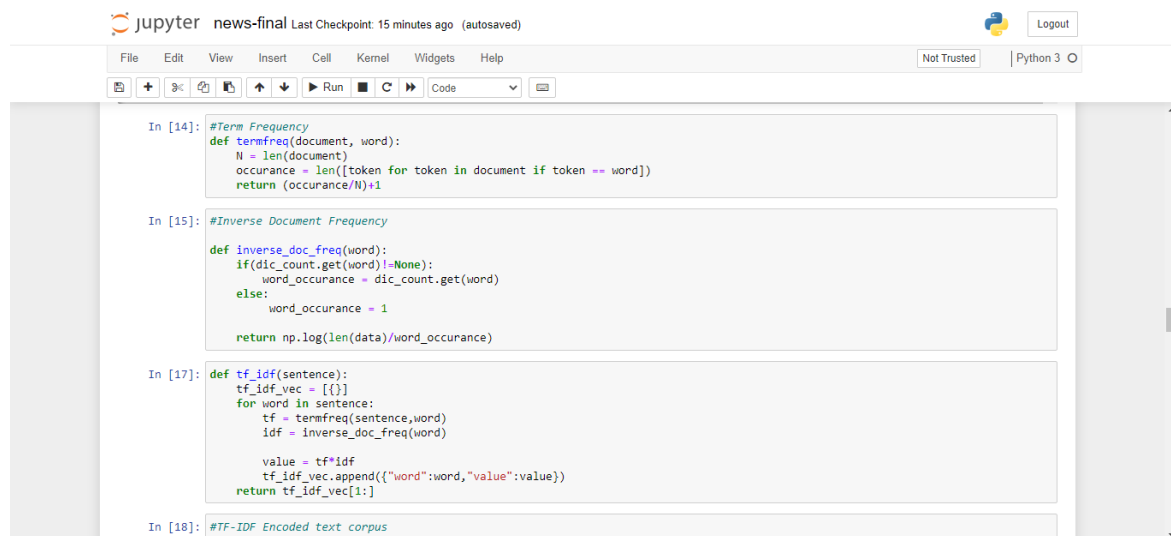
و  $y$  به این معنی است که هر دو  $x$  و  $y$  باید برآورده شوند، در حالی که OR بولی همین دو عبارت به این معنی است که حداقل یکی از این دستورات باید برآورده شود. هر تعداد گزاره منطقی را می توان با استفاده از سه عملگر بولی ترکیب کرد.

## 6- پیاده سازی مدل برداری (tf-idf)

مدل بازایی tf-idf یک معیار آماری است که میزان ارتباط یک کلمه را با یک سند در مجموعه ای از اسناد ارزیابی می کند. اغلب به عنوان عامل وزن توسط موتورهای جستجو برای امتیازدهی و رتبه بندی ارتباط یک سند با توجه یک پرس و جو استفاده می شود. مقدار tf-idf با ضرب دو معیار انجام می شود.

$$w_{t,d} = (1 + \log tf_{t,d}) \times \log_{10}(\frac{N}{df_t})$$

در tf-idf با افزایش متناسب با تعداد دفعاتی که یک کلمه در یک سند ظاهر می شود، کار می کند، اما با تعداد اسنادی که حاوی کلمه هست جبران می شود. بنابراین کلماتی که در هر سند رایج هستند با وجود اینکه ممکن است بارها ظاهر شوند، رتبه پایینی دارند، زیرا آن سند معنی خاصی ندارند. با این حال اگر کلمه ای بارها در یک سند ظاهر شود در حالی که بارها در اسناد دیگر ظاهر نشود، احتمالاً به این معنی است که مرتبط است.



```

jupyter news-final Last Checkpoint: 15 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3
In [14]: #Term Frequency
def termfreq(document, word):
    N = len(document)
    occurrence = len([token for token in document if token == word])
    return (occurrence/N)+1

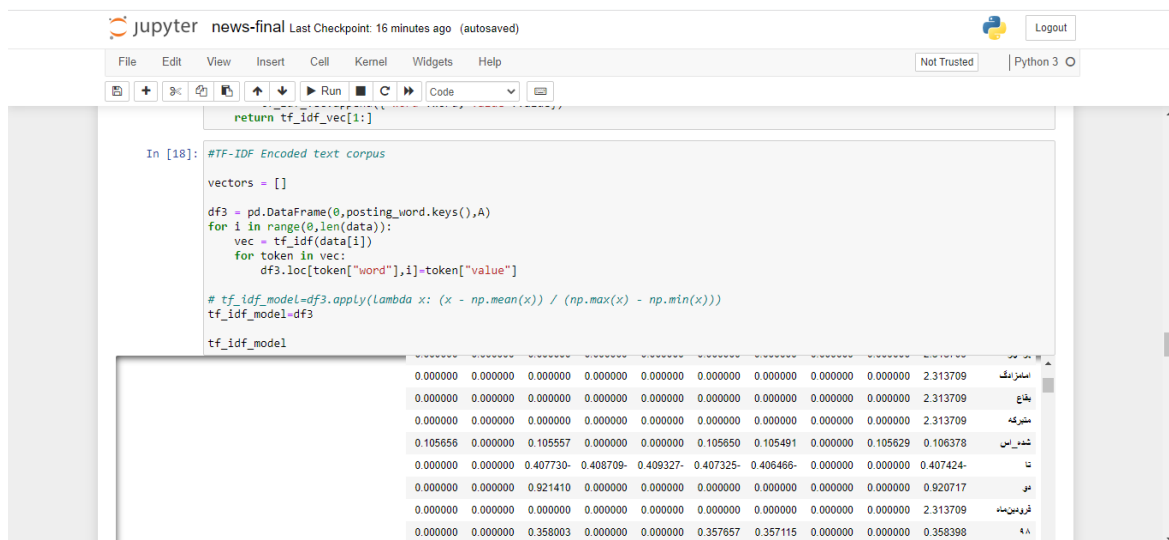
In [15]: #Inverse Document Frequency
def inverse_doc_freq(word):
    if(dic_count.get(word)!=None):
        word_occurrence = dic_count.get(word)
    else:
        word_occurrence = 1
    return np.log(1+len(data)/word_occurrence)

In [17]: def tf_idf(sentence):
tf_idf_vec = []
for word in sentence:
    tf = termfreq(sentence,word)
    idf = inverse_doc_freq(word)

    value = tf*idf
    tf_idf_vec.append({"word":word,"value":value})
return tf_idf_vec[1:]

In [18]: #TF-IDF Encoded text corpus

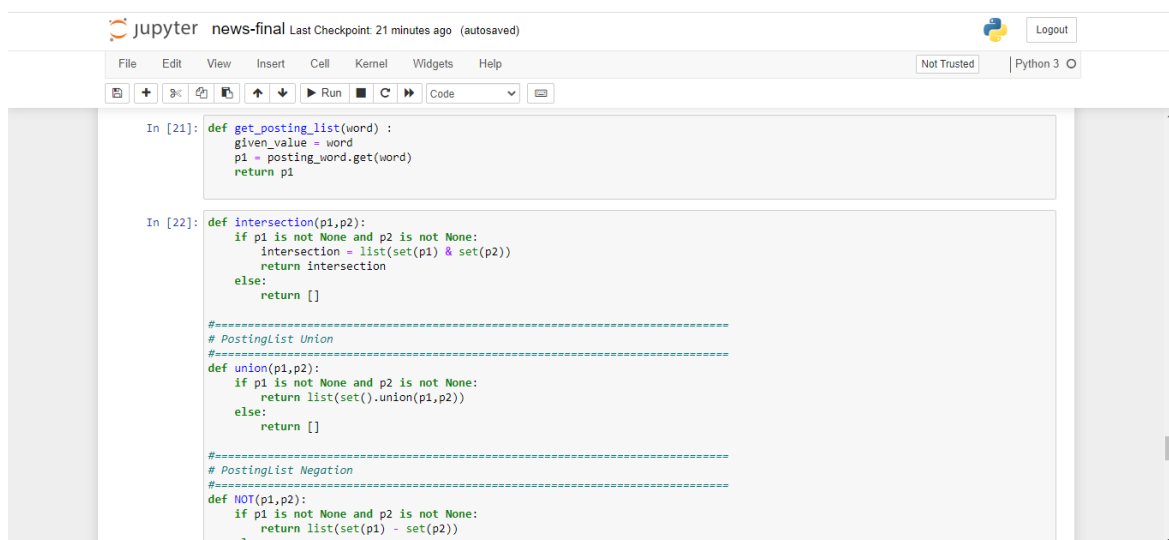
```



شکل 6-1: ساخت مدل برداری

## 7- پرس و جو

در این مرحله برای دو مدل بولی و برداری یک پرس و جو نوشته شده و نتایج برگردانده شده توسط دو مدل را در شکل 7-1 و 7-2 مشاهده می کنید.



```

jupyter news-final Last Checkpoint: 22 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Code

# =====
# PostingList Negation
# =====
def NOT(p1,p2):
    if p1 is not None and p2 is not None:
        return list(set(p1) - set(p2))
    else:
        return []

In [23]: def query_handler(query,posting_word):

    print(query)
    queryB=[]
    normalizer = Normalizer()
    norm = normalizer.normalize(query)
    print('norm: ', norm)
    # تبدیل متن جستجو شده به توکن
    tokenizer = WordTokenizer()
    token = tokenizer.tokenize(norm)
    # print('token: ', token)
    # رشته باقی فاصله بر قانون
    stem = []
    stemmer = Stemmer()
    for i in range(0, len(token)):
        stem.append(stemmer.stem(token[i]))

    query = query.split(" ")
    term = query[0]
    posting = get_posting_list(stemmer.stem(term))

```

```

jupyter news-final Last Checkpoint: 23 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Code

query = query.split(" ")
term = query[0]
posting = get_posting_list(stemmer.stem(term))
documents = posting
for index in range(1,len(query)):
    if(query[index] == "AND"):
        op = '&'

    elif(query[index]=="OR"):
        op = '|'
    elif(query[index]=="NOT"):
        op = '!'
    else:
        if(op == '&'):
            term = normalizer.normalize(query[index])
            term = stemmer.stem(term)
            term = get_posting_list(term)
            documents = intersection(documents,term)
        elif(op == '|'):
            term = normalizer.normalize(query[index])
            term = stemmer.stem(term)
            term = get_posting_list(term)
            documents = union(documents,term)
        elif(op == '!'):
            term = normalizer.normalize(query[index])
            term = stemmer.stem(term)
            term = get_posting_list(term)
            documents = list(set(documents) - set(term))

return documents

```

```

jupyter news-final Last Checkpoint: 25 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Code

import IPython.display as display
from IPython.display import display
textBox=widgets.Text(
    value= '',
    placeholder="Enter your query",
    Description='',
    disabled=False
)
button = widgets.Button(description="Search")

Box = widgets.HBox([textBox, button])

textValue = ''

def on_button_clicked2(b):
    global textValue
    textValue=textBox.value
    print(textValue)
    results = query_handler(textValue,posting_word)
    print('result is: ',results)

button.on_click(on_button_clicked2)
Box

نتایج AND به NOT گزارش
گزارش AND به NOT نتایج
گزارش AND به NOT نتایج
norm
result is: [0, 2, 3, 4, 5, 6, 7, 8, 9]

```

شکل 7-1: نتایج پرس و جوی مدل بولی

jupyter news-final Last Checkpoint: 18 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```

In [20]: textBox=widgets.Text(
value= '',
placeholder="Enter your query",
Description="",
disabled=False
)
button = widgets.Button(description="Search")

Box = widgets.HBox([textBox, button])

textValue = ''
cosine_sim=[]
def on_button_clicked(b):
    global textValue
    textValue=textBox.value
    print(textValue)
    query=[]
    #value is accessible now!
    normalizer = Normalizer()
    norm = normalizer.normalize(textValue)
    # print('norm: ', norm)
    # تبدیل متن خبر به مال شده به توکن
    tokenizer = WordTokenizer()
    token = tokenizer.tokenize(norm)
    # print('token: ', token)
    # ریشه بانی مبتنی بر قانون
    stem = []
    stemmer = Stemmer()

```

jupyter news-final Last Checkpoint: 18 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```

norm = normalizer.normalize(textValue)
# print('norm: ', norm)
# تبدیل متن خبر به مال شده به توکن
tokenizer = WordTokenizer()
token = tokenizer.tokenize(norm)
# print('token: ', token)
# ریشه بانی مبتنی بر قانون
stem = []
stemmer = Stemmer()
for i in range(0, len(token)):
    stemWord=stemmer.stem(token[i]);
    if(stemWord in word_set):
        stem.append(stemWord)
# print('stem: ', stem)

# ذخیره متن خبر به مال شده
query.append(stem)
print(query)
#TF-IDF Encoded text corpus

vectors2 = []
A=['Query']
query_model = pd.DataFrame(0,posting_word.keys(),A)
for token in query:
    term=tf_idf(token)
    for t in term:
        query_model.loc[t["word"],A]=t["value"]

# tf_idf_query_model=query_model.apply(Lambda x: (x - np.mean(x)) / (np.max(x) - np.min(x)))
tf_idf_query_model=query_model

```

jupyter news-final Last Checkpoint: 19 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```

vectors2 = []
A=['Query']
query_model = pd.DataFrame(0,posting_word.keys(),A)
for token in query:
    term=tf_idf(token)
    for t in term:
        query_model.loc[t["word"],A]=t["value"]

# tf_idf_query_model=query_model.apply(Lambda x: (x - np.mean(x)) / (np.max(x) - np.min(x)))
tf_idf_query_model=query_model
print(tf_idf_query_model)

from sklearn.metrics.pairwise import cosine_similarity
for i in range(0,len(data)):
    x = np.array(tf_idf_model[i]).reshape(1,-1)
    y = np.array(tf_idf_query_model['Query']).reshape(1,-1)
    cosine_sim.append(cosine_similarity(x, y))

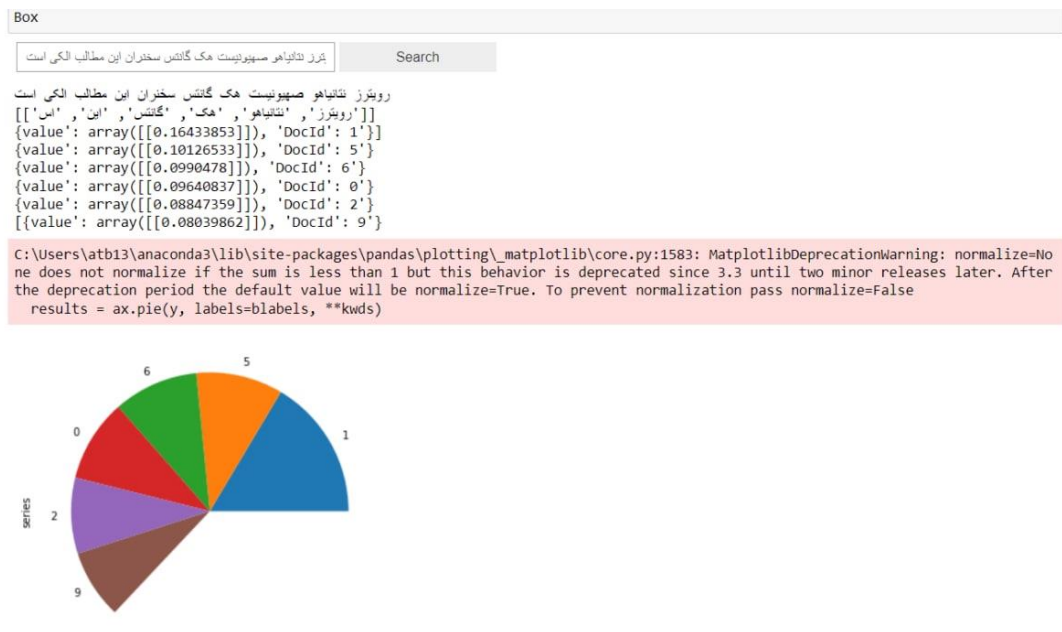
print(np.sort(cosine_sim))
button.on_click(on_button_clicked)
Box

```

واحد مسکونی Search

واحد مسکونی  
[[ 'مسکونی', 'واحد' ]]  
[[ .0 ]]

[[ .0 ]]



شکل 7-2: نتایج پرس و جوی مدل برداری

## 8- منابع

- [1] [https://en.wikipedia.org/wiki/Boolean\\_model\\_of\\_information\\_retrieval](https://en.wikipedia.org/wiki/Boolean_model_of_information_retrieval)
- [2] <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>