گزارش پروژه – مرحله اول

زهرا گنجی 9531802

ساخت شاخص معکوس:

برای ساخت شاخص معکوس بخش های زیر باید پیاده سازی شود:

- واکشی هر خبر از مجموعه داده ورودی و قرار دادن آن در str برای انجام پردازش های مربوطه

```
file_names = ["ir-news-0-2.csv", "ir-news-2-4.csv", "ir-news-4-6.csv", "ir-news-6-8.csv", "ir-news-8-10.csv","ir-news-10-12.csv"]
for l in range(len(file_names)):
    df = pd.read_csv(r"./doc_collection/" + file_names[l])
    contents = pd.DataFrame(df, columns=['content'])
    for i in range(len(contents)):
        str = contents.loc[i, "content"]
        news.append(str)
```

- حالت اول:
  - انجام پیش پردازش روی متن هر خبر

```
def pre_process_1(str,sw):
    str = remove_html_tags(str)
    str = remove_punct(str)
    token_list = tokenize(str)
    token_list = remove_sw(token_list, sw)
    return token_list
```

✓ حذف برچسپ های HTML

```
def remove_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    str = soup.get_text()
```

✓ حذف علایم نگارشی و اعداد

```
def remove_punct(text):
    punctuations = string.punctuation + "،"
    def change(ch):
        if ch in punctuations or ch.isdigit():
            return " "
        else:
            return ch
    no_punct = "".join([change(ch) for ch in text])
    return no_punct
```

✓ استخراج توکن

```
def tokenize(text):
    tokens = re.split('\W+', text)
    return tokens
```

✓ حذف کلمات پرتکرار

```python
def making_sw(sw_dir):
    sw = []
    with open(sw_dir, encoding="utf8") as f:
        lines = f.readlines()
        for w in lines:
            sw.append(re.findall('\S+', w)[0])
    # adding br and empty string to stop words
    sw.append('br')
    sw.append('')
    return sw

def remove_sw(tokens,sw):
    text = [w for w in tokens if w not in sw and len(w)>=2]
    return text
```

- حالت دوم
  - انجام پیش پردازش روی متن هر خبر

```python
def pre_process_2(str,sw):
    str = normalize(str)
    token_list = tokenize(str)
    token_list = stemming(token_list)
    token_list = remove_sw(token_list, sw)
    return token_list
```

✓ نرمال سازی متن

1. جایگزین کردن کاراکتر های عربی با کاراکتر های فارسی و حذف اعراب و حذف انواع نیم فاصله و جایگزینی آن ها با فاصله

```python
def normalize(text):
    # Replace Arabic letters with Persian letters
    text = re.sub("[ۀ]", "ه", text)
    text = re.sub("ى", "ی", text)
    text = re.sub("ك", "ک", text)
    text = re.sub("ؤ", "و", text)
    text = re.sub("إ", "ا", text)
    text = re.sub("أ", "ا", text)
    text = re.sub("ٱ", "ا", text)
    text = re.sub("ة", "ه", text)
    # Replace all types of half-distance with distance
    text = re.sub("\u200c", " ", text)
    text = re.sub("\u200d", " ", text)
    text = re.sub("\xa0", " ", text)
    # Eliminate the vowels
    noise = re.compile("""ّ|ُ|ِ|ٍ|ٌ|َ|ً|ْ""",re.VERBOSE)
    text = re.sub(noise, '', text)
```

2. حذف mention ها به اسامی و url ها و کاراکتر های خاص و اموجی ها

```python
# delete mentions and peoples name
text = re.sub(r'@[A-Za-z0-9_]+', ' ', text)
# delete url
text = re.sub(r'https?://[^ ]+', ' ', text)
# delete url
text = re.sub(r'www.[^ ]+', ' ', text)
# Delete specific characters
text = re.sub(r"[a-zA-Z!$()&@0-9:\\#/|{}<>?~.\"\'_»«;,]", " ", text)
# delete emoji
emoji_pattern = u'([\U00002600-\U000027BF])|([\U0001f300-\U0001f64F])|([\U0001f680-\U0001f6FF])|([\U0001F1E0-\U0001F
text = re.sub(emoji_pattern, ' ', text)
```

3. حذف تگ های html، علایم نگارشی و اعداد

```
# delete html tags
soup = BeautifulSoup(text, "html.parser")
text = soup.get_text(" ")

# remove punctuations marks
punctuations = string.punctuation + "،" + "؛"
def change(ch):
    if ch in punctuations or ch.isdigit():
        return " "
    else:
        return ch
no_punct = "".join([change(ch) for ch in text])
return no_punct
```

✓ استخراج توکن:

```
def tokenize(text):
    common_cw = [
    after_verbs = ['ام', 'ای', 'است', 'ایم', 'اید', 'اند', 'بودم', 'بودی', 'بود']
    before_verbs = ['خواهم', 'خواهی', 'خواهد', 'خواهیم', 'خواهید', 'می', 'خواهند']
    ends = ['ان', 'ات', 'ار', 'اد', 'ام', 'ای', 'ای', 'ای', 'ای', 'ای', 'ای', 'ای', 'ای', 'ای', 'ای', 'ای']
    for cw in common_cw:
        if cw in text:
            str_rep = cw.replace(" ","\u200c")
            text = text.replace(cw,str_rep)
    tokens = text.split()
    for t in range(len(tokens)):
        if tokens[t] in after_verbs:
            tokens[t-1] = tokens[t-1] +"_"+ tokens[t]
            tokens[t-1].replace(" ","\u200c")
            tokens[t] = " "
        if tokens[t] in before_verbs:
            tokens[t + 1] = tokens[t] +"_"+ tokens[t+1]
            tokens[t + 1].replace(" ","\u200c")
            tokens[t] = " "
        if tokens[t] in ends:
            tokens[t - 1] = tokens[t - 1] +"_"+ tokens[t]
            tokens[t-1].replace(" ","\u200c")
            tokens[t] = " "
    new_tokens = []
```

```
for t in range(len(tokens)):
    if tokens[t] != " " and tokens[t] != "":
        new_tokens.append(tokens[t])
return new_tokens
```

در این قسمت باید موارد زیر رعایت شوند:

1. افعال به هر شکلی در ورودی ظاهر شده اند باید یک کلمه در نظر گرفته شوند. برای این منظور دو لیست برای پیشوند ها و پسوند های افعال در نظر گرفته می شود و بررسی می شود اگر در متن به فعلی با پیشوند و پسوند برخوردیم ، فاصله ی بین این پیشوند ها و پسوند ها با فعل را به نیم فاصله تبدیل کنیم و آن ها را با هم یک token در نظر بگیریم.

```
after_verbs = ['ام', 'ای', 'است', 'ایم', 'اید', 'اند', 'بودم', 'بودی', 'بود']
before_verbs = ['خواهم', 'خواهی', 'خواهد', 'خواهیم', 'خواهید', 'می', 'خواهند']
```

2. اگر کلمه ای با علامت جمع و پسوند دیگری به کار رفته بود، فاصله ی بین کلمه و پسوند را به نیم فاصله تبدیل می کنیم و کلمه با پسوند خود یک کلمه در نظر گرفته می شود. برای این منظور لیستی از پسوند های کلمه تهیه کردیم تا اگر کلمه ای قبل از این علایم واقع شده بود، با این علایم یک کلمه در نظر گرفته شود.

```
ends = ['ات' ,'ان' ,'ترین' ,'تر' ,'م' ,'ت' ,'ش' ,'ی' ,'ئی' ,'ها'
'ا' ,'پنیر' ,'بخش' ,'مند' ,'گیر']
```

3. عبارت های ترکیبی پرکاربرد که غالبا در کنار یک دیگر ظاهر می شوند، یک کلمه در نظر گرفته شوند و فاصله ی بین بخش های مختلف این عبارات به نیم فاصله تبدیل شوند. برای این منظور یک لیست از این عبارات تهیه می شود تا اگر در متن ظاهر شدند یک کلمه در نظر گرفته شوند.

```
common_cw = ['رسمی غیر' ,'قانونی غیر' ,'تفاوت بی' ,'سپرده سر' ,'عنوان به' ,'خاطر به' ,'به
به' ,'پیش از بیش' ,'چند هر' ,'چه گر' ,'ای ایده' ,'نظیر بی' ,'انگیز شگفت' ,'جمهور رئیس' ,'ویژه
فیما' ,'چه چنان' ,'این بر بنا' ,'حال ای علی' ,'ذلک مع' ,'انجام سر' ,'آنچه هر' ,'نحایت بی' ,'گم در سر' ,'راستی
بین']
```

.

✓ ریشه یابی کلمات:

برای ریشه یابی کلمات از کتاب خانه ی آماده ی parsivar استفاده شده است.

```
def stemming(tokens,stemming_list):
    my_stemmer = FindStems()
    new_tokens = []
    new = ""
    term = " "
    for w in tokens:
        term = w
        new = my_stemmer.convert_to_stem(w)
        # deal with
```

برای کلماتی که دو ریشه برای آن ها توسط این کتاب خانه تولید می شود، ریشه ای به عنوان خروجی در نظر گرفته شده است که حروف آن در خود کلمه دیده شود. برای مثال ریشه های کلمه ی "رفته است" ، "رو" و "رفت" می شوند که چون حروف کلمه ی "رفت" در خود کلمه آمده است، رفت به عنوان ریشه ی نهایی در نظر گرفته شده است.

```
    # dealing with 2 answers,and select better one
    if "&" in new:
        index = new.find('&')
        if new[index+1:] in w:
            new = new[index+1:]
        else:
            new = new[:index]
```

این کتاب خانه مصدر فعل ها را به ریشه ی آن ها تبدیل نمی کرد که این مورد را نیز دستی هندل کرده ام.

```
if new == w:
    if w[-1] == 'ن':
        w = w[:-1] + " " + w[-1]
    new = my_stemmer.convert_to_stem(w)
    if new[-1] == 'ن' and new[-2] == " ":
        new = new.replace(" ن","")
```

لیست کلماتی که به هر کدام از کلمات زیر نگاشت می شود: (به دلیل زیاد شدن زمان اجرای برنامه فقط برنامه را برای 200 خبر اول در هر کدام از فایل های csv ران کردم پس کلا 1200 خبر خوانده و پردازش کرده ام، پس این لیست ها مربوط به پیش پردازش این 1200 خبر هستند)

گفت: ['گفتند', 'گفته بود', 'گفت ان', 'می گفت', 'گفته است', 'گفته ام', 'گفتم', 'خواهد گفت', 'گفتن', 'گفتید', 'گفته اند', 'خواهند گفت', 'می گفتند', 'نگفته اند', 'می گفتم', 'گفتیم']
گو: ['می گوید', 'بگویم', 'گوید', 'بگوید', 'بگو', 'می گویم', 'میگوید', 'گوانگ', 'گوترش', 'گویم', 'گوهای', 'نگوید']
رود: ['رودی', 'رودها']
رو: ['برویم', 'بروند', 'می رود', 'برود', 'می روند', 'می روی', 'نرود', 'روین', 'بروید', 'میرود', 'فرارو', 'بروم', 'روتر']
خواه: ['می خواهد', 'نخواهد', 'بخواهند', 'بخواهد', 'بخواهیم', 'نخواهم', 'می خواهیم', 'خواهی ها', 'می خواهند', 'نخواهیم', 'می خواهید', 'می خواهم', 'نخواهند', 'نخواهی', 'نخواهید', 'بخواهید', 'خواهی های']
سپاس: []
هنر: ['هنرهای']
شریف: ['شریف ترین', 'شریفی']
دوست: ['دوستان', 'دوستش', 'دوستانش']
یاد: ['یاد ان']
توان: ['بتوانند', 'نتوانستیم', 'توانستیم', 'می تواند', 'می توانستیم', 'می توانند', 'می توانست', 'توانست', 'نتوانند', 'می توانید', 'توانند', 'نتوانستند', 'بتوانیم', 'می توانیم', 'نتوانست', 'توانشان', 'نتوانیم', 'بتوان', 'می توانم', 'می توانستند', 'توانستم', 'نتوانسته بود', 'نتوانسته اند', 'توانیم', 'توانستند', 'توانسته اند', 'توانسته ایم', 'میتوانید', 'توانید', 'توانستید', 'توانمان', 'نتوانسته است', 'نتوانسته ایم', 'توانسته بود', 'بتوانید', 'توانم', 'بازتوانی', 'نتوانستم']
شنو: ['بشنود', 'بشنویم', 'بشنوند', 'بشنوید', 'می شنوید']
کرد: ['کرده است', 'کردن', 'کردند', 'کردیم', 'خواهد کرد', 'خواهیم کرد', 'کرده اند', 'نکرد', 'کردها', 'کردید', 'کرده بود', 'کرده ام', 'نکرده ام', 'می کرد', 'نکرده اند', 'کرده ای', 'کردم', 'می کردند', 'نکردند', 'کرده ایم', 'می کردی', 'می کردم', 'خواهند کرد', 'خواهم کرد', 'نکرده است', 'کرد ها', 'نکردیم', 'کرده بودم', 'می کردید', 'کردهای', 'می کردیم', 'کرده اید', 'خواهدکرد', 'نکردم', 'نکرده ایم']
ساز: ['سازی', 'می سازد', 'سازد', 'سازان', 'بسازیم', 'بسازد', 'سازند', 'سازها', 'سازهای', 'بسازند', 'سازیم', 'می سازند']
دان: ['دانند', 'می داند', 'بدانند', 'دانستند', 'می دانند', 'بدانید', 'می دانم', 'بدانیم', 'دانیم', 'دانستم', 'می دانیم', 'می دانید', 'می دانی', 'دانستن', 'دانم', 'ندانست', 'می دانست', 'ندانسته اند', 'دانسته بود', 'دانان', 'می دانستم', 'می دانستیم', 'بدانم']

- ساخت شاخص معکوس

برای ساخت شاخص معکوس یک دیکشنریterm_termid_match در نظر می گیریم که هر کلمه را به id متناظر به آن کلمه نگاشت می دهد تا در شاخص معکوس به جای خود کلمه از id آن استفاده کنیم. همچنین یک لیست از جفت های termed و docid کلمات ایجاد می کنیم. شمارنده های doc_id

و term_id را نیز بعد از خواندن هر خبر از مجموعه خبر ها و بعد از قرار دادن هر token از متن خبر در دیکشنری یکی اضافه می کنیم.
بعد از پیش پردازش و استخراج لیست token هر خبر، روی این لیست for می زنیم.
به ازای هر token:

- اگر در دیکشنری term_termid_match بود، یعنی قبلا آن کلمه را در دیکشنری داشته ایم، پس دیگر آن را به term_termid_match اضافه نمی کنیم و فقط termid مرتبط با آن را از این دیکشنری می خوانیم و در لیست termid_docid_pairs نگاه می کنیم اگر جفت این termid و docid را داشتیم یعنی یک بار دیگر این کلمه در این سند(خبر) تکرار شده بوده است پس دیگر لازم نیست این جفت را به termid_docid_pairs اضافه کنیم چون در شاخص معکوس برخلاف شاخص positional تکرار های مختلف کلمه در یک سند را در نظر نمی گیریم. اگر این جفت termid و docid در لیست نبودند پس آن را به لیست اضافه می کنیم.
- اگر در دیکشنری term_termid_match نبود، به این دیکشنری آن را اضافه می کنیم و term_id را به عنوان id آن در نظر می گیریم. سپس term_id و doc_id را به لیست term_termid_match اضافه می کنیم.

```
for token in token_list:
    total_token_number = total_token_number + 1
    # if we had term in dictionary
    if token in term_termid_match.keys():
        pair = []
        pair.append(term_termid_match[token])
        pair.append(doc_id)
        term_frequency[token] = term_frequency[token] + 1
        # if we dont had this termid docid pair exactly in our list we dont append pair in list
        if pair not in termid_docid_pairs:
            termid_docid_pairs.append(pair)

    # if we didnt have that term in our dictionary
    else:
        pair = []
        term_termid_match[token] = term_id
        pair.append(term_id)
        pair.append(doc_id)
        termid_docid_pairs.append(pair)
        term_id = term_id + 1
        term_frequency[token] = 1

    T_M_dict[total_token_number] = len(term_termid_match)
if i == 500:
    break
```

بعد از اضافه کردن تمام جفت term_id,doc_id ها به لیست،این لیست که براساس docid مرتب شده است را براساس termid مرتب می کنیم.

```
# sort by terms
termid_docid_pairs.sort(key=lambda x: x[0])
# print(termid_docid_pairs)
```

بعد از مرتب سازی جفت ها براساس termid، شاخص معکوس را می سازیم. به این صورت که شاخص معکوس را در یک دیکشنری با نام inverted_index تعریف می کنیم که key ها term_id های مجزا و مقدارهر key، لیست docid هایی می شود که کلمه در آن ها ظاهر شده است. در واقع کلید ها دیکشنری و مقادیر هر کلید postings list ها را تشکیل می دهند.

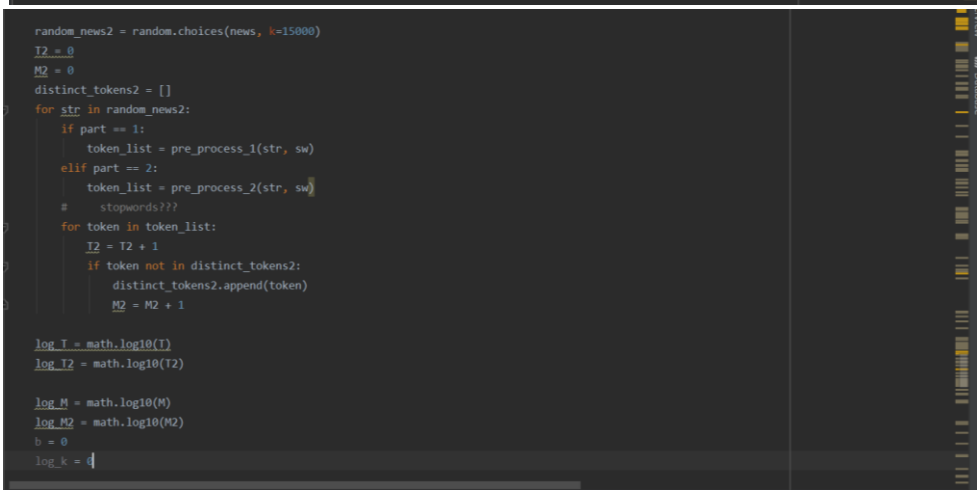
به ازای هر جفت termid و docid از لیست جفت ها، چک میکنیم اگر termid در از قبل در کلید های inverted_index وجود داشت یعنی از قبل در دیکشنری وارد شده است پس فقط docid را به لیست مقدارهایش اضافه می کنیم و اگر از قبل وجود نداشت، termid و docid را به دیکشنری اضافه می کنیم.

```
# inverted index construction
inverted_index = {}
for pair in termid_docid_pairs:
    if pair[0] in inverted_index.keys():
        inverted_index[pair[0]].append(pair[1])
    else:
        inverted_index[pair[0]] = [pair[1]]
if part == 1:
    return inverted_index,T_M_dict,news,term_frequency
else:
    return inverted_index, T_M_dict, news, term_frequency,stemming_list
```

- بررسی قانون heaps

در زمان پیش پردازش هر خبر، آن را به یک لیست به نام news اضافه کرده بودیم. اکنون از این لیست که شامل تمامی اسناد است، دو زیر مجموعه ی 5000 و 15000 تایی را به صورت تصادفی انتخاب می کنیم و بعد از انجام پیش پردازش های متناسب با شاخص معکوس مورد نظر روی خبر ها، تعداد کلمات مجزا(M) و تعداد کل کلمات (T) این دو مجموعه را حساب می کنیم و از آن ها logمیگیریم تا با استفاده از 4 مقدار logM,logM2,logT.logT2 ثابت های b,logk (پارامتر های قانون heaps) را در خط logM = logk + b logT را بدست بیاوریم.

```
random_news = random.choices(news, k=5000)
T = 0
M = 0
distinct_tokens = []
for str in random_news:
    if part == 1:
        token_list = pre_process_1(str,sw)
    elif part == 2:
        token_list = pre_process_2(str,sw)
    #     stopwords???
    for token in token_list:
        T = T + 1
        if token not in distinct_tokens:
            distinct_tokens.append(token)
            M = M + 1
```



```
random_news2 = random.choices(news, k=15000)
T2 = 0
M2 = 0
distinct_tokens2 = []
for str in random_news2:
    if part == 1:
        token_list = pre_process_1(str, sw)
    elif part == 2:
        token_list = pre_process_2(str, sw)
    #     stopwords???
    for token in token_list:
        T2 = T2 + 1
        if token not in distinct_tokens2:
            distinct_tokens2.append(token)
            M2 = M2 + 1

log_T = math.log10(T)
log_T2 = math.log10(T2)

log_M = math.log10(M)
log_M2 = math.log10(M2)
b = 0
log_k = 0
```

```
b = (log_M - log_M2) / (log_T - log_T2)
log_k = log_M - (b * log_T)
k = 10 ** log_k
```

سپس از این خط برای پیش بینی اندازه ی دیکشنری (تعداد ترم های مجزا) به ازای تعداد ترم هایی که در collection داریم ، استفاده می کنیم و نتیجه ی این پیش بینی را به ازای اضافه کردن هر term و محاسبه ی تعداد کل ترم ها تا آن کلمه ، با یک منحنی در نمودار نشان می دهیم. پس این منحنی در نمودار اندازه ی دیکشنری را در حالت پیش بینی شده و با خطی که بدست آوردیم نشان می دهد.

و یک منحنی دیگر نیز در نمودار داریم که مقدار واقعی اندازه ی دیکشنری یا تعداد ترم های مجزا را با اضافه کردن ترم ها نشان می دهد.

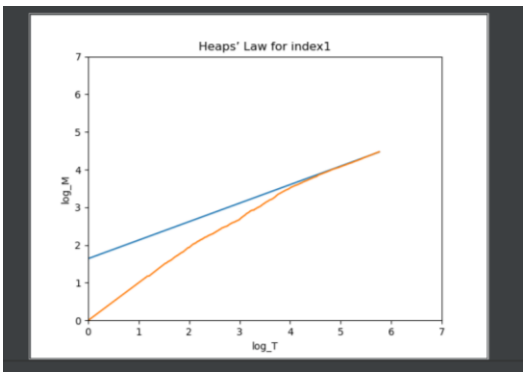
پس می توانیم در این نمودار این دو منحنی مقادیر پیش بینی شده و واقعی را با یک دیگر مقایسه کنیم.

تابع heaps_law را برای هر دو شاخص معکوس با توجه به پیش پردازش های متفاوت آن ها، اجرا می کنیم و دو نمودار برای هر یک بدست می آوریم. برای شاخص معکوس اول part = 1 در ورودی قرار می دهیم و برای شاخص معکوس دوم part = 2 در ورودی قرار می دهیم.

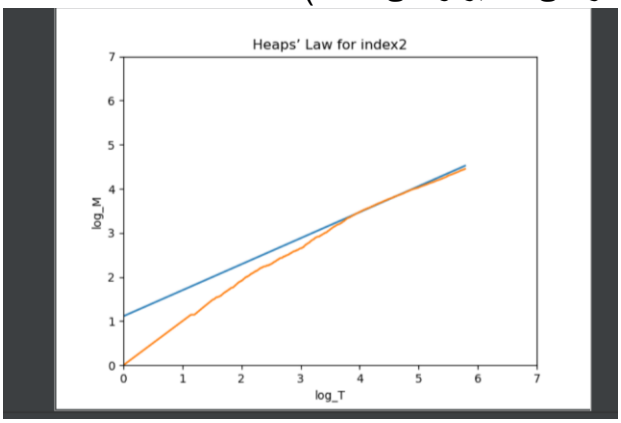
```
values_T = T_M_dict.keys()
values_T = list(values_T)
log_T = []
log_M = []
pred_log_M = []
for t in values_T:
    pred_log_M.append(log_k + (b * math.log10(t)))
    log_M.append(math.log10(T_M_dict[t]))
    log_T.append(math.log10(t))

plt.xlabel("log_T")
plt.ylabel("log_M")
plt.xlim([0, 7])
plt.ylim([0, 7])
plt.plot(log_T, pred_log_M)
plt.plot(log_T, log_M)
if part == 1:
    plt.title("Heaps' Law for index1")
    plt.savefig('Heaps_law1.png')
elif part == 2:
    plt.title("Heaps' Law for index2")
    plt.savefig('Heaps_law2.png')
plt.clf()
```

به دلیل زیاد شدن زمان اجرای برنامه فقط برنامه را برای 500 خبر اول در هر کدام از فایل های csv ران کردم و در نهایت به جای 50000 خبر کلا 3000 خبر را index کرده ام، پس این نمودار ها مربوط به این 3000 خبر هستند

- نمودار قانون heaps برای index در حالت اول: (منحنی آبی رنگ مقادیر پیش بینی شده و منحنی نارنجی مقادیر واقعی هستند)



- نمودار قانون heaps برای index در حالت دوم: (منحنی آبی رنگ مقادیر پیش بینی شده و منحنی نارنجی مقادیر واقعی هستند)



مقایسه ی نمودار های heaps در دو حالت ساخت شاخص:

خطی که برای پیش بینی تعداد کلمات مجزا در صورت داشتن تعداد کل کلمات مجموعه اسناد، بر روی داده ها fit می شود، در نمودار شاخص اول مقادیری که برای کلمات مجزا یعنی تعداد کلمات دیکشنری پیش بینی می کند بیش تر از نمودار شاخص دوم است. زیرا با داشتن تعداد کل یکسان کلمات در دو نمودار، تعداد کلمات متمایز در شاخص اول بیش تر از شاخص دوم است.این به دلیل این است که نرمال کردن اشکال مختلف کلمات و تبدیل آن ها به یک شکل و به یک کلمه و همچنین در ریشه یابی کلمات درپیش پردازش شاخص دوم ، کلمات با اشکال مختلف به یک شکل که همان ریشه ی آن ها است تبدیل می شوند، پس تعداد کلمات مجزا کاهش می یابد زیرا کلمات مختلف به یک شکل تبدیل شده اند.

- بررسی قانون zipf

تعداد تکرار هر کلمه را در بررسی هر خبر در create_index شمرده ایم و در دیکشنری term_frequency ذخیره کرده ایم. کلمه ی با بیش ترین تکرار، در دیکشنری value ی بیشینه دارد. پس ماکزیمم value در دیکشنری که تعداد تکرار اولین کلمه ی پرتکرار دیکشنری است را مساوی k در نظر می گیریم. به همین ترتیب تعداد تکرار واقعی دومین کلمه ی پر تکرار را از دیکشنری بدست می آوریم و به همین تریتیب تا انتها و در هر مرحله log rank و log cf کلمه را در لیست های log_rank و log_cf_real اضافه می کنیم. و منحنی مقادیر واقعی تعداد تکرار را در نمودار به ازای هر کلمه نشان می دهیم.

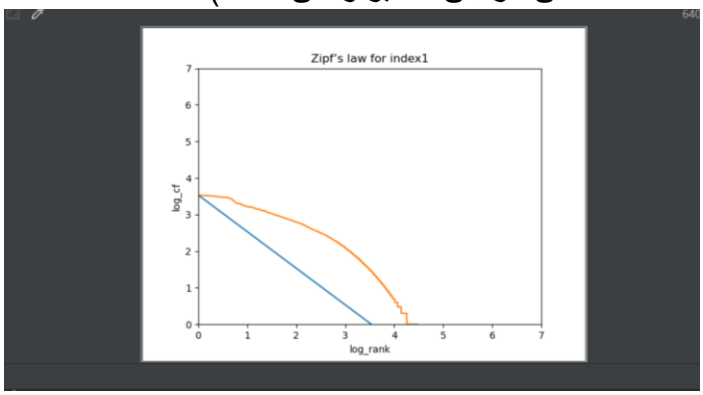
```
cf_real = []
log_cf_real = []
cf_real.append(max(values))
del values[values.index(max(values))]
while len(values) != 0:
    max_freq= max(values)
    cf_real.append(max_freq)
    del values[values.index(max_freq)]
for cf in cf_real:
    log_cf_real.append(math.log10(cf))
```

برای پیش بینی تعداد تکرار کلمات با قانون zipfهم، بعد از بدست آوردن k، logcf که مقدار پیش بینی شده ی تعداد تکرار کلمات است را با فرمول logk – log rank بدست می آوریم. و در لیست log_cf قرار می دهیم. و و منحنی مقادیر پیش بینی شده ی تعداد تکرار را در نمودار به ازای هر کلمه نشان می دهیم.

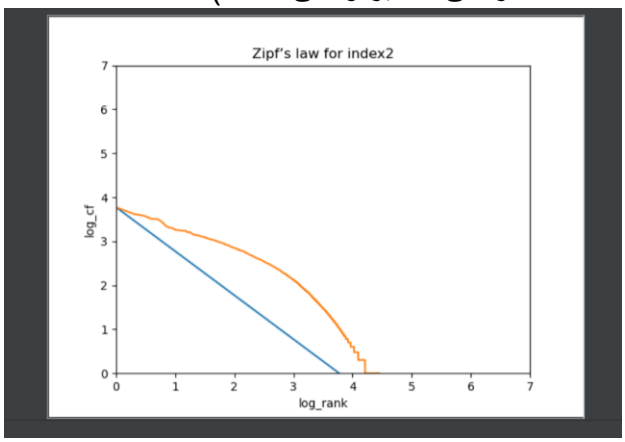
```
def zipf_law(term_frequency):
    values = term_frequency.values()
    values = list(values)
    log_cf = []
    k = max(values)
    log_cf.append(math.log10(k))
    log_k = math.log10(k)
    log_rank = [math.log10(1)]
    number_of_tokens = len(term_frequency)
    for rank in range(1,number_of_tokens):
        log_cf.append(log_k - math.log10(rank+1))
        log_rank.append(math.log10(rank+1))
```

به دلیل زیاد شدن زمان اجرای برنامه فقط برنامه را برای 500 خبر اول در هر کدام از فایل های csv ران کردم و در نهایت به جای 50000 خبر کلا 3000 خبر را index کرده ام، پس این نمودار ها مربوط به این این 3000 خبر هستند

- نمودار قانون zipf برای index در حالت اول: (منحنی آبی رنگ مقادیر پیش بینی شده و منحنی نارنجی مقادیر واقعی هستند)



- نمودار قانون zipf برای index در حالت دوم:(منحنی آبی رنگ مقادیر پیش بینی شده و منحنی نارنجی مقادیر واقعی هستند)



مقایسه ی نمودار های zipf در دو حالت ساخت شاخص:

با مشاهده ی دو نمودار متوجه می شویم تعداد پر تکرار ترین کلمه ی دیکشنری در شاخص دوم بیش تر است. و در کل مشاهده می شود کلمات با رنک برابر، در شاخص دوم تعداد تکرار بیش تری از تعداد تکرارشان در شاخص اول دارند. این می تواند به دلیل پیش پردازش های متفاوت این دو شاخص باشد. در شاخص دوم، با نرمال کردن اشکال مختلف کلمات و تبدیل آن ها به یک شکل و به یک کلمه، تعداد تکرار کلمه بالا می رود. همچنین در ریشه یابی کلمات در ساخت شاخص دوم، کلمات با اشکال مختلف به یک شکل که همان ریشه ی آن ها است تبدیل می شوند، پس تعداد تکرار کلمه ی ریشه زیاد می شود.