به نام خدا

گزارش تمرین سوم بازیابی پیشرفته اطلاعات

سامانه بازیابی اطلاعات ساده بر اساس مقالات علمی

گروه ۳۰

پارسا حسینی

درسا مجدی

امیررضا باقری

چکیده روند پروژه

در این تمرین هدف آن است که یک سامانه بازیابی اطلاعات ساده بر اساس مقالات علمی پیاده‌سازی شود. در بخش اول اطلاعات تعدادی مقاله از سایت Semantic Scholar جمع‌آوری شد. در مرحله بعدی این اطلاعات تمیز و پیش پردازش اولیه بر روی هر بخش آن اعمال شد. در گام سوم، یک سامانه جستجوی ساده مبتنی بر ۴ روش بازیابی پیاده‌سازی شد. نهایتا، عملکرد پیاده‌سازی را با استفاده از معیار MRR ارزیابی کردیم.

بخش ۱. دریافت داده‌ها

سایت Semantic Scholar یک api در اختیار ما قرار می‌دهد که می‌توانیم با استفاده از آن، مقالات مرتبط با موضوع مورد نظر و اطلاعات مربوط به هر مقاله را دریافت کنیم. کد مربوط به جمع‌آوری داده در فایل Crawl_Semantic_Scholar.ipynb قابل مشاهده است.

در تابع request_papers_by_keywords به تعداد دفعات درخواست (no_requests)، هر دفعه ۱۰۰ مقاله مرتبط با کلمات کلیدی (keywords)، را می‌دهد که اطلاعات آن شامل مواردی است که در متغیر fields قرار گرفته است. سپس این اطلاعات در یک دیتافریم پانداس و سپس در یک فایل csv ذخیره شد تا بتوان در مراحل بعدی از آن استفاده کرد و نیازی به جمع‌آوری دوباره داده نباشد.

```python
# Request to SemanticScholar API
def request_papers_by_keywords(no_requests, keywords, fields):
    papers = []
    for i in tqdm(range(no_requests)):
        response = requests.get(f'https://api.semanticscholar.org/graph/v1/paper/search?query={keywords}&offset={100*i}&limit=100&fields={fields}')
        js = response.json()
        if 'data' in js:
            papers.append(js['data'])
        else:
            print()
            print(f'An error occured in step {i}')
        # sleep(3.1)
    return list(chain.from_iterable(papers))
```

شکل 1. تابع request_papers_by_keywords

```
[ ]  # no_requests must be less than 100. Otherwise uncomment the sleep in above cell.
     no_requests = 99 # Each request returns 100 papers
     keywords = 'Attention+Transformer' # Separated by +
     fields = 'paperId,url,title,abstract,year,authors,fieldsOfStudy,citationCount,referenceCount' # Separated by ,
     data = request_papers_by_keywords(no_requests, keywords, fields)

     100%|██████████| 99/99 [00:38<00:00,  2.55it/s]

[ ]  len(data)

     9899
```

شکل 2. جمع آوری مقالات مرتبط با کلمات کلیدی Attention و Transformer

بخش ۲. پیش‌پردازش اولیه متن

در این بخش ابتدا داده‌های جمع‌آوری شده در گام قبلی، در یک دیتافریم ذخیره شده و سپس با استفاده از کلاس Preprocess، پیش‌پردازش مورد نیاز برای هر بخش را روی آن اعمال می‌کنیم.

دو روش متفاوت برای پیش‌پردازش به کار گرفته شد:

1. استفاده از کتابخانه spacy
2. استفاده از امکاناتی که کتابخانه nltk در اختیار ما قرار می‌دهد.

هنگام استفاده از کتابخانه spacy، ابتدا یک مدل بر روی متن اعمال می‌شود و آن را به صورت مجموعه‌ای از spacy.token ها در می‌آورد. در این روش می‌توان به ویژگی‌های هر spacy.token دسترسی داشت از جمله اینکه stopword است یا نه (token.is_stop) یا آنکه جزو علائم نقطه‌گذاری است یا نه (token.is_punct) و ... . بعد از آنکه مدل بر روی متن اعمال شد، با استفاده از تابع normalize_sentence و مشخص کردن نوع پیش پردازش توسط متغیرهای بولین stopword_removal و punctuation_removal و lower_case و lemmatize، متن normalized شده را دریافت می‌کنیم.

توابع پیش‌پردازش با استفاده از spacy را در شکل ۳ مشاهده می‌کنید:

```
def spacy_model(self, raw_text):
    return self.normalize_sentence(self.nlp(raw_text))

def normalize_sentence(self, tokenized_sent, stopword_removal=True, punctuation_removal=True,
                       lower_case=False, lemmatize=True, minimum_length=2):
    normalized_sent = tokenized_sent

    if(stopword_removal):
        normalized_sent = [token for token in normalized_sent
                           if not(token.is_stop)]
    if (punctuation_removal):
        normalized_sent = [token for token in normalized_sent
                           if not(token.is_punct)]
    if (lower_case):
        normalized_sent = [token.text.lower() for token in normalized_sent]

    if (minimum_length > 1):
        normalized_sent = [token for token in normalized_sent
                           if len(token) > minimum_length]
    if (lemmatize):
        normalized_sent = [token.lemma_ for token in normalized_sent]

    return normalized_sent
```

شکل 3. پیش‌پردازش با استفاده از کتابخانه spacy

برای استفاده از امکانات کتابخانه nltk از جمله دریافت stopwordها، lemmatizer و stemmer توابعی جداگانه برای هر پیش‌پردازش تعریف می‌کنیم و بسته به نوع متن مورد نظر هر کدام از این توابع روی متن

اعمال می‌شوند. در تابع run یک پیش‌پردازش جامع و در تابع simple یک پیش‌پردازش ساده مناسب برای بخش نویسنده پیاده‌سازی شده‌اند.

```
def remove_punctuation(self, text):
    return "".join([i for i in text if i not in string.punctuation])

def tokenize(self, text):
    return word_tokenize(text)

def remove_stopwords(self, tokens, minimum_length=2):
    return [t for t in tokens if t not in self.stopwords and len(t) >= minimum_length]

def lower_case(self, tokens):
    return [t.lower() for t in tokens]

def lemmatize(self, tokens):
    return [self.wordnet_lemmatizer.lemmatize(t) for t in tokens]

def stem(self, tokens):
    return [self.sno_stemmer.stem(t) for t in tokens]

def correct_typo(self, tokens):
    return [self.spell.correction(t) if len(self.spell.unknown([t])) > 0 else t for t in tokens]
```

شکل 4. توابع جداگانه برای هر مرحله از پیش‌پردازش

```
def run(self, raw_text, correction=False, stem=False):

    text = self.remove_punctuation(raw_text)

    tokens = self.tokenize(text)

    tokens = self.remove_stopwords(tokens)

    tokens = self.lower_case(tokens)

    if correction:
        tokens = self.correct_typo(tokens)

    if stem:
        tokens = self.stem(tokens)

    tokens = self.lemmatize(tokens)

    return tokens
```

شکل 5. تابع run برای پیش‌پردازش جامع

```
def simple(self, raw_text):
    text = self.remove_punctuation(raw_text)
    tokens = self.tokenize(text)
    tokens = self.lower_case(tokens)
    return tokens
```

شکل 6. تابع simple برای پیش‌پردازش ساده مناسب برای بخش نویسنده

بخش ۳. دریافت مقاله‌های مرتبط با کوئری

۳.۱ – روش بازیابی Boolean برای بخش‌های نویسنده و عنوان

در روش boolean، هدف آن است که مشخص کنیم که کلمه مورد نظر در چه مقالاتی آمده است (1) و در چه مقالاتی نیامده است (0). یک طریقه پیاده‌سازی آن، استفاده از inverted_index است به این صورت که یک دیکشنری تعریف می‌کنیم که در آن کلیدها، کلمات بخش مورد نظر هستند و مقادیر متناظر هر کلید، یک لیست از شماره مقالاتی است که کلمه در آنجا آمده است. بنابراین می‌توانیم به ازای هر کوئری، مشخص کنیم که هر کلمه آن در چه مقالاتی آمده‌اند و آن مقالات را به ترتیب میزان تطابق با کوئری خروجی دهیم.

برای بخش نویسنده و بخش عنوان، یک دیکشنری مجزا تعریف کرده و به همان صورت گفته شده آنها را مقدار می‌دهیم. این دو دیکشنری را در یک دیکشنری به نام inverted_indices ذخیره کرده که بعدا می‌توان با توجه به نوع کوئری، به دیکشنری بخش نویسنده (author) یا بخش عنوان (title) دسترسی پیدا کرد.

```
[ ]  def get_inverted_index_authors(df):
         doc_authors_list = [ast.literal_eval(author) for author in df['authors']]
         doc_author_names = [[preprocessor.simple(author['name']) for author in authors_list] for authors_list in doc_authors_list]
         doc_author_dict = defaultdict(list)
         for i, doc_authors in enumerate(doc_author_names):
             for author in doc_authors:
                 for token in author.split():
                     doc_author_dict[token].append(i)
                     doc_author_dict[token] = list(dict.fromkeys(doc_author_dict[token]))
         return doc_author_dict
```

شکل 7. تابع get_inverted_index_author برای دریافت دیکشنری inverted_index مربوط به بخش نویسنده

```
def get_inverted_index_titles(df):
    doc_title_words_list = df['clean_title'].str.split().tolist()
    doc_title_dict = defaultdict(list)
    for i, doc_titles in enumerate(doc_title_words_list):
        for token in doc_titles:
            doc_title_dict[token].append(i)
            doc_title_dict[token] = list(dict.fromkeys(doc_title_dict[token]))
    return doc_title_dict
```

شکل 8. تابع get_inverted_index_titles برای دریافت دیکشنری inverted_index مربوط به بخش عنوان

```
def get_inverted_indices(df):
    doc_author_dict = get_inverted_index_authors(df)
    doc_title_dict = get_inverted_index_titles(df)
    inverted_indices = {'author': doc_author_dict, 'title': doc_title_dict}
    return inverted_indices

inverted_indices = get_inverted_indices(df)
print(inverted_indices)
```

شکل 9. تابع get_inverted_indices برای ذخیره دیکشنری‌های مربوط به نویسنده و عنوان در یک دیکشنری کلی برای استفاده در بخش query

برای هر کوئری، با توجه به اینکه مربوط به چه بخشی است (عنوان یا نویسنده) ابتدا پیش‌پردازش متناسب با آن روی کوئری اعمال می‌شود و سپس اندیس k مقاله اولی که بیشترین میزان تطابق را با کوئری از لحاظ تعداد کلمات در بخش مورد نظر داشته باشند، خروجی داده می‌شوند.

```
[ ]  def bool_query(query, section, get_inverted_indices=get_inverted_indices, df=df, k=10):
         if section == 'title':
             query = preprocessor.run(query)
         elif section == 'author':
             query = preprocessor.simple(query)
         doc_list = []
         for word in query.split():
             doc_list += inverted_indices[section][word]
         doc_list = sorted(doc_list, key=doc_list.count, reverse=True)
         doc_list = list(dict.fromkeys(doc_list))
         return doc_list[:k]
```

شکل 10. تابع bool_query که با توجه به بخش مورد سوال (query) تعداد k نتیجه اول را بر می‌گرداند.

۳.۲ – بازیابی مبتنی بر tf-idf برای بخش‌های عنوان و چکیده

ابتدا نیاز است تا برای هر کلمه تعیین کنیم که در چند سند (Document) آمده است و همچنین در هر سند، هر کلمه چند بار تکرار شده است. در تابع df_count، تعداد سندهایی که هر کلمه در آنها آمده است در دیکشنری word_dict و تعداد تکرار هر کلمه در هر سند، در لیست docs_dict ذخیره شده و خروجی داده می‌شوند. در دیکشنری word_dict کلیدها کلمات هستند و مقادیر متناظرش تعداد سندهایی است که کلمه در آنها آمده است. docs_dict لیستی از دیکشنری ها است که در آن برای هر سند یک دیکشنری ایجاد شده است که در آن کلیدها کلمات موجود در آن سند و مقادیر متناظر هر کلید، تعداد تکرار آن کلمه در سند است.

```
[ ]  def df_count(section):
       docs = select_doc_type(section)
       all_vocab = list(set((" ".join(docs)).split()))
       N = len(docs)
       docs_dict = []
       for i in range(N):
         docs_dict.append(Counter(docs[i].split()))
       word_dict = dict((word,0) for word in all_vocab)
       for docs in docs_dict:
         for key in docs.keys():
           word_dict[key] += 1
       return word_dict, docs_dict
```

شکل 11. تابع df_count برای تشکیل دیکشنری‌های word_dict و docs_dict

در تابع tf_idf پس از آنکه پیش‌پردازش بر روی query اعمال شد و کلمات آن جدا شدند، به ازای همه سندها و به ازای هر کلمه در query، فرکانس رخداد آن کلمه در سند از docs_dict گرفته می‌شود و با توجه به فرمول زیر امتیاز محاسبه می‌شود:

$$score(t,D) = \frac{\#(t,D)}{\#Number\ of\ words\ in\ doc[i]} \times \log \frac{N}{\sum_{D:t\in D} 1}$$

سپس امتیاز همه کلمات در کوئری با یکدیگر جمع شده و به عنوان امتیاز نهایی هر سند با توجه به میزان تطابق با کوئری، در لیست results ذخیره می‌شود. در نهایت نتایج به ترتیب امتیاز مرتب می‌شوند و k سند با بیش‌ترین ارتباط خروجی داده می‌شوند.

```
[ ]  def tf_idf(query, k, word_dict, docs_dict, section):
       docs = select_doc_type(section)
       prep = Preprocess()
       query = prep.run(query)
       query_terms = query.split()
       N = len(docs_dict)
       results = []
       for i in range(N):
         score = 0
         for term in query_terms:
           freq = docs_dict[i].get(term)
           if freq == None:
             continue
           score += (freq/sum(docs_dict[i].values()))*math.log(N/word_dict[term])
         results.append((score,i))
       results.sort(key = lambda x: x[0], reverse = True)
       indexs = [x[1] for x in results]
       return indexs[:k]
```

شکل 12. تابع tf_idf که امتیاز کوئری را برای هر مقاله محاسبه کرده و تعداد k مقاله با بیشترین امتیاز را خروجی می‌دهد

۳.۳ – بازیابی مبتنی بر transformer برای بخش چکیده

در این روش با استفاده از یک مدل transformer آماده، بردارهای embedding مربوط به هر متن و بردار embedding مربوط به کوئری محاسبه شده، سپس میزان شباهت این بردارها با استفاده از cosine similarity محاسبه شده و نهایتا k مقاله‌ای که بیشتری شباهت را داشتند بازگردانده می‌شوند.

Transformer Retrieval

```
[ ]  !pip install sentence-transformers
```

```
[ ]  from sklearn.metrics.pairwise import cosine_similarity
     from sentence_transformers import SentenceTransformer, util
     device = 'cuda' # cpu / cuda
```

```
[ ]  model = SentenceTransformer('multi-qa-MiniLM-L6-cos-v1', device=device)
```

شکل 13. دانلود مدل transformer آماده

```
[ ]  docs = select_doc_type('abstract')
     embeddings = model.encode(docs.tolist(), show_progress_bar=True, device=device)
```

```
[ ]  preprocessor = Preprocess()
     embeddings.shape
```

شکل 14. اعمال مدل بر روی متن چکیده

```
[ ]  def most_similar(query, embeddings=embeddings, k=10):
         query = preprocessor.run(query)
         query_embedding = model.encode(query, device=device)
         cosine_scores = util.dot_score(query_embedding, embeddings).detach().cpu().numpy()[0]
         similar_ix = np.argsort(cosine_scores)[::-1][:k]
         return similar_ix, cosine_scores
```

```
[ ]  def run_transformer(query):
         start_time = time.time()
         print (f'Query: {query}')
         indx, scores = most_similar(query)
         show(indx, scores=scores)
         print()
         print(f'Execution time: {time.time()-start_time}')
```

شکل 15. محاسبه میزان شباهت بردارهای embedding کوئری و متن و خروجی دادن k نتیجه مرتبط

۳.۴ – بازیابی مبتنی بر میانگین وزن‌دار (برحسب tf-idf) بردارهای تعبیه برای بخش چکیده

در این روش، از مدل fasttext برای تبدیل هر کلمه به بردار بهره می‌گیریم، سپس با استفاده از مقادیر tf-idf متناظر با کلمه به عنوان وزن و بردارهای حاصل از مدل fasttext، یک بردار embedding برای متن و یک بردار embedding نیز برای کوئری تهیه می‌کنیم. میزان شباهت این بردارها را با cosine similarity محاسبه و نهایتا k مقاله‌ای که بیشترین شباهت را داشته باشند، باز می‌گردانیم.

در مرحله اول پارامترهای مدل تعیین می‌شوند و مدل fasttext بر روی کلمات نرمال‌شده بخش چکیده train می‌شود. در مرحله بعدی، در دیکشنری term2idf، مقادیر idf مرتبط با هر کلمه را ذخیره می‌کنیم که می‌توان بعدا از آن به عنوان وزن برای محاسبه میانگین وزن‌دار بردارهای تعبیه استفاده کرد.

```
[ ]  term_df = df_count('abstract')[0]
     term2idf = {}
     N = len(docs)
     for key, val in term_df.items():
       term2idf[key] = math.log(N/val)

     term2idf['introduce']
```

شکل 16. نگهداری idf برای هر کلمه بخش abstract در دیکشنری term2idf

در تابع embed، بردار embedding متن براساس بردارهای حاصل از مدل fasttext و وزن‌های حاصل از term2idf ساخته و خروجی داده می‌شود. ابتدا به ازای هر کلمه در متن، در صورتی که هم در vocab مدل

train و هم در دیکشنری term2idf باشد، بردار حاصل از مدل fasttext آن به لیست بردارهای متن و وزن idf آن به لیست وزن‌های متن اضافه می‌شود. سپس تابع softmax بر روی لیست وزن‌ها اعمال شده و نهایتا بردار وزن در بردارهای حاصل از fasttext در یکدیگر ضرب شده و به عنوان بردار embedding متن، خروجی داده می‌شوند.

```
[ ]  def embed(text):
         vectors = []
         weights = []
         for term in text.split():
             if term in fasttext_model.wv.vocab.keys() and term in term2idf.keys():
                 vectors.append(np.array(fasttext_model.wv[term]))
                 weights.append(term2idf[term])
         if len(weights) == 0:
             return np.zeros(embedding_size)
         weights = np.array(softmax(weights))
         vectors = np.array(vectors)
         return weights @ vectors
```

شکل 17. تشکیل بردارهای embedding حاصل از میانگین وزن‌دار بردارهای حاصل از fasttext

برای هر کوئری، بردار embedding آن محاسبه شده و میزان شباهت آن با استفاده از cosine similarity با بردار embedding متن بدست می‌آید. نهایتا k مقاله‌ای که بیشترین میزان شباهت را داشته باشند خروجی داده می‌شوند.

```
[ ]  fasttext_embeddings = docs.map(embed).to_numpy()
     fasttext_embeddings = np.array([np.array(e) for e in fasttext_embeddings])
     fasttext_embeddings.shape
```

```
[ ]  def fasttext_query(query, embeddings=fasttext_embeddings, k=10):
         start_time = time.time()

         clean_query = preprocessor.run(query)
         query_embedding = embed(clean_query).reshape(1, -1)

         cosine_scores = cosine_similarity(query_embedding , embeddings)[0]
         similar_ix = np.argsort(cosine_scores)[::-1][:k]
         return similar_ix, cosine_scores
```

شکل 18. محاسبه میزان شباهت بردارهای embedding متن و کوئری، و بازگرداندن k مقاله با بیشترین شباهت

بخش ۴. ارزیابی روش‌های بازیابی

در این بخش تعدادی کوئری برای هر بخش آماده شده و نتایج آنها برای هر روش در دیتافریم مخصوص به خودشان قرار گرفته شد. سپس این نتایج در فایل‌های csv مجزایی ذخیره شده و نهایتا در یک داک تست تجمیع شدند. نتایج هر کوئری به صورت لیستی از عناوین مقالات مرتبط است، بنابراین نیاز است تا برای ارزیابی، با مراجعه به مقاله صحت نتایج بررسی شوند.

همچنین از روش درهم‌سازی نیز استفاده شده است. به این صورت که به طور مثال برای بخش عنوان، اندیس‌های مقالات خروجی داده شده از روش Boolean و روش tf-idf در یک لیست اندیس قرار گرفته و ترتیب آنها تغییر می‌کند.

```
[ ] name_queries = ['Ashish Vaswani',
                    'Xi Wang',
                    'Jiquan Ngiam',
                    'Illia Polosukhin',
                    'Jacob Devlin',
                    'Kristina Toutanova',
                    'david',
                    'ahmad',
                    'schmid',
                    'fatemeh'
    ]

    title_queries = ['Attention is All you Need', # 19
                     'DA-DETR: Domain Adaptive Detection Transformer by Hybrid Attention', # 31
                     'Lattice-based Transformer', # 1162
                     'Leadership versus Management in Public Organizations',
                     'Chinese transformer',
                     'Prediction of chemical reaction yields using deep learning', #1598
                     'Sentiment Analysis Methods for HPV Vaccines Related Tweets Based on Transfer Learning',
                     'Sentiment Analysis',
                     'transformers in economics',
                     'Evaluating Transformers'
    ]

    abstract_queries = ['The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration.', # 19
                     'Electromagnetic Interference of Electronic Transformer', # 19
                     'Spreading fake news',
                     'Knowing how to lead does not mean knowing how to dominate, but to know how to convince people to work for a common goaf',
                     'DETR algorithm',
                     'Prediction of chemical reaction yields using deep learning',
                     'Sentiment Analysis Methods for HPV Vaccines Related Tweets Based on Transfer Learning',
                     'Sentiment Analysis',
                     'transformers in economics',
                     '''we first propose a policy gradient
    reinforcement learning (RL)-based optimal decoupling
     capacitor (decap) design method for 2.5-D/3-D
    integrated circuits (ICs) using a transformer
    network.'''
    ]
```

شکل 19. کوئری‌های مربوط به هر بخش

```
[ ]  def get_result(queries, type):
         res_df = pd.DataFrame(columns = ['query/result'])
         for i in range(len(queries)):
             query = queries[i]
             tmp = []
             tmp.append(f'q{i+1} = {query}')

             if type == 'author_boolean':
                 indx = bool_query(query, 'author')
             elif type == 'title_boolean':
                 indx = bool_query(query, 'title')
             elif type == 'title_tfidf':
                 indx = tf_idf(query, 10, word_dict, docs_dict, 'title')
             elif type == 'abstract_tfidf':
                 indx = tf_idf(query, 10, word_dict, docs_dict, 'abstract')
             elif type == 'abstract_transformer':
                 indx = most_similar(query)[0]
             elif type == 'abstract_fasttext':
                 indx = fasttext_query(query)[0]
             else:
                 return 'invalid'

             for ind in indx:
                 tmp.append(df['title'].iloc[ind])
             tmp_df = pd.DataFrame(data = tmp, columns = ['query/result'])
             res_df = res_df.append(tmp_df)
         res_df['user1'] = ''
         res_df['user2'] = ''
         res_df['user3'] = ''
         return res_df
```

شکل 20. تابع get_result که اندیس‌های خروجی از هر روش در آن محاسبه و در دیتافریم ذخیره می‌شود

```
[ ]  author_boolean_df = get_result(name_queries, 'author_boolean')
     title_boolean_df = get_result(title_queries, 'title_boolean')
     title_tfidf_df = get_result(title_queries, 'title_tfidf')

     author_boolean_df.to_csv('/content/drive/MyDrive/author_boolean.csv')
     title_boolean_df.to_csv('/content/drive/MyDrive/title_boolean.csv')
     title_tfidf_df.to_csv('/content/drive/MyDrive/title_tfidf.csv')
```

```
[ ]  abstract_tfidf_df = get_result(abstract_queries, 'abstract_tfidf')
     abstract_transformer_df = get_result(abstract_queries, 'abstract_transformer')
     abstract_fasttext_df = get_result(abstract_queries, 'abstract_fasttext')

     abstract_tfidf_df.to_csv('/content/drive/MyDrive/abstract_tfidf.csv')
     abstract_transformer_df.to_csv('/content/drive/MyDrive/abstract_transformer.csv')
     abstract_fasttext_df.to_csv('/content/drive/MyDrive/abstract_fasttext.csv')
```

شکل 21. ذخیره دیتافریم‌های حاصل از نتایج کوئری‌ها در فایل‌های csv

```
[ ]  # title evaluation
     word_dict, docs_dict = df_count('title')
     title_eval = pd.DataFrame(columns = ['query/result'])
     for i in range(len(title_queries)):
         query = title_queries[i]
         tmp = []
         indx_bool = bool_query(query, 'title')
         indx_tfidf = tf_idf(query, 10, word_dict, docs_dict, 'title')
         tmp.append(f'q{i+1} = {query}')
         all_indx = indx_tfidf + indx_bool
         all_indx = set(shuffle(all_indx))
         for ind in all_indx:
             tmp.append(df['title'].iloc[ind])
         tmp_df = pd.DataFrame(data = tmp, columns = ['query/result'])
         title_eval = title_eval.append(tmp_df)
     title_eval['user1'] = ''
     title_eval['user2'] = ''
     title_eval['user3'] = ''
     title_eval.to_csv('title.csv')
```

شکل 22. استفاده از روش درهم‌سازی برای بخش عنوان (title)

نتایج حاصل از ارزیابی، در گوگل شیت زیر آمده است:

https://docs.google.com/spreadsheets/d/1nZNZJqwV0jO3qidjMFgmN6hK4uoi0eDV0DsqQv3VKjI/edit?usp=sharing

همان طور که مشاهده می‌شود، با معیار ارزیابی MRR، نتایج مربوط به کوئری‌های بخش عنوان و نویسنده خصوصا در روش بولین، اعداد نسبتا بالایی بدست آمده‌اند که با توجه به ماهیت این بخش‌ها و اینکه انتظار داریم خود ورودی به صورت کامل در جواب باشد، منطقی است.

در بخش چکیده اعداد MRR کمتر شده‌اند و در بین روش‌های مختلف، استفاده از transformerها بهترین نتیجه را می‌دهد.