بسمه تعالی

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

پروژه پایانی

درس پردازش زبان‌های طبیعی

عنوان پروژه

سامانه‌ی پاسخگویی به سوالات پزشکی دو زبانه بر اساس مقالات

استاد درس

جناب آقای دکتر عسگری

اعضای گروه

علیرضا صاحبی

محمدعلی صدرایی جواهری

زهرا یوسفی

مهدی همتیار

علی صفرپور دهکردی

بهمن ماه ۱۴۰۰

چکیده

امروزه با توجه به فراوانی مقالات پزشکی در زمینه های مختلف، بازیابی اطلاعات دقیق و کاملا قابل اطمینان پزشکی امری دشوار است. سرویس UptoDate مجموعه‌ای از مقالات با اطلاعات کاملا قابل اطمینان را در دسترس قرار داده است. اما کمبودی که وجود دارد، عدم پشتیبانی آن از زبان فارسی برای بازیابی اطلاعات و مقالات پزشکی است. در این پروژه سعی شده است با استفاده از مدل زبانی و روش های Cross Lingual Projection، بتوان سامانه‌ای را ایجاد کرد که با وارد کردن کوئری فارسی، نزدیک ترین مقالات و پاراگراف‌های مرتبط به آن را از میان مقالات مختلف انگلیسی این سایت استخراج کرد.

فهرست مطالب

۱- مقدمه و شرح مسئله

در دنیای امروزی بیماران، پزشکان و مراقبین سلامت همواره در حال تولید اطلاعات و داده‌های پزشکی هستند و حجم این اطلاعات به سرعت در حال افزایش است. موتورهای جستجوی مختلفی جهت استخراج اطلاعات از این منابع وجود دارد که اکثرا تک زبانه هستند و با تمرکز بر زبان انگلیسی هستند. هدف از این پروژه طراحی سامانه‌ای دوزبانه جهت پاسخ به سوالات پزشکی با استفاده مقالات پزشکی است. این سامانه، سوالات کاربران به زبان‌های فارسی یا انگلیسی را دریافت کرده و پاسخ مناسب را به آنان به زبان انگلیسی برمی‌گرداند. برای این امر، از مقالات کراول شده از وب‌سایت www.uptpdate.com که منبعی برای مقالات پزشکی در زمینه‌های مختلف است، استفاده می‌کنیم.

در این گزارش ابتدا ساختار داده‌ی پروژه را معرفی و نحوه‌ی استخراج اطلاعات مورد نیاز از آن را بررسی می‌کنیم. همچنین نحوه‌ی تولید داده‌ی ارزیابی جهت بررسی مدل نهایی رو ارائه می‌دهیم. در قسمت بعدی به روش‌های مطرح شده در پروپوزال پروژه، مشکلات آن‌ها و تغییراتی که در روند حل مسئله به وجود آمد می‌پردازیم و روش‌های نهایی حل مسئله را بررسی می‌کنیم. در نهایت به جمع‌بندی و کارهای آینده جهت بهبود روش ارائه شده می‌پردازیم.

۲- داده‌ها

در این قسمت از گزارش به نحوه‌ی جمع‌آوری داده، تمیز کردن داده و تهیه‌ی داده‌ی ارزیابی می‌پردازیم.

۲-۱- جمع‌آوری داده‌ها

برای این پروژه از مقالات کراول شده از سایت www.uptodate.com استفاده می‌کنیم که منبعی برای مقاله‌های پزشکی در زمینه‌های مختلف است. مجموعه داده‌ی ما حاوی ۱۸۴۸۸ مقاله است.
فایل‌های مرتبط با کراول کردن این وب‌سایت در ریپازیتوری پروژه در پوشه‌ی 00_uptodate_crawler قرار دارند. این کراولر تلاش می‌کند با استفاده از نام کاربری و پسورد اکانت وب‌سایت وارد وب‌سایت شود. مکانیزم امنیتی این سایت به این صورت است که اگر درخواست‌ها تنها از نوع GET باشند آن اکانت بلاک می‌شود. در نتیجه، این کراولر برای جلوگیری از بلاک شدن درخواست‌ها توسط دو وب‌سایت، پس از هر بیست ریکوئست، یک بار از سایت خارج و دوباره وارد سایت می‌شود. علاوه بر این کار، با یک dummy search (با متدی به همین نام در فایل uptodate_crawler.py) بعد از ورود دوباره به سایت، کلمه‌ی "drug" را در سایت سرچ می‌کنیم تا از بلاک شدن درخواست‌های بعدی جلوگیری کنیم. مقاله‌های کراول شده به صورت یک دیتابیس MongoDB و در یک فایل JSON ذخیره کردیم.

```
00_uptodate_crawler > main.py
1   from connector import *
2   from uptodate_crawler import UptodateCrawler
3   import pymongo
4
5   MONGO_INFO = 'mongodb://root:example@localhost:27017/?authSource=admin&readPreference=primary&appname=MongoDB%20Compass&ssl=fa
6
7   IRAN_PAPER_LOGIN = ['mohalisad', 'p:159357']
8   UPTODATE_LOGIN = ["mod723bh2@icloud.com", "7Pc1V9KPUBaz"]
9
10  if __name__ == '__main__':
11      db_connection = pymongo.MongoClient(MONGO_INFO)
12      connector = DirectConnector(*UPTODATE_LOGIN)
13      crawler = UptodateCrawler(connector, db_connection["uptodate"])
14      crawler.start_crawl()
```

فایل JSON حاوی فایل‌های مقاله‌های UptoDate، حاوی لیستی از دیکشنری‌هاست و هر عضو این لیست یکی از مقاله‌هاست و طول لیست یا تعداد مقاله‌ها براب با ۱۸۴۸۸ مقاله است. شکل زیر نمونه‌ای از ساختار یکی از مقاله‌های موجود در این لیست را نشان می‌دهد:

topicInfo حاوی اطلاعات مختلفی در مورد مقاله از جمله عنوان و زبان آن و bodyHtml حاوی متن اصلی مقاله است.

۲-۲- مرتب‌سازی داده‌ها

بنابر حالت معمول تسک‌های مختلف که با داده‌ها سر و کار دارند، نیاز است تا به نحوی داده‌ها آماده گردند تا در زمان اجرای الگوریتم‌های پردازش داده خروجی مناسب‌تری دریافت گردد. همچنین با توجه به دوزبانه بودن داده‌های این پروژه بدیهی است که باید به صورت موازی توابعی برای هر دو زبان با توجه به نیاز هر یک تهیه گردد. جزئیات بیشتر پیاده‌سازی توابع استفاده شده در این قسمت در پوشه‌ی 01_mongodb_to_parags در ریپازیتوری پروژه مشاهده کنید.

یکی از بخش‌های مهم اینچنین تسک‌ها پاک کردن علامات علامات نگارشی است که می‌تواند کار پردازش را بسیار ساده کند. در ادامه تکه کدی که برای انگلیسی و به طور مشابه برای فارسی استفاده می‌شود را مشاهده می‌نمایید که لیستی از این علامت‌ها را با دستور جایگزینی در متن حذف می‌کنند.

```
def en_remove_punc(s):
  punc = '"#\'*+,-/:;<=>@[\]^_`{|}~\'●,•()»«---،−؛,--'
  table = str.maketrans(dict.fromkeys(punc, ' '))
  new_s = s.translate(table)
  new_s = ' '.join(new_s.split())
  return new_s
```

همچنین تابع نرمال‌ساز مطابق کد زیر ارائه گردیده است که با حذف برخی کاراکترها متن را خالص‌تر می‌کند و از آن مهم‌تر با کوچک کردن همه حروف، متن را یک‌دست می‌کند. یا مثلا در خط چهارم کد زیر اطلاعات سایتیشن‌ها که اکنون بی مصرف هستند را حذف نموده‌ایم. هم‌چنین خط ششم نقطه آخر جمله را از متن جدا می‌کند اما در مورد اعداد اعشاری می‌تواند مدیریت کند تا به چالش نخورد. همچنین خطوط آخر برای این است که فرمت این علامت‌ها مشترک شود تا بی دلیل با تنوع علامات روبه رو نشویم.

```
def en_normalizer(text):
  text = text.lower()
  text = text.replace('\xa0','')
  text = re.sub(r"\[[\d| ]+\]", " ", text)
  text = en_remove_punc(text)
  text = re.sub(r"(\w{2,}| )\.([^0-9]|\n|$)", r"\1 . \2", text)
  text = re.sub(r"!", " ! ", text)
  text = re.sub(r"\?", " ? ", text)
  text = re.sub(r"؟", " ؟ ", text)
  text = re.sub(r" +", " ", text)
  return text
```

در مورد زبان فارسی نیز چنین موردی وجود دارد. تبدیل ارقام به فارسی، تبدیل حروف چند شکلی از عربی به فارسی مانند حرف ک و حرف ی. مدیریت نیم‌فاصله و کاراکترهای زائد از دیگر مواردی است که انجام شده است. همچنین برای داده فارسی نیز به طور مشابه محاسباتی انجام شده است که در تکه کد زیر مشاهده می‌فرمایید:

```
def fa_normalizer(text):
  text = arToPersianChar(text)
  text = arToPersianNumb(text)
  text = text.replace('\xa0','')
  text = text.replace('ً', '')
  text = fa_remove_punc(text)
    normalizer = Normalizer(persian_style = False, punctuation_spacing = False,
affix_spacing = False)
  text = normalizer.normalize(text)

  text = text.replace('\u200c',' ')
  text = text.replace('\u200b',' ')

  text = re.sub(r"(\w{2,}| )\.([^0-9]|\n|$)", r"\1 . \2", text)

  text = re.sub(r'([\d+])\.([\d+])', r'\1٫\2', text)
  text = re.sub(r'([\d+])/([\d+])', r'\1٫\2', text)
```

```
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\?", " ? ", text)
    text = re.sub(r" +", " ", text)
    text = re.sub(r" +", " ", text)
    return text
```

در تکه کد فوق دستورهایی برای تبدیل علامت‌ها به فرمت فارسی رایج، حذف نیم‌فاصله‌ها به این دلیل که گاهی رعایت نمی‌شوند، جدا کردن نقاط از متن مگر در اعداد اعشاری و ... مشاهده می‌شود. دو خط اول نیز عملا دو دیکشنری از حروف خاص و ارقام هستند که از عربی به فارسی بازگردانی می‌شوند تا در متون مختلف یکتا باشند.

۲-۳- تهیه‌ی داده‌ی ارزیابی

با توجه به ساختار هر مقاله که در قسمت قبل معرفی شد، نیاز است آن قسمت از دیکشنری هر مقاله که برای حل مسئله‌ی مطرح شده در این پروژه نیاز است استخراج و برای پردازش آماده گردد.

در بخش جمع‌آوری داده‌ها ذکر شد که داده تهیه شده به صورت JSON است و این در حالی است که کار با داده متنی بسیار آسان‌تر است. از طرفی بخش زیادی از داده‌های جمع‌آوری شده مفید نیستند و باید حذف شوند. مثلا در ستون topicInfo تنها ۴ دسته از داده‌ها مفید هستند که عبارتند از subtype، type، id و title که id به عنوان شناسه‌ی هر مقاله کاربرد دارد. عنوان هم متن با title مشخص شده است و نوع و زیرنوع نیز در ادامه کاربرد معینی دارند. تابع get_paper_info با دریافت دیکشنری هر مقاله اطلاعات اصلی آن مقاله را استخراج کرده و به شکل یک دیکشنری در اختیار ما قرار می‌دهد:

```
def get_paper_info(row):
    return {
        'id': row['topicInfo']['id'],
        'type': row['topicInfo']['type'],
        'subtype': row['topicInfo']['subtype'],
        'title': row['topicInfo']['title']
    }
```

یک کتابخانه محبوب در مواجهه با داده‌هایی که فرمت json دارند کتابخانه BeautifulSoup می‌باشد.

```
from bs4 import BeautifulSoup
```

یکی دیگر از توابع تعریف شده تابع get_paragraphs است که برای استخراج متن هر پاراگرف از هر مقاله استفاده خواهد شد.

```
def get_paragraphs(row)
```

در این تابع ابتدا یک زیر تابع به نام add_parag تعریف شده است که با دریافت دیکشنری مقاله لیستی از پاراگراف‌های آن مقاله را استخراج می‌کند به طوری که اعضای آن لیست tupleهای سه‌تایی حاوی شماره‌ی پاراگراف در آن مقاله یا id، عنوان مقاله یا title و محتوای پاراگراف یا text است. این تابع پاراگراف‌هایی با طول کمتر از ۵۰ کاراکتر را دور می‌ریزد چرا که اطلاعات زیادی در اختیار ما قرار نمی‌دهند.

```
def add_parag(title, text):
    nonlocal idx
    if len(text) > 50:
        return_value.append((idx,title, text))
        idx += 1
```

در ادامه، تابع get_paragraphs تلاش می‌کند تا به کمک کتابخانه فوق‌الذکر اطلاعات مفید هر مقاله را استخراج کند. کلاس‌های مختلفی از داده HTML که در داده بدنه وجود دارد به کمک soup تحلیل می‌شوند و با تحلیل دستی و بررسی نمونه‌های مختلف در داده خام مشخص شده که کلاس glyph عملا شامل داده مفیدی نمی‌باشد و همچنین اگر داده دارای headingAnchor نباشد داده مفید نبوده و بررسی نباید گردد. Soup بررسی می‌کند که آیا متن مقاله حاوی division یا divای از کلاس headingAnchor هست یا خیر. در صورتی که این شرط برقرار نباشد آن مقاله را کنار می‌گذاریم. نزدیک به ۹۰۰۰ مقاله به دلیل اینکه در قابل تعریف شده صدق نکردند کنار گذاشته شدند.

در دیتای مقاله، divهایی با id = topicText که متن اصلی مقاله است را استخراج کرده و با iterate کردن روی پاراگراف‌ها و تشخیص عناوین برخی پاراگراف‌ها با استفاده از headingAnchor، پاراگراف‌ها و عناوین آن‌ها را استخراج می‌کنیم. در نهایت داده‌ی استخراج شده را به فرم csv ذخیره می‌کنیم. هر سطر این فایل csv شامل id مقاله و یک پاراگراف از آن مقاله، عنوان مقاله و پاراگراف متناظر با id پاراگراف است.

```
body = row['bodyHtml']
soup = BeautifulSoup(body)
for div in soup.find_all("span", {'class':'glyph'}):
    div.decompose()
```

```
if len(soup.find_all("div", {'class':'headingAnchor'})) == 0:
    return None

parags = soup.find("div", {"id": "topicText"})
```

در ادامه نیز با بررسی پاراگراف‌های آن که با کلیدواژه child مشخص شده‌اند مواردی که دارای header و سایر اطلاعات لازم باشند را مشخص نموده و نگهداری خواهند شد. و در نهایت لیستی از این پاراگراف‌ها باز گردانده خواهد شد.

عنوان برخی از مقالات با عبارت Patient Education آغاز شده است. این مقالات title و subtitleهای معنی‌دارتری دارند و برای evaluation مناسب هستند. با توجه به تعداد بالای این مقالات، یک سیزدهم آن‌ها که برابر با ۵۳۸ مقاله است به طور رندوم برای Evaluation انتخاب کردیم. در ادامه با استفاده از google translate عناوین این مقاله‌ها جهت تولید کوئری فارسی ترجمه شد. درستی این ترجمه‌ها و ارتباط آن‌ها با محتوای پاراگراف به صورت دستی بررسی شد و در نهایت از بین این ۵۳۸ پاراگراف، ۳۰۰ پاراگراف برای Evaluation تایید شدند. شکل زیر نمونه‌ای از داده‌های ارزیابی را نشان می‌دهد:

paper_id	parag_id	title middle	subtitle	contcated	ترجمه ماشینی	ترجمه تمیز	text
2159	5	Premenstrual syndrome (PMS) and premenstrual dysphoric disorder (PMDD)	PMS AND PMDD DIAGNOSIS	Premenstrual syndrome (PMS) and premenstrual dysphoric disorder (PMDD) PMS AND PMDD DIAGNOSIS	سندرم پیش از قاعدگی (PMS) و اختلال دیسپریک قبل از قاعدگی (PMDD) PMS و تشخیص PMDD	سندرم پیش از قاعدگی و اختلال دیسپریک قبل از قاعدگی	There is no single test that can diagnose premenstrual syndrome (PMS) or premenstrual dysphoric disorder (PMDD). The symptoms must occur only during the second half (luteal phase) of the menstrual cycle, most often during the five to seven days before the menstrual period, and there must be physical as well as behavioral symptoms. In women with PMS or PMDD, these symptoms should not be present between days 4 through 12 of a 28-day menstrual cycle.
3421	13	Syncope (fainting)	Other causes	Syncope (fainting) Other causes	Syncope (همه‌گی) علل دیگر	علل دیگر عش کردن	Less common causes of syncope include cardiac tumor or blood clot in the arteries supplying the lungs.

۳- بررسی روش‌های مطرح شده در پروپوزال پروژه

در این قسمت به روش‌های مطرح شده در پروپوژال و دلایل استفاده از این روش‌ها و یا کنار گذاشتن آن‌ها می‌پردازیم.

۳-۱- استفاده از مدل‌های مبتنی بر BERT مانند mBERT جهت تولید امبدینگ‌های Dense

روش پیشنهادی پروپوزال پروژه برای تولید امبدینگ‌های Dense با استفاده از BERT، استفاده از BioBERT بود که fine tune کردن آن با استفاده از متون مقالات UptoDate انجام شده است. در ادامه، در ترکیب این روش با مدل ParsBERT گروه تحقیقاتی هوش‌واره، می‌توان از شبکه‌های عصبی برای یادگیری تبدیل امبدینگ‌های ParsBERT به BioBERT استفاده کرد. ورودی این شبکه‌ی عصبی امبدینگ تولیدی توسط ParsBERT بوده و خروجی آن‌ها امبدینگ Dense عبارات است. روش دیگر، استفاده از مدل‌های چندزبانه‌ی BERT مانند mBERT بود.

جهت بررسی این روش، خروجی مدل mBERT را برای تعدادی کلمات و عبارات فارسی و معادل انگلیسی آن‌ها تولید کرده و فاصله‌ی امبدینگ‌ها را مقایسه کردیم. برای اینکار ابتدا خروجی tokenizer برای این عبارات را در اختیار مدل قرار داده تا امبدینگ آن‌ها تولید شوند. با توجه به چند زبانه بودن این مدل، انتظار داشتیم امبدینگ کلمات معادل به هم نزدیکتر باشند در حالی که طبق شکل زیر، امبدینگ کلمه‌ی "hypertension" به امبدینگ کلمه‌ی «پروتئین» نزدیک‌تر است. در نتیجه این روش برای مسئله‌ی ما مناسب نیست زیرا هدف، تولید این امبدینگ برای پاراگراف‌های انگلیسی دیتای UptoDate به نحوی است که مفهوم پاراگراف در بردار امبدینگ خلاصه شده و برای پشتیبانی مدل نهایی از دو زبان فارسی و انگلیسی، بردار انگلیسی و فارسی یک عبارت به همدیگر نزدیک باشند.

بررسی‌های بیشتر نشان داد که مدل‌های مبتنی بر BERT که زبان فارسی را پشتیبانی می‌کنند (شامل mBERT، ParsBERT و ...) با استفاده از متون محدود فارسی آموزش داده شده‌اند و برای کار با متون پزشکی مناسب نیستند و توکنایزر این مدل‌ها، کلماتی مانند «سردرد» را به سه توکن «سر»، «در» و «د» می‌شکنند که با توجه به مفهوم این عبارت در متون پزشکی، مناسب مسئله‌ی ما نیستند. با جستجوی بیشتر با مدل‌هایی مثل SinaBERT که جهت رفع ضعف BioBERT در کار با متون پزشکی فارسی طراحی شده‌اند آشنا شدیم اما این مدل‌ها به صورت آزاد در سایت hugging face قابل دسترس نبود و امکان استفاده از اون رو نداشتیم. از جمله

روش‌هایی که برای حل این مشکل ارائه شد، استفاده از مدل‌های ترجمه‌ی ارائه شده توسط دانشگاه Helsinki بود که جهت ترجمه‌ی متون ۱۳۰۰ زبان به زبان انگلیسی ارائه شده‌اند اما در بین زبان‌های ارائه شده، زبان فارسی وجود نداشت پس این روش کنار گذاشته شد. در نهایت برای حل مشکل امبدینگ به سراغ مدل BERN2 رفتیم که در قسمت‌های بعدی به آن می‌پردازیم.

```
[143]  1 en_data = ['hypertension', 'headache', 'health', 'protein', 'amino acid', 'gout']
       2 fa_data = ['فشار خون',
       3            'سردرد',
       4            'سلامت',
       5            'پروتئین',
       6            'آمینو اسید',
       7            'نقرس']
```

```
[144]  1 en_vecs = []
       2 fa_vecs = []
```

```
[145]  1 for en, fa in zip(en_data, fa_data):
       2   encoded = tokenizer(en, return_tensors='pt')
       3   output = model(**encoded)
       4   en_vecs.append(output['pooler_output'][0].tolist())
       5
       6   encoded = tokenizer(fa, return_tensors='pt')
       7   output = model(**encoded)
       8   fa_vecs.append(output['pooler_output'][0].tolist())
```

```
[146]  1 scipy.spatial.distance.cdist(fa_vecs, en_vecs)
```

```
array([[4.49124442, 4.10290994, 4.56774646, 3.94894188, 4.98715309,
        4.61554972],
       [5.22951501, 6.16316353, 6.10238268, 6.14681543, 3.98063411,
        4.07409184],
       [3.87686932, 4.95372417, 4.51985922, 4.68855847, 3.89093267,
        3.90250398],
       [3.68725995, 6.47504822, 5.66566996, 4.45152113, 5.51842886,
        5.91497204],
       [4.38065885, 6.64847261, 6.15736853, 4.87534678, 5.78684372,
        6.28403543],
       [5.24888415, 6.26492959, 5.83085803, 6.02275735, 3.01358032,
        4.21333113]])
```

۳-۲- مدل DenSPI

همانطور که در پروپوزال پروژه گفته شد، برای استفاده از این مدل پاراگراف های مقالات UptoDate به عنوان سند در نظر گرفته می‌شوند. برای هر Phrase (مجموعه کلمات متوالی با اندازه های مختلف) در هر پاراگراف، یک بردار امبدینگ در نظر گرفته می‌شود که شامل یک قسمت Dense (امبدینگ توکن CLS حاصل از ورودی Phrase به مدل BERT) و یک قسمت Sparse (مجموعه ای از ویژگی های مختلف Phrase، از جمله وجود نام بیماری، دارو، نام‌های بیولوژیکی مانند نام پروتئین‌ها و ... در Phrase) است. برای کوئری هم به همین صورت بردار امبدینگ استخراج می‌شود. با استفاده از ابزار faiss در فضای برداری ایجاد شده، نزدیک‌ترین همسایه به امبدینگ کوئری از میان تمام امبدینگ های Phraseها پیدا می‌شود و به عنوان پاسخ پیشنهادی بازگردانده می‌شود. اما این روش به دلایل زیر کنار گذاشته شد:

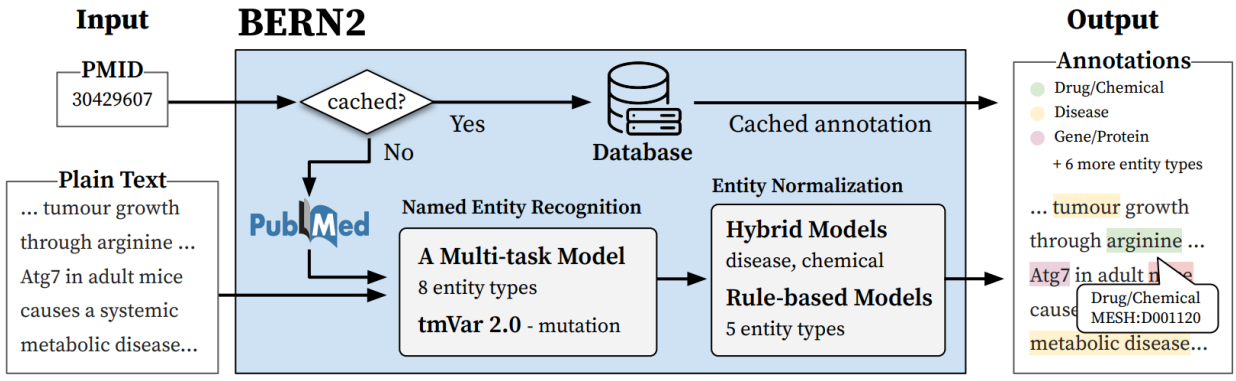
- منسوخ بودن مستندات و برنامه‌های مرتبط با این مدل
- عدم دسترسی به مدل pre-train شده‌ی DenSPI

۴- بررسی پیاده‌سازی نهایی

در این قسمت به پیاده‌سازی‌ها نهایی جهت حل مسئله‌ی مطرح شده در این پروژه می‌پردازیم.

۴-۱- تگ زدن مقالات با استفاده از مدل BERN2

در این مرحله پاراگراف‌های مقالات UptoDate توسط ابزار BERN2 تگ زده شده اند. BERN2 ابزاری است که آزمایشگاه DMIS دانشگاه کره آن‌را توسعه داده است. مدل BERN2 ابزاری برای NER و مبتنی بر مدل BioBERT است. با توجه به افزایش سریع حجم متون پزشکی و اهمیت بالای آن‌ها، Biomedical Text Mining از اهمیت بالایی برخوردار است. این متون حاوی entityهایی از انواع مختلف هستند از جمله ژن‌ها، داروها و ... و مدل BERN2 جهت استخراج و annotate کردن این entityها طراحی شده است. ساختار این مدل به صورت زیر است:



این مدل دو نوع ورودی را دریافت می‌کند: ۱. PMID یا PubMed Reference Number که شناسه‌ای است که مرکز سلامت ملی آمریکا به هر مقاله اختصاص می‌دهد ۲. متن خام پزشکی. با توجه به اینکه کار ما در این پروژه با متن خام مقاله‌ها است، در این گزارش به PMID نمی‌پردازیم.

وقتی BERN2 متن خام را به عنوان ورودی دریافت می‌کند، ابتدا مدل‌های NER موقعیت دقیق entityهای مختلف متن را مشخص می‌کنند. برای اینکار از یک شبکه‌ی عصبی multi-task استفاده می‌شود که با استفاده از دیتاست‌های NER مختلف آموزش داده شده است. این مدل عملیات annotate کردن entityها متن را سرعت می‌بخشد. پس از این مرحله، هر entity به CUI یا Concept Unit Identifierهای خود لینک می‌شوند.

برای دست یافتن به مدل زبانی فارسی بهتر، نیاز به آموزش مدل با متونی داشتیم که دارای موضوعاتی مشابه موضوعات مطرح شده در مقالات UptoDate باشد. برای اینکار ابتدا با استفاده از مدل BERN2، کلمات کلیدی

متون مقاله‌های UptoDate را استخراج و annotate کرده و سپس با استفاده از ویکی‌پدیا، معادل فارسی این کلمات و متون آن‌ها را استخراج کردیم و سپس از متون فارسی استخراج شده برای یادگیری مدل زبانی فارسی استفاده کردیم.

برای استفاده از BERN2، مطابق دستورات ارائه شده در صفحه‌ی گیت‌هاب این مدل، سرور BERN2 را روی کولب(Pro) راه‌اندازی کردیم. علاوه براین با استفاده از REST API، به وب سایت BERN2 به آدرس bern.korea.ac.kr/plain کوئری زدیم. متاسفانه در زمان annotate کردن پاراگراف‌ها، به مدت سه روز سرورهای دانشگاه کره خاموش بودند که روند پروژه را دچار اختلال کرد.

تابع query_plain که پیاده‌سازی آن در شکل زیر آمده است، وظیفه‌ی کوئری زدن را برعهده دارد:

```python
def query_plain(text, url="http://bern2.korea.ac.kr/plain"):
    trys = 0
    while(True):
        try:
            return requests.post(url, json={'text': text}).json()
        except Exception as e:
            #print('\n'+str(len(text))+'\n')
            print(F"Error in request: {str(e)}")
            print('Sleep for 1 seconds')
            time.sleep(1)
            if 'Expecting value' in str(e):
                trys += 1
                text = re.sub(r'www\.\w+\.\w+', '', text)
            if trys > 10:
                return {'annotations':[]}
```

خروجی این تابع که نحوه‌ی پاسخ‌گویی BERN2 به درخواست‌ها ماست، برای هر پاراگراف به صورت زیر خواهد بود که entityهای خاص مانند Skin Cancer یا سرطان پوست را استخراج و نوع آن را مشخص کرده است.


```
{'annotations': [{'id': ['NCBITaxon:9606'],
   'is_neural_normalized': False,
   'mention': 'people',
   'obj': 'species',
   'prob': 0.980154275894165,
   'span': {'begin': 3, 'end': 9}},
  {'id': ['mesh:D012878'],
   'is_neural_normalized': False,
   'mention': 'skin cancer',
   'obj': 'disease',
   'prob': 0.9999760389328003,
   'span': {'begin': 44, 'end': 55}},
  {'id': ['mesh:D004098'],
   'is_neural_normalized': False,
   'mention': 'dihydroxyacetone',
   'obj': 'drug',
   'prob': 0.9996727705001831,
   'span': {'begin': 299, 'end': 315}}],
 'text': 'As people become more aware of the risks of skin cancer from sun exposure and tanning beds,
 'timestamp': 'Tue Feb 15 18:24:35 +0000 2022'}
```

برای استخراج معادل فارسی اصطلاحات پزشکی (که با NER از مجموعه مقالات استخراج کردیم) و اطلاعاتی در مورد آن‌ها از ویکی‌پدیا استفاده کردیم. اینکار در سه مرحله انجام شد و فایل‌های مربوط به این قسمت در پوشه‌ی 03_wiki_crawler در ریپازیتوری پروژه قرار دارند:

۱. wiki_step1.py: این برنامه لیستی حاوی ده کلمه‌ی انگلیسی را دریافت کرده و نزدیک‌ترین عناوین صفحات ویکی‌پدیا به آن کلمات و معادل فارسی آن را در صورت وجود پیدا کرده و به ما برمی‌گرداند. در نهایت با استفاده از فایل wiki_step1_se.py، کلماتی که معادل انگلیسی و فارسی آن‌ها وجود داشتند در فایل paris.json و کلماتی که تنها معادل انگلیسی آن‌ها وجود داشت در فایل orphans.json ذخیره کردیم. با توجه به اینکه این API عناوینی که فاصله‌ی آن‌ها از اصطلاح داده شده از مقدار مشخصی بیشتر بود را دور می‌ریخت، به سراغ روش دیگری رفتیم.

۲. wiki_step2.py: در این فایل علاوه بر جستجوی عناوین صفحات ویکی‌پدیا، محتوای صفحات را نیز بررسی کردیم و اولین نتیجه را به عنوان مقاله مرتبط به آن اصطلاحات ذخیره کردیم. در نهایت، خروجی این برنامه را با خروجی برنامه اول ترکیب کرده و در فایل merge.json ذخیره کردیم و لیستی از عناوین صفحات ویکی‌پدیا به دست آوردیم. این کار با استفاده از فایل merge.py انجام شد. شکل زیر نشان‌دهنده‌ی افزایش طول لیست اصطلاحاتی است که با جستجوی متن صفحات ویکی‌پدیا به دست آوردیم:

	With Search	Without Search
Pair	20780	12765
Orphan	36351	20941

۳. wiki_step3.py: این فایل با ریکوئست زدن به API فارسی و انگلیسی ویکی‌پدیا، متن صفحات ویکی‌پدیای تمام اصطلاحاتی که معادل فارسی و انگلیسی آن‌ها را در مراحل قبلی به دست آمد را به دست آورد.

ترتیب و نحوه‌ی اجرای فایل‌های ذکر شده به صورت زیر است:

- PYTHON wiki_step1.py
- PYTHON wiki_step2.py
- PYTHON wiki_step1_se.py
- PYTHON merge.py
- PYTHON wiki_step3.py

۴-۲- پیاده‌سازی معیار IDF

یکی از معیارهای معروف در پردازش متن روش IDF است که معادل با معکوس فروانی سند می‌باشد. به عنوان مثال کلمه‌ی "the" در متون انگلیسی بسیار استفاده می‌شود و نمی‌تواند بیانگر امتیاز خاصی باشد پس بهتر است از این معیار معکوس فراوانی استفاده گردد. برای پیاده سازی این روش راهبردهای مختلفی از جمله پیاده‌سازی از پایه بررسی گردید اما مشکل سرعت و مصرف ممووری باعث شد تا در نهایت روشی که در ادامه توضیح داده می‌شود انتخاب گردد.

در این روش وجود کتابخانه sklearn بسیار اهمیت دارد به طوری که بر مبنای توابع آن پیاده شده است:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
```

در این روش یک تابع برای محاسبه‌ی TF-IDF تهیه شده است .برای شروع کار نیاز است تا مجموعه کلمات کل متون تشکیل شود و این کار به کمک تبدیل لیست‌ها به مجموعه و محاسبه اجتماع مجموعه‌ها میسر می‌شود. در مرحله‌ی بعد، این تکه کد اجرا می‌شود:

```
pipe = Pipeline([('count',
CountVectorizer(vocabulary=vocabulary)),
                 ('tfid', TfidfTransformer())]).fit(corpus)

idf__ = pipe['tfid'].idf_
```

همانطور که در کد فوق مشاهده می‌شود یک پایپ‌لاین برای محاسبات مد نظر در دسترس خواهد بود که خروجی آن در خط آخر با صدا زدن "idf_" یک لیست از اعداد که همان مقادیر IDF لیست کلمات هستند را محاسبه می‌نماید.

FastText -۳-۴

FastText کتابخانه ای است که توسط تیم تحقیقاتی فیس بوک برای یادگیری کارآمد در نمایش کلمات و طبقه بندی جملات ایجاد شده است. این کتابخانه در جامعه NLP کشش زیادی به دست آورده و یک جایگزین احتمالی برای بسته Gensim است که عملکرد وکتورهای ورد را فراهم می کند. ما در این پروژه از این پکیج استفاده کرده‌ایم. حال به جزئیات اجرایی این مدل می‌پردازیم.

تبدیل کلمه به vec رویکرد مدرن در پردازش زبان طبیعی برای تجزیه و تحلیل یک متن به دو روش است: با نظارت supervised و بدون نظارت unsupervised. نمایش کلمات در بردارها (اعداد) مزایای مختلفی دارد از جمله امکان استفاده از مدل‌های مدرن یادگیری عمیق دستگاه که در ادامه پروژه باید از آن‌ها استفاده کنیم.

FastText در اصل یک الگوی افزودنی از Word2Vec است ولی تفاوت های با آن دارد. تفاوت اصلی میان FastText و Word2Vec استفاده از n-gram است. Word2Vec بردارها را فقط برای کلمات کامل موجود در گروه آموزشی می آموزد. از سوی دیگر، FastText بردارهای n-gram موجود در هر کلمه و همچنین هر کلمه کامل را یاد می‌گیرد. در هر مرحله آموزش در FastText، از میانگین بردار کلمه هدف و بردارهای n-gram مؤلفه آن برای آموزش استفاده می‌شود. تنظیماتی که از خطا محاسبه می شود، به طور یکنواخت برای به‌روزرسانی هر یک از بردارهایی که برای تشکیل هدف ترکیب شده‌اند، استفاده می شود. این محاسبات اضافی زیادی را به مرحله آموزش تحمیل می‌کنند. در هر مرحله، یک کلمه باید اجزای n-gram آن را جمع کند و میانگین آن را محاسبه کند. تجارت، مجموعه ای از بردارهای کلمه‌ای است که شامل اطلاعات زیرکلمه‌ای تعبیه شده هستند. نشان داده شده است که این بردارها با اقدامات مختلفی دقیق‌تر از بردارهای Word2Vec هستند.

ما در این پروژه دو مدل بدون نظارت برای لغات فارسی و انگلیسی بر اساس مقالات پزشکی آموزش داده‌ایم. مدل FastText انگلیسی بر روی متون مقالات UptoDate و FastText فارسی بر روی دادگان بیژن خان، داده‌های سلامت سایت نمناک و HiDoctor و مقالات ویکی‌پدیا نیز آموزش داده شده‌اند تا لغات بیشتری را پوشش داده باشیم. شکل‌های زیر به ترتیب نتایج آموزش FastText برای متون فارسی و انگلیسی را نمایش می‌دهند:

```
1 !./fasttext skipgram -input '$fa_data' -output /content/drive/MyDrive/BIFO/temp/fasttext/model_fa -dim 300 -epoch 20

Read 13M words
Number of words:  50730
Number of labels: 0
tcmalloc: large alloc 2460876800 bytes == 0x55a52c85a000 @  0x7f064c12d887 0x55a5217bfa76 0x55a5217cd63e 0x55a5217d5192
Progress: 100.0% words/sec/thread:   10475 lr:  0.000000 avg.loss:  1.106538 ETA:   0h 0m 0s
```

```
1 !./fasttext skipgram -input '$en_data' -output /content/drive/MyDrive/BIFO/temp/fasttext/model -dim 300 -epoch 20

Read 103M words
Number of words:  162300
Number of labels: 0
tcmalloc: large alloc 2594766848 bytes == 0x560559646000 @  0x7ff2745f0887 0x5605479b3a76 0x5605479c163e 0x5605479c9192
Progress: 100.0% words/sec/thread:    8525 lr:  0.000000 avg.loss:  0.242010 ETA:   0h 0m 0s
```

Cross Lingual Projection -۴-۴

فرض کنیم دو ماتریس X و Y برای زبان‌های مبدا (در این پروژه، زبان فارسی) و مقصد (در این پروژه، زبان انگلیسی) داریم به طوری که سطر i در ماتریس X و Y حاوی امبدینگ یک کلمه‌ی خاص در زبان هر ماتریس باشد. هدف تبدیل این امبدینگ‌ها به یکدیگر است. این تبدیل با یک linear regression با استفاده از پکیج numpy انجام می‌شود. می‌توان با استفاده از gradient descent به نتایج بهتری رسید. پیاده‌سازی این روش را در شکل زیر می‌بینید:

```python
def train(numpy, X, Y, R, epochs=4000, lr=0.0003, log_interval=1000):

    assert X.shape[0] == Y.shape[0] # training size must be the same
    assert X.shape[1] == Y.shape[1] # it is prefered that they have the same embedding size

    m = X.shape[0] # training size
    n = X.shape[1] # embedding size

    for i in range(epochs):
        grad = X.T @ (X @ R - Y) * 2 / m
        R -= lr * grad

        if i % log_interval == 0:
            loss = numpy.sum((X @ R - Y) ** 2) / m
            print(f"loss at iteration {i} is: {loss:.4f}")

    return R

def train_normal_equation(numpy, X, Y):
    R = numpy.linalg.pinv(X.T @ X) @ X.T @ Y
    loss = numpy.sum((X @ R - Y) ** 2) / m
    print(f"loss is: {loss:.4f}")
    return R
```

استفاده از PyTorch حل این مسئله را ساده‌تر می‌کند. برای این کار مدلی حاوی تنها یک لایه‌ی خطی fully connected تعریف می‌کنیم تا بهترین تبدیل خطی برای تبدیل X به Y را به دست آوریم. پیاده‌سازی و آموزش این مدل در شکل زیر آمده است:

```python
def train(X, Y, model, device, epochs=4000, lr=0.0003, log_interval=1000):
    inputs = Variable(torch.from_numpy(X).to(device)).float()
    targets = Variable(torch.from_numpy(Y).to(device)).float()

    criterion = torch.nn.MSELoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)

    for i in range(epochs):
        optimizer.zero_grad()
        preds = model(inputs)
        loss = criterion(preds, targets)

        loss.backward()

        optimizer.step()

        if i % log_interval == 0:
            print(f"loss at iteration {i} is: {loss.item():.4f}")
```

```python
import numpy as np

m = 1000
n = 300

X = np.random.uniform(size=(m, n))
Y = np.random.uniform(size=(m, n))

model = torch.nn.Sequential(torch.nn.Linear(n, n))
model.to(device)

train(X, Y, model, device)
```

Milvus -۴-۵

سامانه‌ی Milvus یک موتور جستجو برای وکتورهاست که به تنهایی نمی‌تواند به عنوان یک موتور جستجو برای متون عمل کند اما در سال گذشته bootcampای جهت ارائه‌ی یک موتور جستجو با استفاده از Milvus برگزار شد که ما هم طبق پیشنهاد دکتری عشگری از جواب همین مسابقه برای این پروژه استفاده می‌کنیم.

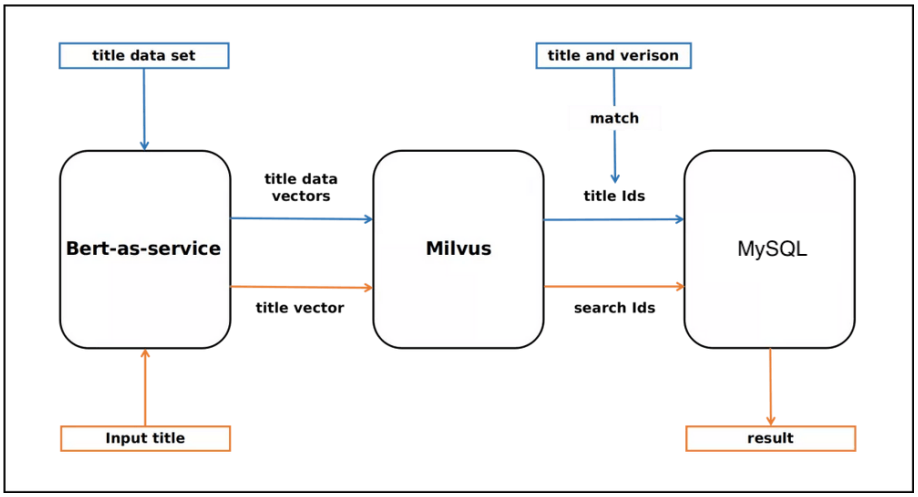
این کدبیس شامل یک فایلdocker-compose حاوی شش container مختلف است که سه عدد از این کانتینرها مرتبط با milvus هستند:

۱. etcd که به عنوان دیتابیس عمل می‌کند.
۲. minio که storage management را بر عهده دارد.
۳. standalone که همان موتور جستجوی Milvus است.

در bootcampای که به آن اشاره کردیم، علاوه بر سه کانتینر ذکر شده، سه کانتینر دیگر نیز به سامانه اضافه شده که به جستجوی متون کمک می‌کنند:

۱. mysql
۲. webclient که کلاینت وب برای جستجوی متن است.
۳. webserver

شکل زیر رابطه‌ی Milvus با سایر componentهای موتور جستجوی متن را نمایش می‌دهد. در این پروژه، به جای استفاده Bert-as-service، از encoder پیاده‌سازی شده در این پروژه استفاده کردیم که داده‌ی ورودی را دریافت کرده و آن‌ها را به وکتور تبدیل می‌کند که به این صورت می‌توان در آن‌ها عملیات جستجو را انجام داد که بخشی از این جستجوها با استفاده از mysql ذخیره می‌شوند. با توجه به این تغییر و پیچیدگی encoder پیاده‌سازی نسبت به Bert-as-service، از کانتینر webserver خود Milvus استفاده نشد. در نتیجه نیاز بود وب سرور دیگری را با توجه به نیازهای مسئله پیاده‌سازی کنیم.



دلیل پیاده‌سازی encoder جدید، تفاوت در زبان کوئری‌ها و پاسخ آن‌هاست زیرا کوئری‌های ما به صورت فارسی و پاراگراف‌های پاسخ به زبان انگلیسی هستند. فایل encode.py حاوی سه متد encoder است. متد اول encode_en است که یک پاراگراف انگلیسی را به عنوان ورودی دریافت کرده و آن را encode می‌کند و وکتور مربوطه را باز می‌گرداند. این متد برای لود کردن داده به Milvus کاربرد دارد. متد دوم encode_fa است که کاربردی مشابه متد قبلی برای متون فارسی دارد. این متد در متد encode_cross استفاده می‌شود. متد encode_cross ابتدا encode_fa را روی پاراگراف ورودی اجرا کرده و با استفاده از Cross Lingual Projection پیاده‌سازی شد در قسمت قبلی، آن را تبدیل به امبدینگ معادل انگلیسی عبارت ورودی کرده و به عنوان ورودی در اختیار Milvus قرار می‌دهد.

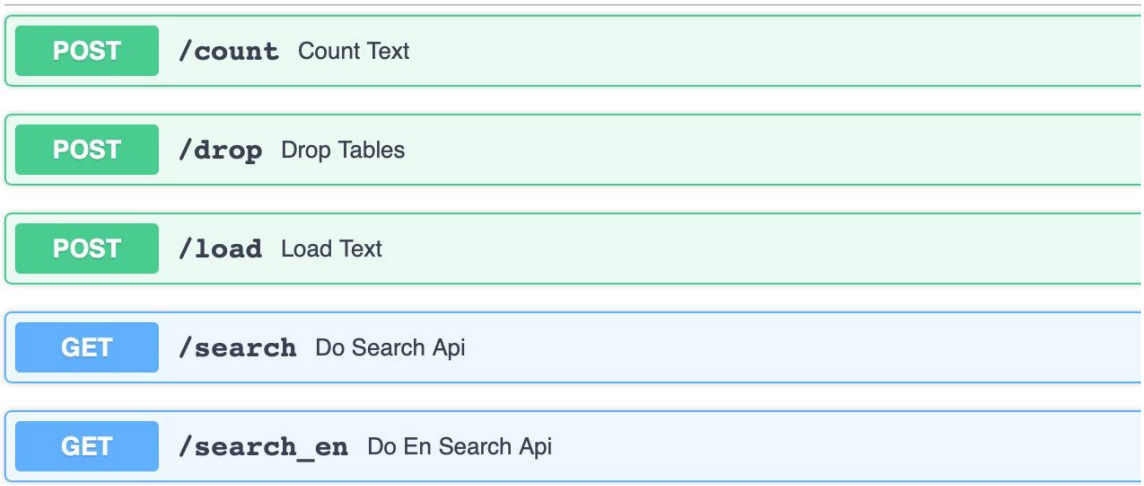
ساختار داده‌ی ورودی Milvus نیز هائز اهمیت است. داده‌ی ورودی Milvus به فرمت csv و با دو ستون است: title و text. title عنوان متن بازگردانده شده و text محتوای آن است.

ساختار title پاراگراف‌های برگردانده شده در پیاده‌سازی این پروژه به صورت زیر است:

ID#PARAG_ID#PAPER_TITLE

ID به آیدی مقاله، PARAG_ID به آیدی پاراگراف برگردانده شده از آن مقاله و PAPER_TITLE به عنوان مقاله‌ی بازگردانده شده اشاره دارد که در قسمت داده‌ی ارزیابی به آن‌ها اشاره شد. با مقایسه‌ی PAPER_ID با PARAG_ID برگردانده شده با مقادیر صحیح آن می‌توان روش پیاده‌سازی را ارزیابی کرد. به عنوان مثال پاراگراف صحیح از مقاله‌ی صحیح بهتر از پاراگراف نادرست از مقاله‌ی صحیح است و پاراگراف نادرست از مقاله‌ی صحیح به پاراگراف نادرست از مقاله‌ی نادرست برتری دارد.

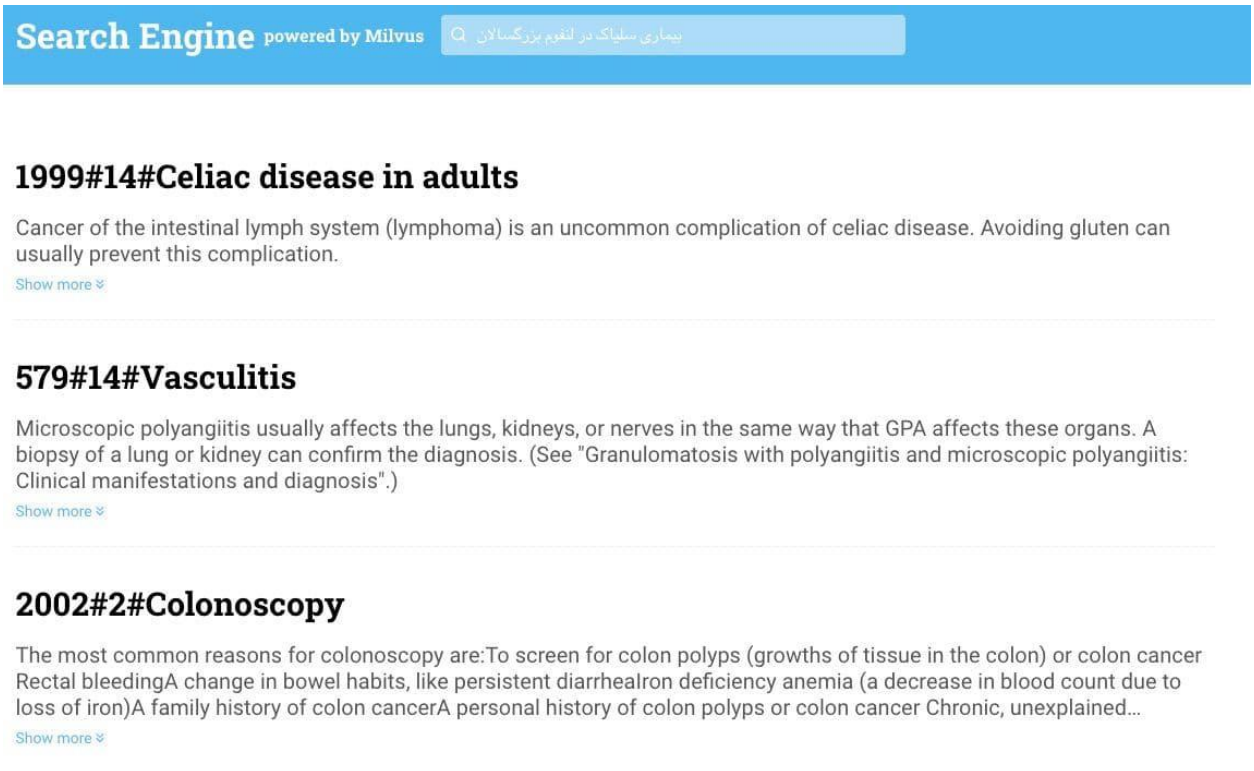
شکل زیر صفحه‌ی API جستجوی Milvus برای زبان‌های فارسی و انگلیسی را نشان می‌دهد. از load جهت لود کردن داده‌های به شکل csv استفاده می‌شود:



۵- سامانه‌ی نهایی و ارزیابی آن

۵-۱- سامانه‌ی نهایی

پس از طی مراحل قبلی، سامانه Mulvis فراهم شده و امکان جستجوی دو زبانه در آن در دو اندپوینت فارسی و انگلیسی فراهم شد که به وسیله آنها میتوان هم به زبان فارسی و هم به زبان انگلیسی برای جستجو استفاده کرد. تصویر زیر نمونه‌ای از جستجوی عبارت فارسی "بیماری سلیاک در لنفوم بزرگسالان" در سامانه را نشان می‌دهد. همان‌طور که مشخص است، سامانه به عنوان اولین نتیجه، پاراگراف مربوط به عبارت جستجو شده و همچنین مقاله آن را به عنوان اولین نتیجه بازگردانده است.



۵-۲- ارزیابی

برای اینکه بسنجیم سامانه با چه دقتی می‌تواند نتایج را گزارش کند، از معیار MRR استفاده کرده ایم. برای محاسبه این معیار، کافی است میانگین معکوس رتبه‌های نتیجه درست را برای تعدادی کوئری ارزیابی محاسبه کنیم. هرچه این مقدار بیشتر شود، کارایی مدل نیز بهتر خواهد بود. برای ارزیابی مدل، بردار امبدینگ ۳۰۴

پاراگراف استخراج شده (به دلیل کمبود وقت به ناچار تعداد کمی از پاراگراف‌ها برای بازیابی مورد استفاده قرار گرفتند اما این امکان کاملا وجود دارد که تمام پاراگراف‌ها برای بازیابی استفاده شوند) و روی ۳۰۴ کوئری دست‌ساخته (که در قسمت های قبلی گزارش آمده است) نتایج بازیابی بدست آمد. مقدار MRR برای جستجوی انگلیسی برابر با ۰.۴۵ و برای جستجوی فارسی برابر با ۰.۱۴ شده است. در جستجوی انگلیسی و فارسی به ترتیب در ۹۸ و ۲۲ مورد از ۳۰۴ کوئری، اولین نتیجه گزارش شده توسط سامانه برابر با پاراگراف و مقاله درست بوده است. همچنین به ترتیب ۷۰ و ۲۰۱ مورد در ۹ نتیجه گزارش شده وجود نداشته است. (نکته قابل توجه این است که تنها ۹ نتیجه اول گزارش شده در نظر گرفته شده است و اگر نتیجه مطلوب در ۹ موردی که توسط سامانه گزارش شده است نبوده باشد، مقدار آن را برای محاسبه MRR برابر با ۰ در نظر گرفته ایم)

علت کم بودن مقدار MRR برای آزمایش فارسی، می‌تواند دو مورد باشد:

۱) کم بودن داده فارسی برای یادگیری مدل fasttext و در نتیجه ضعیف بودن نتیجه امبدینگ فارسی(همان‌طور که در مقدار loss مدل fasttext فارسی در مقایسه با مدل انگلیسی در تصویر موجود در قسمت توضیحات fasttext قابل مشاهده است).

۲) ضعیف بودن نحوه انجام Cros Lingual Projection انجام شده. متاسفانه به دلیل کمبود وقت، امکان آزمایش روش های مختلف برای این قسمت وجود نداشت.

۶- جمع بندی

همانطور که ذکر شد، در دنیای امروزی حجم داده‌ها و اطلاعات پزشکی به سرعت در حال افزایش است و هدف از این پروژه طراحی سامانه‌ای دوزبانه جهت پاسخ به سوالات پزشکی با استفاده مقالات پزشکی است. این سامانه، سوالات کاربران به زبان‌های فارسی یا انگلیسی را دریافت کرده و پاسخ مناسب را به آنان به زبان انگلیسی برمی‌گرداند. در این گزارش ابتدا به داده‌ی جمع‌آوری شده برای این مسئله و پیش پردازش‌های انجام شده روی آن پرداختیم. سپس نحوه‌ی تولید داده‌ی ارزیابی را بررسی کردیم. در قسمت‌های بعد، روش‌های مطرح شده جهت حل این مسئله، شامل روش‌های کنار گذاشته شده و یا برگزیده را معرفی و بررسی کردیم. در نهایت، سامانه‌ی نهایی را نمایش دادیم و عملکرد آن را بررسی کردیم.

۷- کارهای آینده

یکی از کارهایی که میتوان برای بهبود نتیجه استفاده کرد، این است که کوئری کاربر نیز توسط BERN2 مراحل annotate را بگذراند تا بتوان کلمات کلیدی کوئری کاربر را (مانند استفاده از idf)، در بردار امبدینگ کوئری تاثیر بیشتری داد. همچنین، مانند هر پروژه موجود در هوش مصنوعی، مشکل گرسنگی اطلاعات در این پروژه نیز وجود دارد که در صورت دست‌یابی به مقالات پزشکی بیشتر و بهتر، بهبود می‌یابد. همچنین، حجم متون تخصصی فارسی و انگلیسی استفاده شده جهت کمک به سامانه می‌تواند افزایش پیدا کند. به ویژه اینکه متون تخصصی دارای ویژگی‌ها و اصطلاحات خاص خود هستند و افزایش حجم آن‌ها به یادگیری هر چی بیشتر و بهتر مدل کمک می‌کند. در مورد همچنین می‌توان به بررسی بیشتر مدل‌های خانواده‌ی BERT جهت بهبود این پروژه پرداخت که از جمله آن می‌توان به ظرفیت بالقوه bioBERT اشاره نماییم.

٨- مراجع

Sung, M., Jeong, M., Choi, Y., Kim, D., Lee, J. and Kang, J., 2022. BERN2: an advanced neural biomedical named entity recognition and normalization tool. *arXiv preprint arXiv:2201.02080*.

https://www.uptodate.com/

https://github.com/dmis-lab/bern

https://github.com/google-research/bert/blob/master/multilingual.md

https://github.com/seominjoon/denspi

https://www.mediawiki.org/wiki/API:Main_page

https://fasttext.cc/

https://milvus.io/

https://github.com/milvus-io/bootcamp/tree/master/solutions/text_search_engine

https://www.researchgate.net/publication/2526963_Cross_Lingual_Medical_Information_Retrieval_through_Semantic_Annotation

In the name of Allah

Sharif university of technology
Department of Computer Engineering

Final project
Natural Language Processing

Title

Bilingual Medical Question Answering System Based on Articles

Professor
Dr. E. Asgari

Team members
Ali Reza Sahebi
Mohammad Ali Sadraei Javaheri
Zahra Yousefi
Mehdi Hemmatyar
Ali Safarpoor-Dehkordi

January - February 2022