بسم الله الرحمن الرحیم

دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

بازیابی پیشرفته‌ی اطلاعات
نیم‌سال دوم تحصیلی ۰۱-۰۰

پروژه‌ی پایانی
نوشته‌های مرتبط با سلامت/بیو

محمدمهدی ابوترابی ۹۸۱۰۵۵۵۷
یاسمن زلفی موصلو ۹۸۱۰۵۷۹۵
فاطمه عسگری ۹۸۱۰۵۹۲۱

توضیحات کلی پروژه

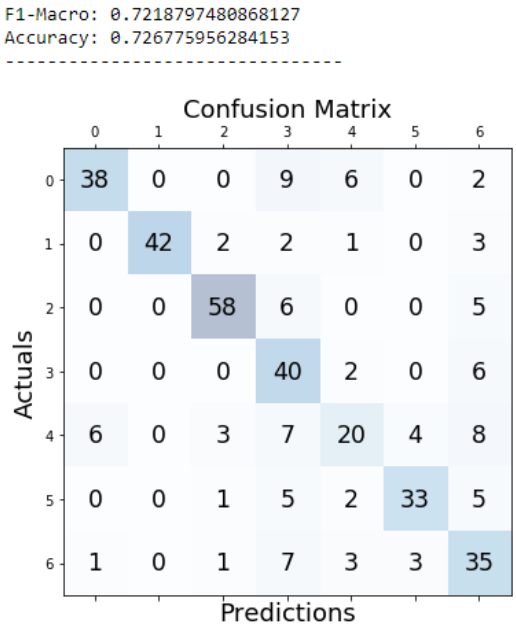
به صورت کلی پروژه از سه پروژه‌ی back-end و front-end و developement تشکیل شده که پروژه‌ی developement شامل همان بخش‌هایی است که در تمرین‌های طول ترم پیاده‌سازی شد و شامل نوتبوک‌هاست که مدل‌ها نیز از روی آن train شده است. پروژه‌ی back-end یک پروژه‌ی Django است و پروژه‌ی front-end نیز با استفاده از react پیاده‌سازی شده است.
قسمت‌هایی که به پروژه‌ی پایانی افزوده شده‌اند در ادامه به تفصیل توضیح داده شده‌اند:

دسته‌بندی و خوشه‌بندی

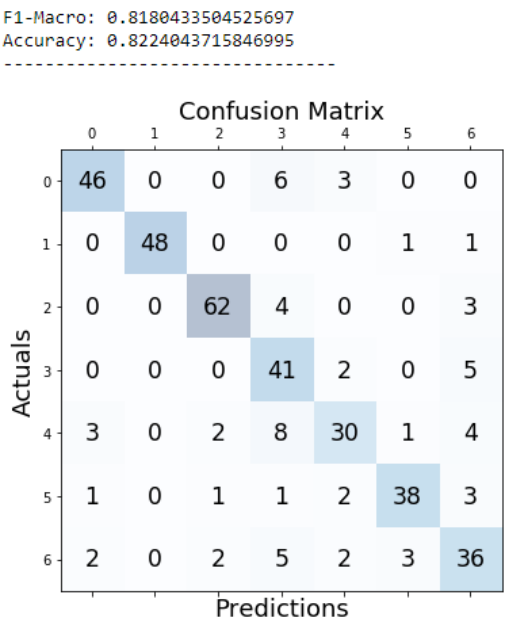
در بخش development در پوشه‌ی notebooks یک پوشه‌ی main_classification اضافه شده است که شامل بخش دسته‌بندی است. ابتدا دیتا را که شامل categoryهاست لود می‌کنیم و بعد یک امبدینگ fasttext از آن می‌گیریم که به نسبت ۱۹ به ۱ بین text و title میانگین می‌گیریم که برای هر داک یک امبدینگ به دست آید. حال که برای هر داک یک امبدینگ داریم، مقادیر x ما را تشکیل می‌دهند. مقادیر y نیز همان کتگوری‌هاست. داده‌ی train و test را جدا می‌کنیم و به سه روش دسته‌بندی را انجام می‌دهیم:

- روش naive bayes
- روش Logistic regression
- روش transformers

دو روش اول را به کمک کتابخانه‌ی scikit learn انجام دادیم. این دو روش همان روش‌های سنتی هستند که خروجی‌هایشان هم در زیر قابل مشاهده است. این مدل‌ها ترین و تست شده‌اند و برای پروژه‌ی back-end ذخیره شده‌اند.



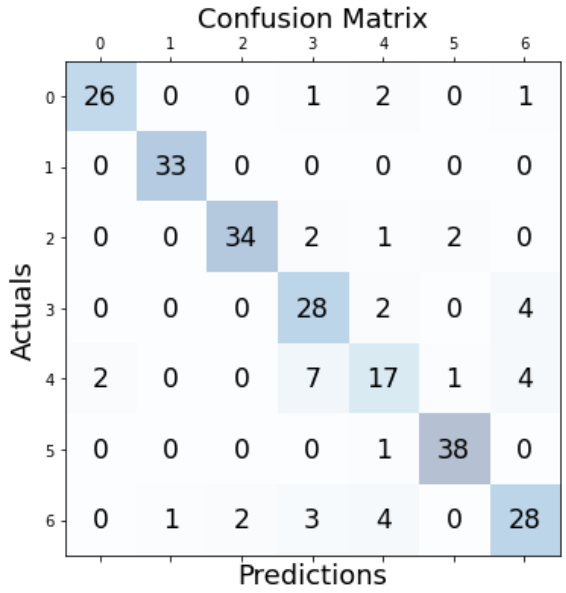
Naive Bayes



Logistic Regression

برای روش transformers نیز از یک مدل pre-trained فارسی به نام bigbird استفاده کردیم. سپس دیتاست‌ها را از روی دیتایمان ساختیم و مدلمان را ترین کردیم. این ترین را نیز به کمک google colab و تنها به اندازه‌ی 10 epoch انجام دادیم. بعد از ترین، قسمت تست را انجام دادیم و معیارها را به دست آوردیم. این مدل را هم برای استفاده در پروژه‌ی back-end ذخیره کردیم. خروجی ارزیابی برای این مدل به شرح زیر است:

```
F1-Macro: 0.8311190353650467
Accuracy: 0.8360655737704918
--------------------------------
```



Transformers

در نهایت نتیجه گرفتیم که مدل transformer از همه قوی‌تر است. بعد از آن مدل logistic regression و در آخر naive bayes.

برای بخش خوشه‌بندی باید بهبود انجام می‌دادیم که از روش‌های مختلفی مانند Birch و GussianMixture استفاده کردیم. در نهایت هم با عوض کردن وزن‌های امبدینگ (یعنی نسبت وزن title و text برای گرفتن میانگین امبدینگ) و normalize l2 کردن امبدینگ‌ها، بهبود حاصل شد. به طور مثال در تمرین چهارم که خوشه‌بندی را انجام داده بودیم Purity Score برابر 0.596138 شده بود ولی با این کارها به Purity score برابر 0.612162 رسیدیم.
نتایج کامل شامل تمامی scoreها و همچنین نمودارهای PCA در نوتبوک موجود است.
در پروژه‌ی back-end نیز یک پکیج classification داریم و یک پکیج clustering. به صورت کلی وقتی سرور بالا می‌آید تمام requirementهای لازم و مدل‌های لازم خوانده می‌شود و دیگر هنگام request آمدن چیزی لود نمی‌شود. در پکیج classification سه مدل classifier داریم که همین سه روش بیان شده در بالا هستند. هر روش از مدل ذخیره‌شده لود می‌شود. این روش‌ها با استفاده از مدل‌های مربوطه، پرسمانی که ورودی که می‌گیرند را می‌توانند predict کنن که متعلق به چه کلاسی است.
در رابط کاربری front نیز بخش دسته‌بندی به این شکل است که دو قسمت دارد که در بخش اول کابر پرسمان و مدلی که می‌خواهد با آن دسته‌بندی انجام شود را وارد کرده و خروجی را مشاهده می‌کند. در بخش دوم هم معیارها را نشان می‌دهیم.
برای خوشه‌بندی نیز داخل پروژه‌ی back-end یک پکیج clustering داریم که اینجا همان روش Kmeans است که مدل ذخیره‌شده را لود می‌کند و سپس می‌بیند پرسمانی که کاربر داده در چه خوشه‌ای قرار می‌گیرد و kتا نزدیک‌ترین داکهایی که در آن خوشه به این پرسمان قرار گرفته‌اند را به عنوان نتیجه برمی‌گرداند. در رابط کاربری هم دو بخش داریم که یکی پرسمان را گرفته و نتیجه را نشان می‌دهد و بخش دیگر معیارها و نتایج ارزیابی را نشان می‌دهد.

گسترش پرس‌وجو

برای هر چهار مدل بازیابی که داشتیم، یعنی fasttext و transformers و tfidf و booleen، همچنین برای روش elastic گسترش پرس‌وجو را انجام دادیم.

برای tfidf و fasttext و transformers از الگوریتم Rocchio برای گسترش پرس‌وجو استفاده کردیم چراکه امبدینگ برداری دارند. یک کلاس پدر به نام BaseModel نوشتیم که شامل یک تابع rocchio است که عملیات گسترش را انجام می‌دهد. در این الگوریتم ۱۰ تا داک نزدیک و دور را امبدینگ‌هایشان را می‌گیریم و سپس امبدینگ داک اصلی را به علاوه‌ی میانگین ده تا داک نزدیک منهای میانگین ده تا داک دور می‌کنیم و امبدینگ پس از گسترش را به دست می‌آوریم. حال از این امبدینگ به دست آمده برای عملیات بازیابی و جستجو استفاده می‌کنیم.
برای boolean هم ابتدا کمی مدل boolean را تغییر دادیم. در آن از امبدینگ fasttext کمک می‌گیریم. ابتدا یک دور با پرسمان اصلی جستجو را انجام می‌دهیم. سپس با استفاده از fastext برای هر کلمه، کلمات مشابهش را که similarityشان بین ۰/۸ تا ۰/۹۸ است در نظر می‌گیریم و سپس از بین کلمات مشابه یکی را رندوم برمی‌داریم و به جای کلمه‌ی اصلی می‌گذاریم. اینگونه کل پرسمان عوض می‌شود و روی پرسمان جدید جستجو را انجام می‌دهیم. این کار را ۵ بار تکرار می‌کنیم (در هر بار operatorهای بین کلمات مانند همان پرسمان اصلی است فقط خود کلمات تغییر می‌کنند.) و سپس بین نتایج ۶ پرسمان (یکی اصلی و ۵ تا مشابه) or گرفته و kتا از نتایج را به صورت رندوم برمی‌گردانیم.
برای elastic نیز یک گسترش پرس‌وجو زدیم که در ادامه در قسمت elastic توضیح خواهیم داد.

موتور جستجوی Elastic Search

برای موتور جستجوی خواسته‌شده از elastic search استفاده کردیم که برای اجرا ابتدا باید نصب شود. سپس اگر به درستی نصب شده باشد، کلاس هنگام لود شدن آن را بالا می‌آورد. یک تابع insert_data_and_create_index دارد که ایندکس مربوطه را می‌سازد و فرمت دیتاهایمان را درست کرده و داخل آن ایندکس قرار می‌دهد. یک بار این کار انجام شده و دیتایمان روی elastic ذخیره شده است و روش سریعی هم هست. یک تابع delete_index هم دارد که اگر خواستیم ایندکسی را حذف کنیم از آن می‌توانیم استفاده کنیم.
برای جستجوی پرسمان نیز ابتدا پیش‌پردازشی که در جاهای دیگر هم داشتیم را انجام می‌دهیم و سپس جستجو را انجام می‌دهیم. قابلیت گسترش پرس‌وجو را هم در این قسمت پیاده‌سازی کردیم بدین صورت که شبیه همان روش boolean کلمات را تغییر می‌دهد (به ازای هر توکن داخل پرسمان یک توکن جایگزین می‌کند) و سپس نتایج که برای پرسمان تغییر یافته به دست آمده را با نتایج پرسمان اصلی or می‌کند.
نکته: هنگام کوئری زدن به موتور جستجوی elastic به عنوان boost=2 می‌دهیم و به متن boost=1 که متن‌مان را هم در عنوان و هم در متن‌ها سرچ کند.

واسط کاربری مبتنی بر وب

یک پروژه‌ی بک‌اند با استفاده از Django و یک پروژه‌ی فرانت‌اند به وسیله‌ی فریم‌ورک React توسعه داده شده است که هر یک به صورت جداگانه به شکل local بالا می‌آیند و می‌توان از آن‌ها استفاده کرد. طریقه‌ی راه‌اندازی این دو پروژه در readme توضیح داده شده است.
پروژه‌ی بک‌اند یک سری api view دارد که آن‌ها را طراحی کرده‌ایم و فرانت به وسیله‌ی آن apiها اطلاعات خواسته شده را به کاربر نشان می‌دهد. مدل‌ها در همان ابتدای بالا آمدن پروژه‌ی بک‌اند لود می‌شوند. برای query retrieval که در همان تمرین سوم کلاس‌بندی کرده بودیم که مدل‌ها لود شوند و کار کنند و در اینجا هم از همان‌ها استفاده کردیم.
در بخش‌های قبلی قسمت‌های مختلف واسط کاربری اعم از خوشه‌بندی و دسته‌بندی و... توضیح داده شد. آخرین قسمت واسط کاربری Link Analysis است که برای هر دسته، مرتبط ترین و نامرتبط ترین جملات را برمی‌گردانیم. برای این قسمت، همان زمانی که داشتیم train می‌کردیم، نتایج را ذخیره کردیم و همان‌ها را برمی‌گردانیم.

توجه: تمامی مدل‌های استفاده شده در این لینک بارگذاری شده‌اند.

ارزیابی MRR

در نهایت همانطور که از ما خواسته شده بود، همان روش‌های بازیابی اطلاعات را (boolean و tfidf و fasttext و transformers) این بار با استفاده از گسترش جستجو امتحان کردیمو نتایج ارزیابی برای ۱۰ کوئری به ازای هر مدل در این داک آورده شده است. نتایج تمرین سوم که بدون گسترش پرس‌وجو بود نیز به طور کامل در گزارش تمرین سوم وجود دارد. مقادیر MRR برای هر روش به شرح زیر است:

- روش boolean: 0.438439
- روش tf-idf: 0.9666666667
- روش fasttext: 0.85111
- روش transformer: 0.88611

که نسبت به حالت بدون گسترش جستجو (تمرین سوم) در مدل‌های tfidf و fasttext بهبود ایجاد شده است.
همچنین برای روش elastic نیز در همین داک ارزیابی MRR انجام شده که مقدار آن 0.8964266667 است که نتیجه‌ی بسیار خوبی است و از اغلب روش‌های دیگر بهتر شده است.