بازیابی پیشرفته اطلاعات

فاز دوم پروژه

اعضای گروه:

حامد علی‌محمدزاده - ۹۶۱۰۲۰۲۹

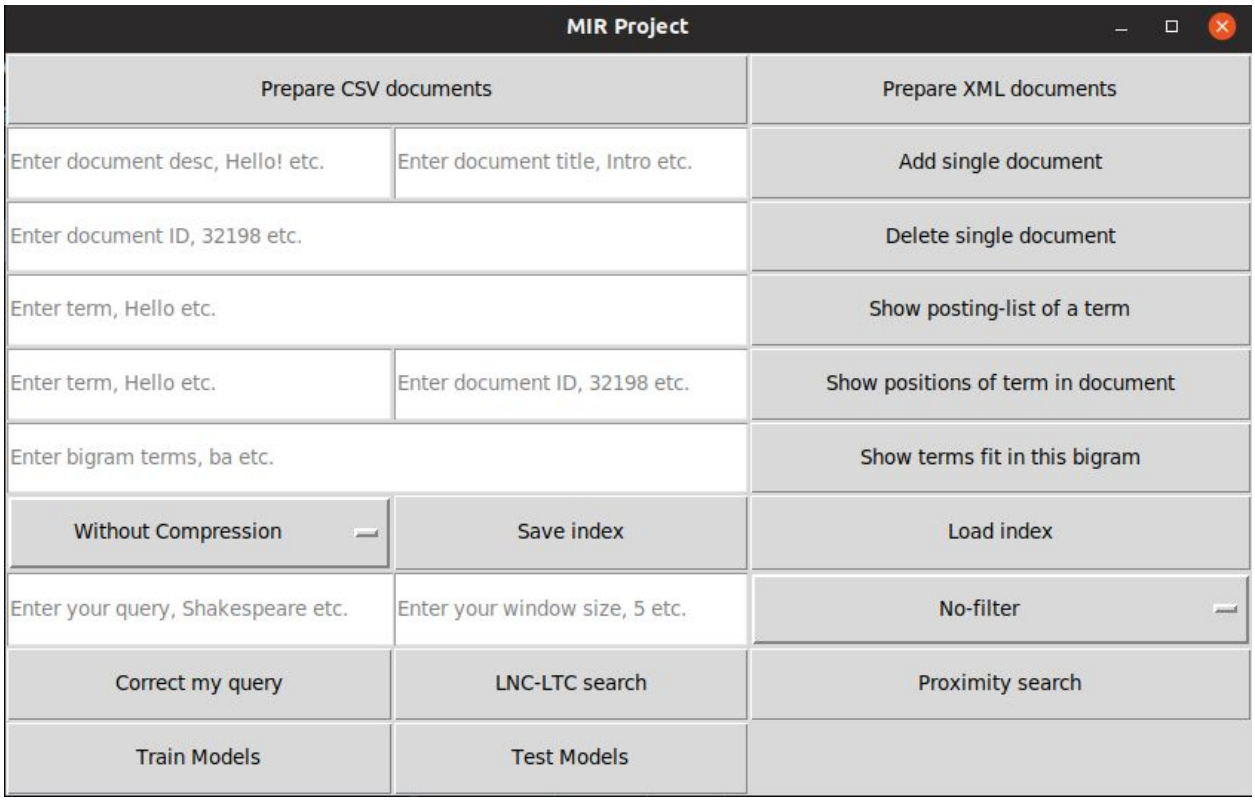
حمیدرضا هدایتی - ۹۶۱۰۹۹۳۹

آرمین سعادت بروجنی - ۹۶۱۰۵۸۲۹

آذر ۱۳۹۹

بخش صفر: معرفی کنسول

در فاز اول برای ایجاد رابط کاربری مناسب و امکان استفاده و تست کردن تمام امکانات سامانه به شکل راحت از کتابخانه tkinter استفاده شد. در این فاز نیز فیچرهای train و test مدل‌ها و همچنین فیلتر برای سرچ به این رابط اضافه شد که در شکل زیر مشاهده می‌شود.



برای استفاده از امکانات دسته‌بندی در ابتدا باید دکمه Train Models انتخاب شده و با انتخاب مجموعه داده مورد نظر مدل‌ها train شوند. با اتمام این فرآیند، پنجره زیر ظاهر شده و تمام موفقیت‌آمیز آموزش را خبر می‌دهد.

پس از آن برای ارزیابی مدل‌ها می‌توان دکمه Test Models را زد و با انتخاب مجموعه داده مورد نظر فرایند test انجام شود. پس از پایان رسیدن این فرآیند، نتایج ارزیابی دسته‌بندها بر اساس معیارهای گفته شده نمایش داده می‌شود.

توجه شود که فرآیندهای ذکر شده روی هر ۴ مدل classifier اعمال می‌شود.

دکمه دیگری که اضافه شده در شکل بالا No-filter است که میتوان مقدار آن را به مقادیر Popular و Not-Popular تغییر داد. در هر حالت عملیات سرچ به صورت گفته شده خواهد بود.

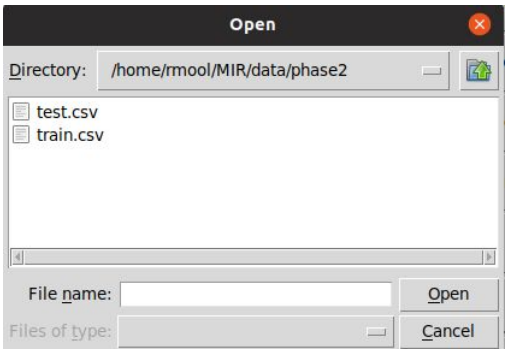
- اگر No-filter باشد، عملیات سرچ روی تمام مجموعه داده‌ها انجام می‌شود بدون توجه به دسته‌ی تعیین شده برای آن‌ها. یعنی دقیقا مطابق فاز ۱ است پس به نوعی backward compatible است.

- اگر گزینه Popular انتخاب شود، عملیات سرچ روی داده‌های پربازدید که همان داده‌های عضو دسته ۱ هستند صورت می‌گیرد.

- اگر گزینه Not-Popular انتخاب شود، عملیات سرچ روی داده‌های کم‌بازدید که همان داده‌های عضو دسته ۱- هستند صورت می‌گیرد.

نکته مهم این است که داده‌های فاز ۱ در هنگامی که در سامانه لود می‌شوند (با انتخاب گزینه Prepare CSV document) دسته‌بندی می‌شوند. بنابراین مهم است که قبل از این کار مدل‌ها train شده باشند.

همچنین نحوه انتخاب مجموعه داده برای Train و Test نیز توسط کاربر از طریق رابط برنامه انتخاب می‌شود و کاملا داینامیک است. به شکل زیر:



بخش یک: پیاده‌سازی دسته‌بندها

پیش‌پردازش)

هر سطر از جدول یک داکیومنت در نظر گرفته شده است که از description و title تشکیل شده است. در گام اول متن هر داکیومنت با استفاده از توابع فاز ۱ و مشابه با همان فرآیند تمیز شد. به عبارتی stop word ها حذف شد، کلمات به حالت ساده خود رفته و سایر مواردی که در مرحله پیش‌پردازش متنی صورت می‌گیرد انجام گرفت.

همچنین ستون views به عنوان target value برای هر داکیومنت ذخیره شد.

فضای tf-idf)

پس از آن هر داکیومنت به فضای برداری tf-idf برده شد. به این معنا که هر داکیومنت به یک بردار k بعدی تبدیل شد که k تعداد ترم‌های موجود در کل مجموعه داده است. (مجموعه داده train) که مقدار i ام هر بردار، مقدار tf-idf ترم i ام در لغتنامه است. که tf-idf برابر (tf * log(N/df است.

البته برای بهینه کد زدن و اشغال کم‌تر حافظه فقط ترم‌هایی که در هر داکیومنت وجود داشتند ذخیره شدند. در نهایت هر داکیومنت به شکل یک dict در پایتون ذخیره شد.

پیاده‌سازی دسته‌بندها)

هر دسته‌بند در قالب یک class پایتون نوشته شده است. هر کدام از آن‌ها تعدادی تابع مشابه دارند که مهم‌ترین آن‌ها عبارتند از:

- **train(document_set):**

  یک مجموعه داده گرفته و با توجه به الگوریتم دسته‌بند برای یادگیری، پارامترها (و در صورت لزوم hyper-parameter ها) مدل را تعیین می‌کند.

- **predict(document):**

  یک داکیومنت گرفته و با توجه به مدل دسته‌بند به دست آمده، مشخص می‌کند که این داکیومنت به کدام دسته قرار دارد. یعنی تابع مقدار ۱ یا ۱- خروجی می‌دهد.

- **test(document_set):**

یک مجموعه داده به عنوان test set دریافت کرده و برای هر داکیومنت روی این مجموعه، تابع predict را صدا می‌زند تا دسته هر داکیومنت مشخص شود. سپس دسته پیش‌بینی شده هر داکیومنت با دسته واقعی آن مقایسه شده و conjugation table داده‌های تست به دست می‌آید.

- **get_accuracy():**

با توجه به conjugation table مقدار accuracy را خروجی می‌دهد.

- **get_precision_c1():**

با توجه به conjugation table مقدار precision را برای دسته پربازدید (که با ۱ مشخص می‌شود) خروجی می‌دهد.

- **get_precision_c2():**

با توجه به conjugation table مقدار precision را برای دسته کم‌بازدید (که با ۱- مشخص می‌شود) خروجی می‌دهد.

- **get_racall_c1():**

با توجه به conjugation table مقدار recall را برای دسته پربازدید (که با ۱ مشخص می‌شود) خروجی می‌دهد.

- **get_racall_c2():**

با توجه به conjugation table مقدار recall را برای دسته کم‌بازدید (که با ۱- مشخص می‌شود) خروجی می‌دهد.

- **get_F1_c1():**

با توجه به conjugation table مقدار F1_score را برای دسته پربازدید (که با ۱ مشخص می‌شود) خروجی می‌دهد.

- **get_F2_c2():**

با توجه به conjugation table مقدار F1_score را برای دسته کم‌بازدید (که با ۱- مشخص می‌شود) خروجی می‌دهد.

در ادامه به جزئیات پیاده‌سازی هر دسته‌بند پرداخته می‌شود.

1. Naive Bayes:

روش بیز ساده‌لوحانه یک مدل احتمالاتی است که از قانون بیز استفاده می‌کند و با به دست آوردن پارامترهای مناسب از روی داده‌های یادگیری، احتمال قرارگیری یک داکیومنت در یک دسته خاص را به دست می‌آورد.

در نهایت به ازای یک داکیومنت هر دسته که احتمال بیشتری داشت، به عنوان دسته‌ی پیش‌بینی شده انتخاب می‌شود.

علت وجود عبارت ساده‌لوحانه در نام‌گذاری آن به دلیل فرض‌های ساده‌کننده‌ای است که این مدل در نظر گرفت است. به این

صورت که احتمال حضور داکیومنت d در یک دسته را برابر با ضرب احتمال حضور کلمات داکیومنت d در آن دسته قرار داده است. به عبارتی بین حضور کلمات مختلف در یک داکیومنت یا دسته خاص استقلال قائل شده است.

رابطه بیز به شکل زیر است:

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)}$$
$$P(c|d) \propto P(c)P(d|c)$$

که با فرض استقلال نتیجه می‌دهد:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

که t_k توکن‌های موجود در داکیومنت d است. پس تعداد تکرار هم مهم است.

عبارت بالا به دلیل ضرب تعداد زیادی عدد کوچکتر از ۱ به صفر بسیار نزدیک می‌شود که در انجام محاسبات با کامپیوتر ممکن است به underflow منجر شود. برای همین لگاریتم عبارت بالا مد نظر قرار می‌گیرد و در نهایت دسته انتخابی آن دسته‌ای می‌شود که بیشترین مقدار را به خود اختصاص داده است.

پس با فرض داشتن مقادیر $P(t_k|c)$ و $P(c)$، داده d اینگونه دسته‌بندی می‌شود:

$$c_{\text{map}} = \underset{c \in \mathbb{C}}{\text{argmax}} \; [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

پارامترهای این مدل از روی داده‌های یادگیری مطابق با روابط زیر حساب می‌شوند:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}} \qquad \hat{P}(c) = \frac{N_c}{N}$$

یک مشکل این مدل این است که اگر در داکیومنت d کلمه‌ای آمده باشد که احتمال حضور آن در دسته c صفر تخمین زده شده باشد، احتمال آن دسته برای داکیومنت d صفر مطلق می‌شود هرچند که ممکن است کلمات مرتبط بسیاری داشته باشد. برای جلوگیری از این حالت روش Add_one smoothing استفاده می‌شود که عملا فرض می‌کند هر term حداقل یک بار در هر دسته ظاهر شده است. در این صورت پارامترها از رابطه زیر به دست می‌آیند:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

در کد نیز از همین معادله استفاده شد. با صدا زدن تابع train و پاس دادن مجموعه داده یادگیری، این پارامترها به دست آمده و پس از آن با صدا زدن تابع predict و با استفاده از prediction rule گفته شده در بالا دسته هر داکیومنت تعیین می‌شود. در ضمن از تمام داده train استفاده شد.

3. k-NN:

شرح الگوریتم:

الگوریتم k-NN یک الگوریتم یادگیری supervised است. برای پیدا کردن دسته یک data point این الگورتیم ابتدا kتا نزدیکترین data point به داده مورد نظر را پیدا کرده و طبق اینکه بیشتر به چه دسته‌ای تعلق دارند داده مورد نظر را به آن دسته اختصاص می‌دهد. برای محاسبه فاصله بین دو داده که در اینجا بردارهایی n بعدی هستند روش‌های مختلفی می‌توان استفاده کرد، فاصله اقلیدسی، منهتنی و مینکوفسکی، در اینجا ما از روش
منهتنی استفاده کرده‌ایم.

نحوه پیاده‌سازی

کلیت پیاده‌سازی این الگوریتم شامل این ۳ مرحله است:

1. محاسبه فاصله یک داده تست با همه‌ی داده‌های آموزشی برای پیدا کردن نزدیکترین‌ها
2. مرتب‌سازی فاصله‌ها و پیدا کردن kتا از نزدیکترین داده‌ها
3. پیدا کردن پرتکرارترین لیبل. این لیبل به عنوان دسته‌ی داده‌ی تست قرار می‌گیرد.

توضیح پیاده‌سازی توابع موجود در class دسته‌بند (بخش یک)

- **train(document_set):**

با توجه به این که k-NN فرایند یادگیری انجام نمی‌دهد در این تابع صرفا داده‌های آموزشی به همراه برچسب هرکدام دریافت می‌شوند و پس از یک پیش پردازش که لیبل را به عنوان درایه‌ی آخر بردارهای داده‌ی آموزشی قرار می‌دهد، آن‌ها را در مدل ذخیره می‌کند.

- **predict(document):**

در این تابع مراحل یک تا سه توضیح داده در بالا پیاده سازی شده‌اند و در پایان برای هر داده‌ی ورودی یک دسته اختصاص می‌یابد. این تابع ابتدا یک پیش پردازش روی داده‌ی تست ورودی انحام می‌دهد. به این شکل که بردارها را طوری بازسازی می‌کند

که شامل همه‌ی ترم‌های دیکشنری باشد. در صورتی که یک ترم وجود داشته باشد مقدار آن صف قرار داده می‌شود و اگر یک ترم در داده‌ی تست باشد که در دیکشنری وجود نداشته باشد حذف می‌شود.

- **test(document_set):**

مانند توضیحات داده شده در بخش یک.

پیدا کردن k بهینه

این بخش در تابع train پیاده سازی شده است. ابتدا ۳۰ درصد از داده‌های آموزشی به طور رندوم انتخاب شده و ۱۰ درصد از آن‌ها به عنوان داده validation به طور رندم جدا می‌شد. پس از ذخیره داده آموزشی در مدل، به ازای مقادیر مختلف $k = 1, 5, 9$ مدل تست می‌شود و آن مقدار k که منجر به بیشترین accuracy شود به عنوان k بهینه انتخاب می‌شود.

به دلیل اردر زمانی k-NN که به خاطر محاسبه فاصله هر داده تست از مجموعه آموزشی رخ می‌دهد، از ۳۰ درصد از داده‌های آموزشی استفاده کردیم و از ۱۰ درصد آن به عنوان داده validation استفاده کردیم. انتخاب داده‌ها به طور رندوم انجام می‌شود.

نتیجه:

K بهینه: ۵

accuracy: ۰.۵۳

```
Accuracy for 1-NN: 0.5175438596491229
Accuracy for 5-NN: 0.5350877192982456
Accuracy for 9-NN: 0.4649122807017544
Selected K for K-NN is 5 with accuracy of 0.5350877192982456
```

4. SVM:

هدف از این الگوریتم ایجاد یک مرز بین مجموعه داده‌هاست. این الگوریتم یک روش supervise است. نحوه کار آن به این شکل است که تعدادی داده برچسب زده شده به آن داده می‌شود و در نهایت یک hyperplane در فضای برداری مسئله ساخته می‌شود که مرز بین دو دسته‌ی مختلف از دادگان است. ابر صفحه‌ای بهتر است که margin به نسب بزرگتری از داده‌های هر دسته داشته باشد.

برای پیاده سازی این روش از دسته‌بند SVC موجود در پکیج sklearn.svm استفاده کرده‌ایم. برای train کردن ابتدا با استفاده از تابع fit_transform در DictVectorizer داده‌های آموزشی ورودی را تبدیل به وکتورهایی می‌کنیم که بتوان برای فیت کردن مدل از آن استفاده کرد. زیرا ممکن است وکتوها ترم‌هایی داشته باشند که در دیکشنری موجود نباشد و برعکس. سپس مدل را train میکنیم.

پیدا کردن C بهینه

این بخش در تابع train پیاده سازی شده است. از ۱۰ درصد از داده‌های آموزشی به عنوان داده validation استفاده شده است. پس از fit کردن مدل با ۹۰ درصد از داده‌های آموزشی، به ازای مقادیر مختلف C = 0.5, 1, 1.5, 2 مدل تست می‌شود و آن مقدار C که منجر به بیشترین accuracy شود به عنوان C بهینه انتخاب می‌شود.

نتیجه:

C بهینه: ۲

accuracy: ۰.۶۷

```
Accuracy for SVM with C = 0.5 is 0.6069868995633187
Accuracy for SVM with C = 1 is 0.6637554585152838
Accuracy for SVM with C = 1.5 is 0.6681222707423581
Accuracy for SVM with C = 2 is 0.6724890829694323
Selected C for SVM is 2 with accuracy of 0.6724890829694323
```

5. Random Forest:

این الگوریتم به این صورت کار می‌کند که تعدادی decision tree روی مجموعه داده‌های یادگیری ساخته و جواب نهایی را میانگین جواب این درخت‌ها در نظر می‌گیرد.

بخش اصلی کد این بخش توسط کتابخانه sklearn پیاده‌سازی شد. به این صورت که تابع predict کلاس RandomForest از یک classifier که sklearn در اختیار گذاشته است استفاده می‌کند و خروجی ۱ یا ۱- می‌دهد و مابقی کارها مشابه سایر الگوریتم‌ها است.

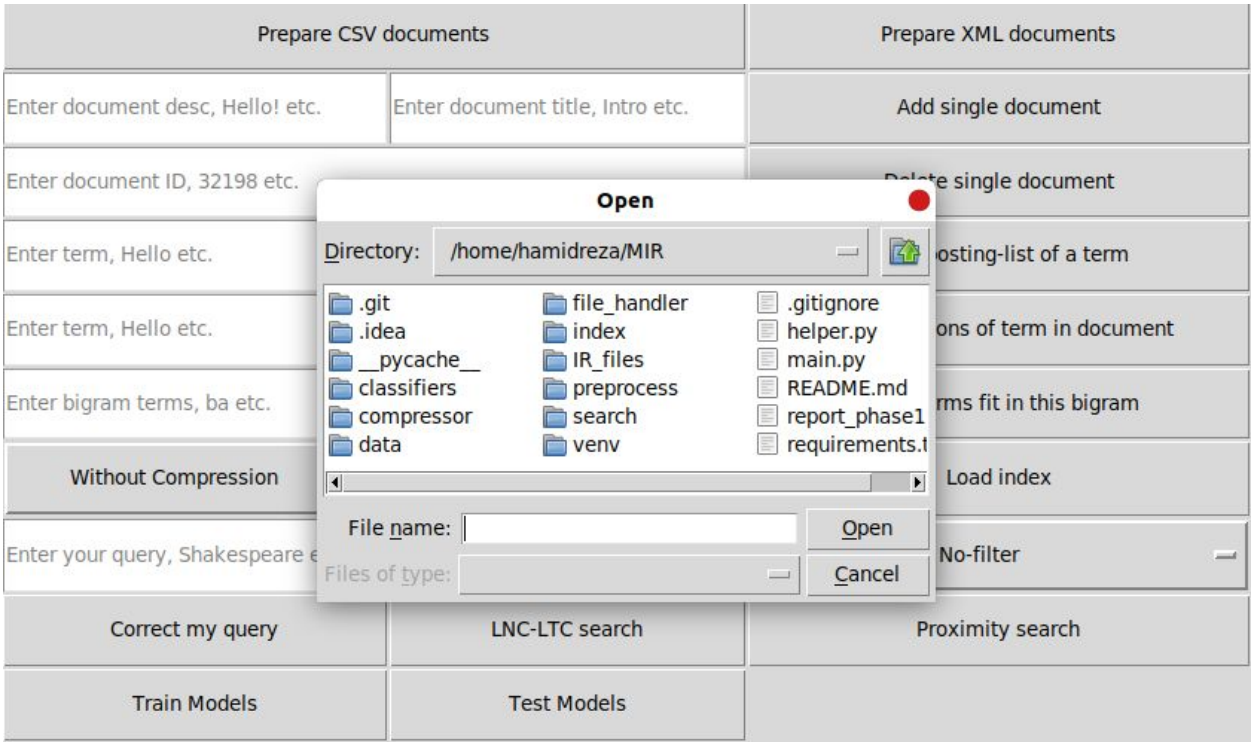
این classifier در هنگام یادگیری (در تابع train) روی داده‌ها fit می‌شود. که برای اینکار می‌توان یک عمق به آن به عنوان ورودی داد که عملا عمق decision tree ها را مشخص میکند. که هر چه این عمق بیشتر باشد دسته‌بند بیشتر روی داده یادگیری fit می‌شود و حتی ممکن است به overfitting منجر شود. در ضمن با افزایش عمق، زمان یادگیری نیز افزایش می‌یابد. با بررسی چندین حالت و دیدن نتایج، مقدار ۳۰ برای عمق انتخاب شد.

برای آموزش این دسته‌بند از تمام مجموعه داده یادگیری استفاده شد.

بخش دو: بهبود سیستم بازیابی اطلاعات فاز اول پروژه

در این بخش ما تلاش کردیم که کدهایی که برای فاز یک زده بودیم را با داده‌های Train شده برچسب بزنیم، در این راستا نخست فرض می‌کنیم که کاربر در رابط کاربری با کلیک کردن بر روی Train Models و انتخاب داده Train نخست مدل را Train می‌کند.

بعد از آن با اضافه کردن داده‌هایی که می‌خواهیم روی آن‌ها Search را انجام دهیم، داده‌ها با استفاده از SVM Classifier برچسب‌گذاری می‌شوند و این برچسب‌ها در دیکشنری Label ذخیره می‌شوند:



بعد از انتخاب فایل ted_talks.csv، داده‌ها به وسیله Trainی که انجام شده برچسب‌گذاری می‌شوند.

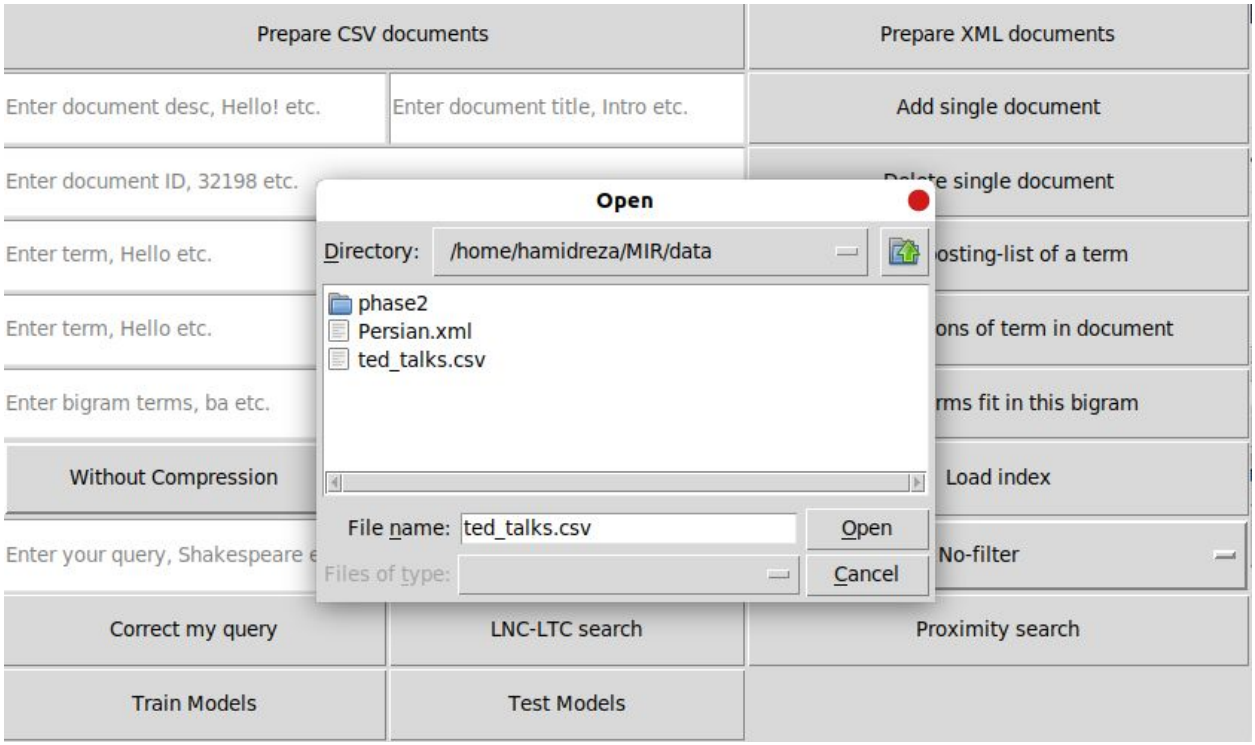
در گذشته می‌توانستیم با سرچ کردن یک عبارت، داکیومنت‌های مرتبط به آن عبارت را پیدا کنیم، هم اکنون گزینه جدیدی به فیلد سرچ کردن اضافه شده که در تصویر هم مشاهده می‌کنید (که گزینه پیش‌فرض آن، No-filter است و به Popular و Not Popular هم می‌تواند تبدیل شود)

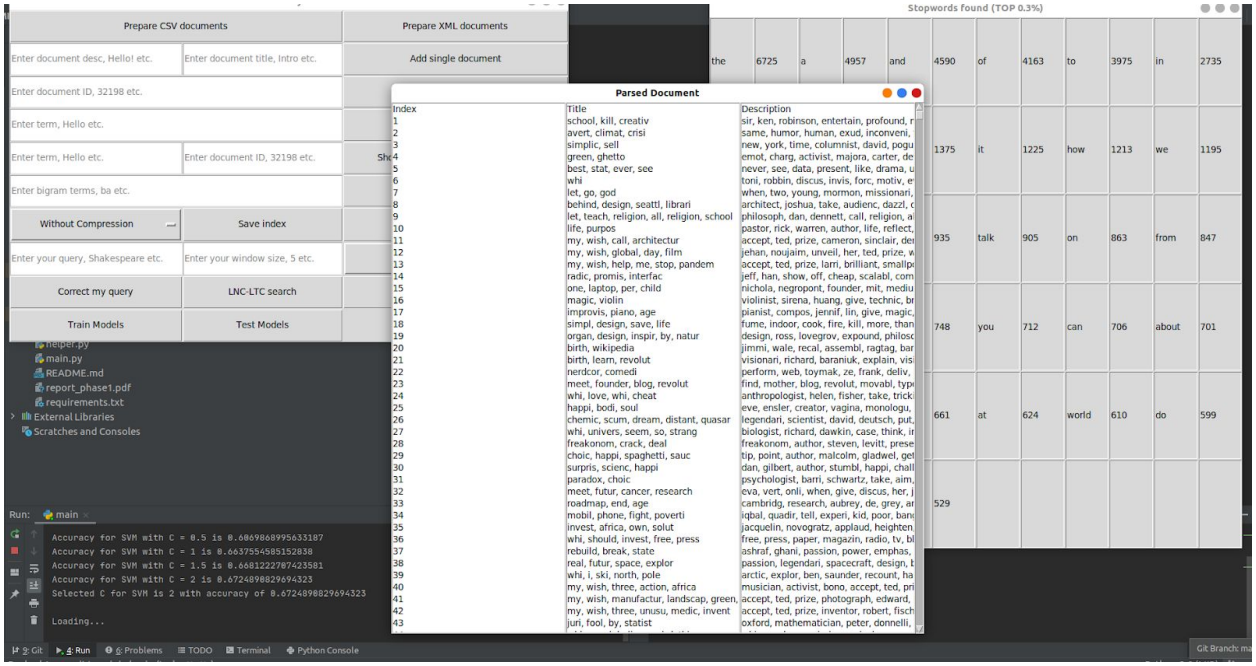
برچسب‌گذاری با استفاده از دسته بند SVM انجام شده‌است زیرا SMV هم accuracy بهتری به ارمغان می‌آورد و هم فرآیند Testing سریع تری دارد که باعث می‌شود برچسب‌گذاری سریع‌تر انجام شود.

پس از برچسب‌گذاری داده‌ها، موقع ذخیره‌سازی و بارگذاری، اعمال روی دیکشنری label هم اعمال می‌شود در نتیجه می‌توانیم هر بار مجبور به Train کردن مدل نشویم.

1. Train Models

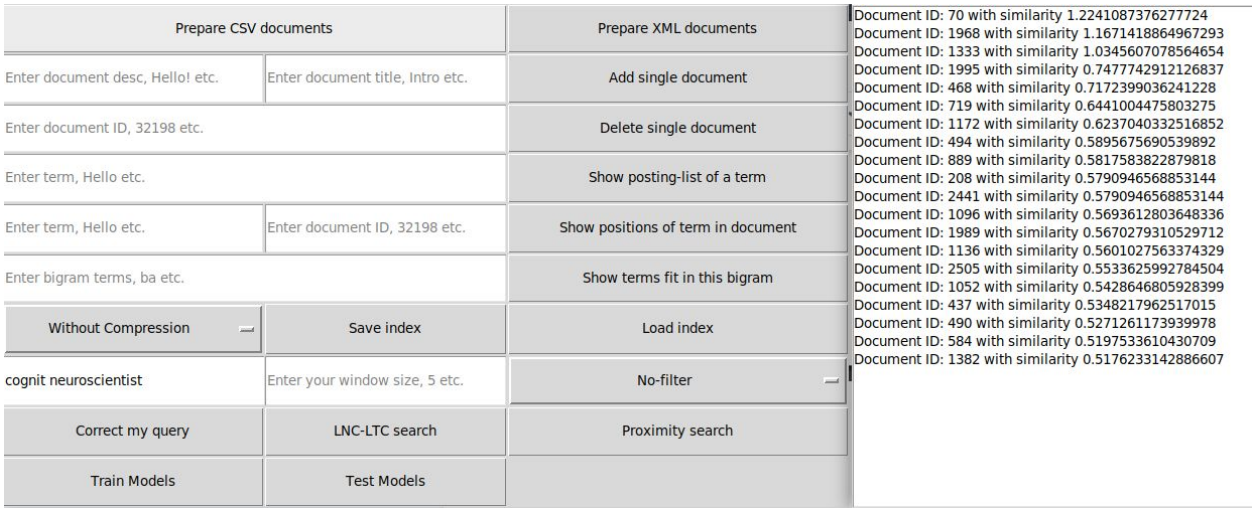
2. Prepare Documents (برچسب گذاری اینجا انجام می‌شود)



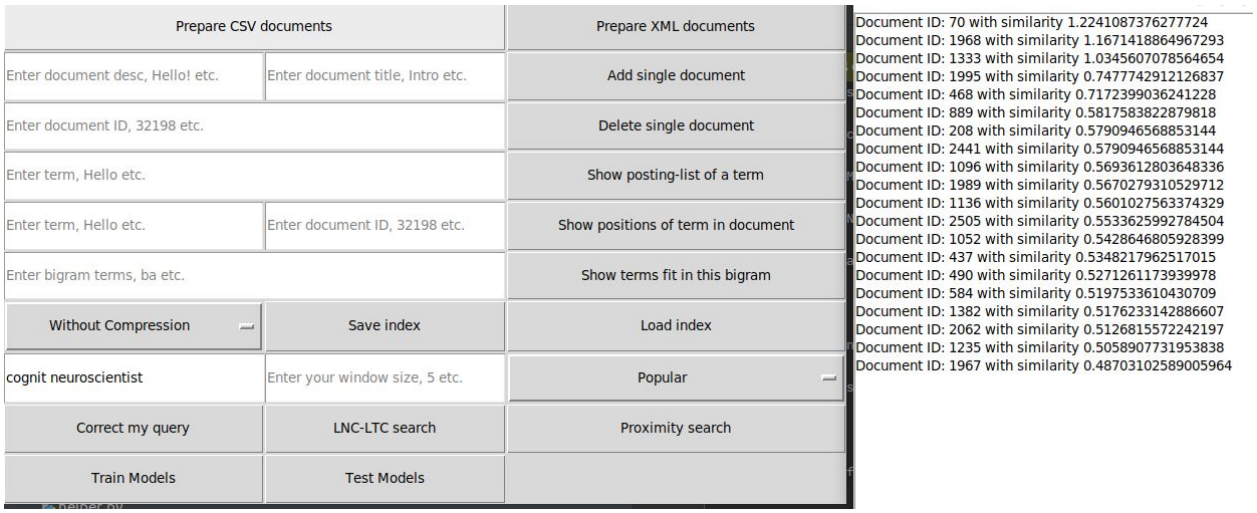


3. Phrase Searching

بدون فیلتر:

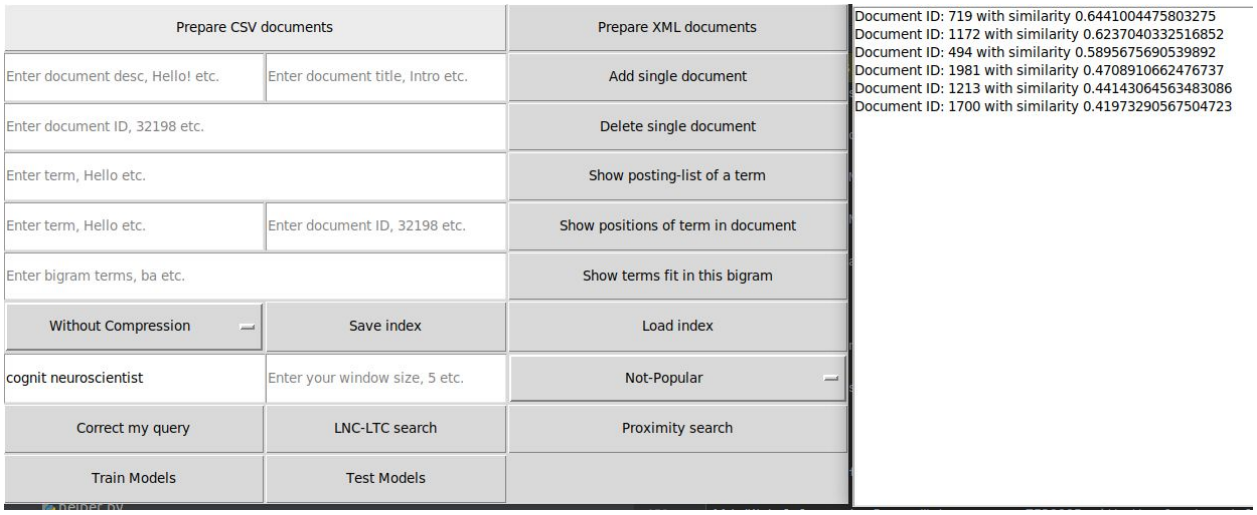


فیلتر Popular:



اگر دقت کنیم می‌بینیم که docuement id 70 به عنوان یک نتیجه popular نشان داده شده که در داده‌ها نزدیک به ۲ میلیون view دارد که نشان می‌دهد برچسب‌گذاری به صورت منطقی‌ای انجام داده شده است.

فیلتر Not-Popular:



با تست کردن این موارد با Proximity search هم به نتایج مشابه خواهیم رسید.

بخش سوم: ارزیابی نهایی

شرح روش‌های ارزیابی)

با اجرای تابع test در هر مدل دسته‌بند، conjugation table برای دسته‌های ۱ و ۱- محاسبه و ذخیره می‌شود. Conjugation table به شکل زیر است:

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

دسته ۱ همان دسته پربازدید است که با c1 نشان داده می‌شود.

دسته ۱- همان دسته کم‌بازدید است که با c2 نشان داده می‌شود.

با توجه به جدول بالا نحوه محاسبه هر یک از معیارهای ارزیابی توضیح داده می‌شود.

- Accuracy = $(TP + TN) \div (TP + FP + TN + FN)$
- Precision_c1 = $(TP) \div (TP + FP)$
- Precision_c2 = $(TN) \div (FN + TN)$
- Recall_c1 = $(TP) \div (TP + FN)$
- Recall_c2 = $(TN) \div (TN + FP)$

رابطه F1 هم به شکل زیر است:

$$F_1 = \frac{1}{\frac{1}{2}\frac{1}{P} + \frac{1}{2}\frac{1}{R}} = \frac{2PR}{P+R}$$

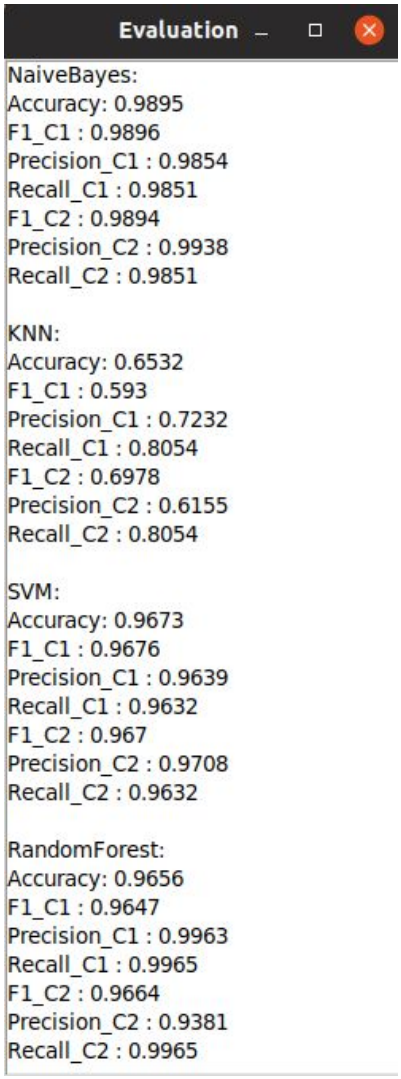
که R همان Recall و P همان Precision است.

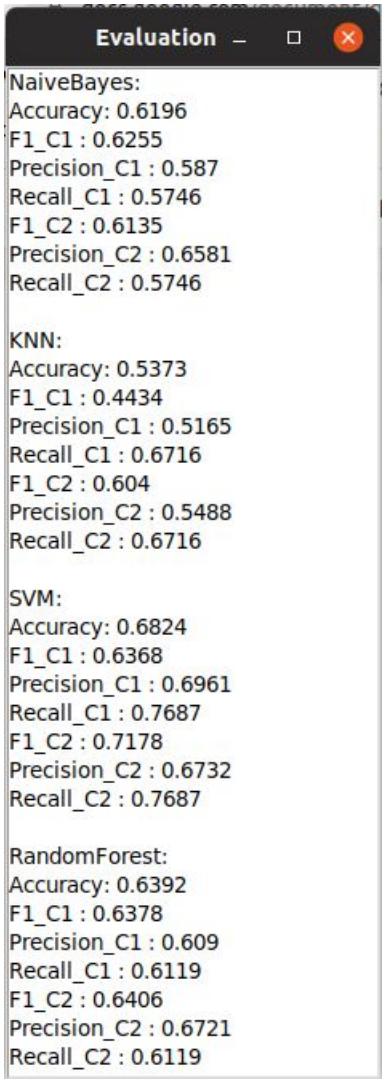
برای به دست آوردن F1 هر دسته از R و P همان دسته استفاده می‌شود.

گزارش نتایج)

نتایج به دست آمده از تست کردن هر ۴ دسته‌بند روی داده‌های آموزش و آزمون به تعبیر هر ۴ روش ارزیابی در شکل زیر آمده است که خروجی رابط کاربری است.

سمت راست مربوط به داده‌های آزمون و سمت چپ مربوط به داده‌های آموزش است.





که اعداد تا ۴ رقم اعشار به بالا گرد شده‌اند.

اطلاعات بالا در جدول زیر قابل مشاهده است.

جدول زیر مربوط به داده‌های آموزش است:

	Accuracy	F_c1	P_c1	R_c1	F_c2	P_c2	R_c2
Naive Bayes	0.9895	0.9896	0.9854	0.9851	0.9894	0.9938	0.9851
5-NN	0.6532	0.593	0.7232	0.8054	0.6978	0.6155	0.8054
SVM(C=1.5)	0.9673	0.9676	0.9639	0.9632	0.967	0.9708	0.9632
RandomForest	0.9656	0.9647	0.9963	0.9965	0.9664	0.9381	0.9965

جدول زیر مربوط به داده‌های آزمون است:

	Accuracy	F_c1	P_c1	R_c1	F_c2	P_c2	R_c2
Naive Bayes	0.6169	0.6255	0.587	0.5746	0.6135	0.6581	0.5746
5-NN	0.5373	0.4434	0.5165	0.6716	0.604	0.5488	0.6716
SVM(C=1.5)	0.6824	0.6368	0.6961	0.7687	0.7178	0.6732	0.7687
RandomForest	0.6392	0.6378	0.609	0.6119	0.6406	0.6721	0.6119

همانطور که ملاحظه می‌شود به طور کلی عملکرد دسته‌بندها روی مجموعه آموزش با اختلاف بهتر است که منطقی هم هست چرا که روی همان مجموعه هم fit شده‌اند.

اما ملاحظه می‌شود که در روش k-NN عملکرد روی مجموعه داده آموزش نیز خیلی بالا نیست.

دلیل این امر این است که برای آموزش k-NN از حدود ۳۰ درصد داده‌های آموزش استفاده شد اما نتیجه بالا ارزیابی k-NN روی تمام مجموعه

داده را نشان می‌دهد. یعنی k-NN هم روی تمام داده آموزش تست شد در صورتی که فقط روی ۳۰ درصد آن train شد.

در نهایت ملاحظه می‌شود که accuracy دسته‌بند SVM روی داده‌های آزمون از همه بیشتر است.

بنابراین از این دسته‌بند برای دسته‌بندی داده‌های فاز ۱ استفاده شد.

پایان!