فایل ضمیمه : hw4.ipynb

فایل خروجی دیتابیس پرسیکا به صورت یک csv دانلود شده. هر ردیف اطلاعات جدول اصلی ۷ ستون دارد که از این ۷ ستون من به دو ستون تیتر خبر (title) و متن (text) به عنوان متغیرهای مستقل و به ستون دسته بندی (category2) به عنوان متغیر وابسته (هدف) نیاز دارم. در فایل persica.csv اطلاعات ستون‌ها هر کدام در یک ردیف نوشته شده. خط‌ها را یکی یکی از ورودی فایل می‌خوانم و از هر ۷ تایی ، خطوط ۱ و ۲ و ۶ (شروع شماره گذاری از ۰) را به ترتیب به عنوان تیتر، متن و دسته‌بندی نگه می‌دارم. در نهایت متن و تیتر را باهم ادغام می‌کنم و حاصل کار یک ستون input_text هست که در ادامه‌ی کار مورد استفاده قرار خواهد گرفت. همچنین در ستون category متغیر هدف را داریم که ۱۱ گروه زیر را شما می‌شود. هدف مساله دسته بندی متن ها به ۱۱ کلاس زیر است.

{'سیاسی', 'فرهنگی', 'فقه و حقوق', 'مذهب' , 'آموزشی', 'اجتماعی', 'تاریخی', 'اقتصادی', 'بهداشتی', 'علمی' , 'ورزشی', 'ی'}

با استفاده از train_test_split.sklearn.model_selection دادگان را به دو گروه تست و آموزش با نسبت ۲ به ۸ تقسیم می‌کنم. از ۳ دسته‌بند روی این داده‌ها استفاده می‌کنم و نتایج کار هر کدام را (precision , recall, f-measure) گزارش خواهم کرد.

قبل از به کار گیری دسته‌بند باید روی متن ورودی به الگوریتم پردازش هایی انجام شود. بخش اول مربوط به تمیزکردن متن و استاندارد سازی متن است که در توضیحات دادگان پرسیکا آمده که این پیش پردازش‌ها انجام گرفته پس اینجا نیازی به آن نداریم. بخش دوم مربوط به تبدیل متن به بردار قابل استفاده در الگوریتم‌های دسته‌بندی است. مراحل کار به صورت زیر است :

۱. تولید یک ماتریس کلمه-سند به روش ساک کلمات : به ازای هر کلمه که در مجموعه دادگان وحود دارد یک ستون دارم و یک سند را به صورت ردیفی نمایش می‌دهم که اگر هر کدام از این کلمات در آن وجود داشت درایه نظیرش ۱ و درغیر این صورت ۰ خواهد بود. این مرحله را به کمک مدل CountVectorizer از sklearn.feature_extraction.text انجام می‌دهم. تا اینجا ساده ترین حالت تبدیل متن به بردار را پیاده کردم.
۲. تولید ماتریس tf-idf از روی مرحله قبلی : به جای درایه‌های صفر و یکی از فرکانس رخ داد کلمات استفاده کنیم تا کلماتی که در مجموع کم تکرارند ولی در متن خاصی بیشتر ظاهر می‌شوند در تشخیص کلاس آن سند اهمیت بیشتری پیدا کنند و بلعکس. برای تولید این ماتریس از مدل TfidfTransformer از sklearn.feature_extraction.text استفاده می‌کنم. ماتریسی که

حاصل این مرحله و مرحله قبلی به دست می‌دهند هر ابعاد 8799x60631 دارند که به ترتیب سطرها برابر تعداد دادگان آموزشی و ستون‌ها برابر دایره کلمات است.

۳. برای کوچکتر کردن بعد فضای ماتریسی که در آن کار می‌کنیم از روش تجریه ی SVD روی بردارهای به دست آمده از مرحله قبل استفاده می‌کنم. این مرحله بردارهایی به دست می‌دهد که ابعاد کوچکتر از 60631 مرحله قبلی دارند ولی حداکثر اطلاعات را با فشرده سازی در خود حفظ می‌کند. این مرحله همان پیاده سازی بخش LSA هست که با مدل TruncatedSVD از sklearn.decomposition اجرا شده. بعد بردارهای خروجی را ۱۰۰ تعین کردم و واضح است که با افزایش این عدد می‌توان دقت را افزایش داد تا جایی که دیگر افزایشی مشاهده نشود.

۴. بردارهای نظیر متن هایی که در مرحله قبل به دست آمد را نرمال می‌کنم تا همه اندازه های برابر داشته باشند و تاثیر اندازه ی بردار در تعین شباهت بین متن ها از بین برود. این بخش با مدل نرم ۲ از Normalizer از sklearn.preprocessing پیاده شده. در دسته‌بند اول (بیز ساده) به دلیل ماهیت بیزی این مدل داده‌های ورودی مدل نمی‌توانستند منفی باشند پس قبل از ورود به مرحله ۵ یک مقیاس پذیری دیگر روی داده‌ها انجام دادم تا همه بردارها با اندازه واحد درایه هایی بین ۰ و ۱ داشته باشند.

۵. یک دسته‌بند تعریف می‌کنم و به ازای ماتریس آموزشی x_train که از مرحله‌های قبل به دست آمده بود آن را آموزش می‌دهم. در این کار سه دسته‌بند استفاده شده به ترتیب بیز ساده، پرسپترون و SVM هستند که هرسه با کلاس‌های sklearn ساخته شده‌اند. برای SVM از هسته ی rbf استفاده کردم. بعد از آموزش هر مدل یک gridsearch اجرا کردم که به ازای پارامتر بعد خروجی تجزیه LSA (SVD) برای اعداد ۱۰۰ و ۲۰۰ و ۳۰۰ اجرا می‌شود. هدف این بود که ببینم با بیشتر کردن این بعد چقدر می‌توان دقت را بهتر کرد. برای SVM این سرچ را به ازای کرنل‌های مختلف از جمله خطی، چندجمله‌ای و سیگموید هم انجام دادم. در تمام موارد عدد ۳۰۰ نتیجه‌ی بهتری داد ولی تفاوت آن با بعد ۱۰۰ در تمام مدل‌ها برابر ۲ تا ۳ درصد روی تمام معیارها بود و ارزش اضافه کردن بعد به اندازه ی ۳ برابر برای این مقدار دقت منطقی نیست. بعد بیشتر هم زمان بر تر هست و هم حجیم تر که از آین صرفه نظر می‌کنم. برای همین مرحله ی ارزیابی را با همین مدل هایی که ساخته شد انجام می‌دهم.

مدل دسته بندی	Naive_Bayes	Perceptron	SVM
دقت با n_components = 100	0.77	0.76	*0.82*
دقت با n_components = 300	0.80	0.78	*0.84*

۶. تمام عملیات ۱ تا ۵ را که به صورت یک pipeline پیاده شده برای متن های x_text هم انجام می‌دهم و برای این مجموعه ی به دست آمده کار دسته‌بندی را با مدلی که در ۵ ساخته شد انجام می‌دهم. برچسب‌های پیش بینی شده روی y_predicted ذخیره می‌شوند و سپس با تابع precision_recall_fscore_support از sklearn.metrics مقادیر precision و recall و f-score را در حالت macro برای کل دسته‌بندی ۱۱ کلاسه محاسبه می‌کنم که نتایج آن به صورت زیر هستند.

مدل دسته بندی	Naive_Bayes	Perceptron	SVM
precision	0.77	0.76	*0.82*
recall	0.78	0.78	*0.82*
f-score	0.77	0.76	*0.82*

بهترین مدلی که برای این مساله‌ی دسته‌بندی ۱۱ کلاسه ساخته شده در اینجا clf_svm یک دسته بند SVM هست که از کرنل rbf استفاده می‌کند و با بردارهای از اندازه ی ۱۰۰ به عنوان بردار نظیر متن LSA کار می‌کند. این مدل به لحاظ هر ۳ معیار گزارش شده وضعیت بهتری نسبت به دو مدل دیگر دارد.