تمرین دوم – گزارش کار

منطق کد:

اول از همه ستون label را از روی ستون suggestion و score تشکیل می‌دهیم و سعی بر این داریم که کامنت‌های نویزی را هم در همین حین حذف کنیم به همین منظور با توجه به توضیحات صفحه دیتاست در کگل زمانی که suggestion مقدار 1 دارد و مقدار score هم از 70 بیشتر است را به عنوان لیبل positive یا 2 برمی‌گردانیم. زمانی که suggestion مقدار 3 دارد و score هم بالای 40 است را به عنوان لیبل nuetral یا 1 برمی‌گردانیم و score کمتر از 40 و مقدار 2 را معادل با لیبل Negative یا 0 برمی‌گردانیم باقی دیتاها که در این شرط قرار نمی‌گیرند را -1 قرار می‌دهیم و به عنوان داده نویزی حذف می‌کنیم. در مرحله بعد پس از تست و متوجه شدن این موضوع که اکثر دیتاهای ما در کلاس 2 قرار می‌گیرند و به همین دلیل مدل ما به سمت 2 میل بیشتری پیدا می‌کند و bias می‌شود upsampling انجام می‌دهیم و تعدادها را برابر می‌کنیم. برای preprocess داده‌ها از کتابخانه‌ها hazm و parsivar استفاده می‌کنیم. یک نرمالایزر، توکنایزر و استمر تعریف می‌کنیم و ابتدا متن را نرمالایز می‌کنیم و در مرحله بعد با استفاده از regex علائم و اعداد و فاصله‌های زیاد را حذف کرده. بعد از آن از توکنایزر استفاده می‌کنیم و در مرحله بعد چک می‌کنیم هر توکن در لیست stopwords هست یا نه و اگر نبود ریشه آن به خروجی اضافه می‌شود و اگر بود هم از آن رد می‌شویم. در نهایت خروجی متن توکنایز شده و در صورت درخواست ریشه‌یابی شده است. در مرحله بعد سعی می‌کنیم از روی متن توکنایز شده vocab را بسازیم. برای این منظور ابتدا دو توکن خاص برای padding و واژگان ناشناخته تعریف می‌کنیم و در هر مرحله برای هر یک از کلمات در صورتی که تعداد آن‌ها از حد ما بالاتر باشد (که دیفالت آن یک است) آن‌ها را با تعدادشان به دیکشنری اضافه می‌کنیم و در نهایت دیکشنری را خروجی می‌دهیم. دو تابع هم برای تبدیل توکن‌ها به اندیس‌های عددی و پد کردن دنباله‌ها هم داریم. در نهایت کلاس اختصاصی دیتاست خودمان را می‌سازیم و با استفاده از dataloader دو نسخه آموزشی و اعتبارسنجی با نسبت 20:80 ایجاد می‌کنیم. در مرحله بعد مدل خودمان را تعریف می‌کنیم. به ترتیب لایه embedding، LSTM(دو طرفه)، dropout، fully connected، batch normalization، ReLU و مجددا fully connected. لایه اول برای تبدیل اندیس کلمات به dense vectors است. (ناگفته نماند که کلمات مشابه در فضا نزدیک به هم قرار دارند بعد از embedding). لایه بعدی LSTM دو طرفه است که مدل از دو جهت متن را می‌خواند تا درک بهتری از آن پیدا کند. لایه بعدی یا dropout برای جلوگیری از بیش‌برازش

(overfitting) است. لایه بعدی وظیفه این را دارد که بردار خروجی از لایه dropout را به فضای میانی نگاشت کند (در LSTM دو طرفه بردار خروجی دو برابر یه طرف است). لایه batch_norm وظیفه این را دارد که خروجی لایه قبل خودش را نرمال‌سازی کند و توزیع داده در طول آموزش را ثابت‌تر نگه می‌دارد. لایه ReLU با خاصیت غیرخطی برای یادگیری ویژگی‌های پیچیده‌تر کاربردی است. در نهایت لایه آخر برای تولید خروجی نهایی مدل است. از تابع ضرر cross-entropy و بهینه‌ساز Adam استفاده کردم. تابع evaluate را داریم که نتیجه مدل را بر روی یک dataloader اجرا می‌کند و امتیازهای model بر روی این dataloader را خروجی می‌دهد. در مرحله بعد با استفاده از روتین pyTorch مدل را آموزش می‌دهیم. و در نهایت هم بر روی 5 داده اول مجموعه داده اعتبارسنجی نتایج را مشاهده می‌کنیم.

روند پیشرفت:

در تست اول نتایج به سمت کلاس 2 که بیشترین کلاس بعد bias شده بود و به همین دلیل از upsampling استفاده کردم. بعد از این نتایج یکم متعادل‌تر شد اما همچنان در تست دوم خروجی‌ها اشتباه داشت به همین دلیل مدل را با اضافه کردن لایه‌های dropout، batch_norm و ReLU و دوطرفه کردن لایه LSTM بهبود بخشیدم و در نهایت به accuracy نزدیک 84 درصد و f1-score نزدیک 48 درصد رسیدم. در خروجی 5 داده اول مجموعه اعتبار سنجی هم 4 تا درست پیش‌بینی شده‌اند که نشان از این دارد که accuracy نزدیک به 84 درصد غیرواقعی هم نبوده.