به نام خدا

پروژه درس یادگیری ماشین

دانشگاه صنعتی شریف

دکتر مطهری

پردیس زهرایی ۹۹۱۰۹۷۷۷

برای این پروژه باید دیتاستی که امتیازات نظرهای مختلفی را شامل می شود، بررسی کنیم و برای داده های دیگر آنها را پیش بینی کنیم.

لینک پروژه:

https://colab.research.google.com/drive/1YhBI3H6eZOvBM7zeYmsiSfdKhwri_v1l?usp=sharing

نکته اولی که وجود دارد این است که این دیتاست دقیق نیست. مثلا یک "نقطه یا ویرگول" می تواند باعث تغییر معنی جمله بشود که در واقعیت نباید باعث همچین اتفاقی بشود و دیتاست هم کمی بایاس دارد از آنجا که تعداد امتیازهای ۲ بیشتر از بقیه است. در داده های ورودی ۴ ستون داریم که ستون sentence_ID و phrase دارای همبستگی شدیدی هستند پس sentence_id را حذف می کنیم و در کل با phrase و sentiment فقط کار میکنیم.

بنظرم مشکل از دیتاست اولیه است که امتیازات اشتباهی به نظرها داده شده است. پس برای همین انجام EDA در آن باعث کمتر شدن دقت می شود هرچند که تمام اینکار ها انجام شده است. علاوه بر اینها دو رویکرد داده شده که رویکرد اولی با اینکه در بسیاری از مواقع به خوبی کار می کند اما در اینجا به دلیل اینکه داده های ورودی تکه های یک جمله هستند و ترتیب و نقطه گذاری و ... در امتیازات اثر مستقیم دارد، استفاده از این رویکرد با خطا همراه است پس بهتر است سراغ رویکرد دوم برویم. استفاده از tf-idf که در بازیابی اطلاعات هم کاربرد دارد در اینجا نمی تواند به خوبی به ما اطلاعات بدهد و نتیجه خوبی از آنها نگرفته ام که دلیل آنها را ماهیت دیتاست می دانم.

برای tokenize وvectorize کردن هم من از روش های مختلفی استفاده کرده ام که Keras tokenizer countvectorizer و یا NLTK split() امتحان کردم ولی countvectorizer بنظرم از بقیه روش ها بهتر است. از روش های embedding هم با fastText پیش رفتم که بنظرم بسیار خوب نتیجه می داد و در شبکه های عصبی به خصوص لایه های کتابخانه keras می توان از آن به خوبی استفاده کرد. پس رویکرد ۱ نسبتا نامطلوب است و استفاده از رویکرد ۲ به دقت کمک زیادی می کند. هرچند کاملا وابسته به نوع مدل دارد برای مدل های ساده تر برای مثال رگرسیون یا درخت

تصمیم رویکرد اول نتیجه بهتری می دهد اما برای شبکه عصبی پیشرفته تر رویکرد دوم و وابسته به نوع مدل می توان تصمیم گرفت.

نکته دوم این است که برای این پروژه من روش ها و مدل ها و هایپرپارامترهای متفاوتی را امتحان کردم و من از CNN ساده و Naïve bayes و deciosn tree classifier و مدل های پیشرفته تر شبکه عصبی استفاده کردم همانطور که در نکته ۱ آورده شده برای شبکه عصبی استفاده از fast_text یا glove بسیار بهتر از رویکرد اول است و برای اینکه از embedding استفاده کنم شبکه های عصبی به کمک کتابخانه keras این قابلیت را به من می داد تا بتوانم از آنها به طور مستقیم استفاده کنم و از ترکیب شبکه عصبی CNN برای افزایش دقت با LSTM ترکیب کرده ام که در زیر هم توضیح آن را داده ام، می توانستم به خوبی از ویژگی های آن استفاده کنم و دقت خوبی هم می داد و کتابخانه keras هم ابزارهای پیاده سازی را فراهم می کند.

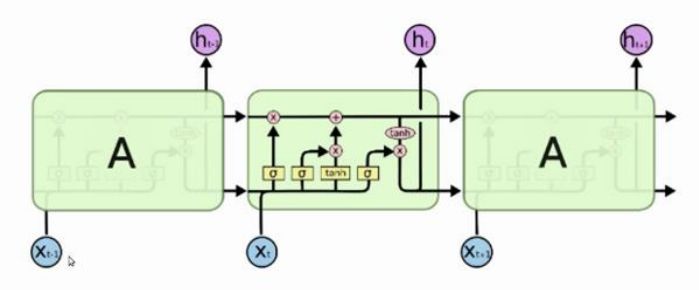
که از توضیحات این مدل در این لینک استفاده کرده ام:

https://medium.com/@mixanyy/different-ways-to-combine-cnn-and-lstm-networks-for-time-series-classification-tasks-b03fc37e91b6

توضیحات مدل :LSTM

LSTM نوعی RNN است که قادر به یادگیری وابستگی های طولانی مدت در داده های متوالی است. برخلاف شبکه‌های عصبی feedforward ، LSTM دارای گیت های بازخوردی است که به آن اجازه می‌دهد تا نه تنها نقاط داده، بلکه کل توالی داده‌ها را پردازش کند.

که ساختاری مانند زیر دارد:

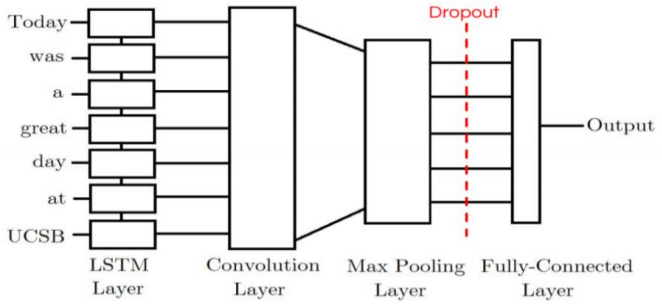


با کتابخانه keras قابل پیاده سازی است.

توضیحات مدل LSTM-CNN

من از این شیوه استفاده کرده ام که بنظرم به خاطر اینکه پترن در نظرات وجود دارد CNN مفید و به خاطر اینکه نظرات هر کدام دنباله ای از یک نظر هستند LSTM مفید است به این صورت عمل کرده که:

خروجی شبکه LSTM که قادر به گرفتن وابستگی های داده است، به عنوان ورودی به CNN داده می شود. اینکار به CNN اجازه می دهد تا بر یادگیری ویژگی ها از خروجی شبکه LSTM تمرکز کند، که سپس می تواند برای طبقه بندی استفاده شود.



حال به توضیحات بخش بخش پروژه را شرح می کنم:

۱) ابتدا به پکیج های استفاده شده و کاربرد های آن ها می پردازیم ، همانطور در بالای کد آورده شده بهتر است ابتدا با GPU سل ها را ران کنیم. و کاربرد کتابخانه ها به طور مختصر به صورت زیر بوده است:

- **"numpy" یک کتابخانه برای کار با آرایه های عددی است.**
- **"pandas" یک کتابخانه برای تجزیه و تحلیل داده ها است.**
- **"tensorflow" یک کتابخانه برای شبکه های عصبی است.**
- **"keras.preprocessing" ابزارهایی را برای کار با داده های متوالی فراهم می کند.**

- "keras.layers" بلوک های ساخت شبکه های عصبی فراهم می کند.
- "keras.models" کلاس هایی را برای آموزش مدل های یادگیری ماشین ارائه می دهد.
- "keras.optimizers" الگوریتم های بهینه سازی را برای آموزش شبکه های عصبی ارائه می دهد.
- "sklearn.naive_bayes" مدل برای Naive Bayes را ارائه می دهد.
- "sklearn.feature_extraction.text" ابزارهایی را برای کار با داده های متنی فراهم می کند.
- "google.colab" ابزارهایی را برای استفاده از نوت بوک های Google Colab ارائه می دهد.
- "sklearn.model_selection" ابزارهایی را برای انتخاب و ارزیابی مدل فراهم می کند.
- "nltk" یک کتابخانه برای پردازش زبان طبیعی است.
- "imblearn.over_sampling" روش هایی را برای نمونه برداری بیش از حد (oversampling) از مجموعه داده های نامتعادل ارائه می دهد.
- "torch" کتابخانه ای برای محاسبات تنسور و شبکه های عصبی است.
- "torch.nn" بلوک های ساختمانی برای ساخت شبکه های عصبی در PyTorch فراهم می کند.

- **"seaborn" کتابخانه ای برای تصویرسازی داده های آماری است.**

- **"gensim.models" مدل هایی را برای یادگیری بدون نظارت ارائه می دهد.**

۲) توضیح رویکرد دوم در پروژه: من از fastText استفاده کرده ام که بیشتر از embedding استفاده شده که بعدا در شبکه به عنوان ورودی داده شده نحوه کار آن به این صورت است: فایلFastText را خط به خط می خواند و کلمه و ضرایب برداری مربوط به آن را استخراج می کند. این بردارها در یک دیکشنری با کلمه به عنوان کلید و بردار به عنوان مقدار ذخیره می شوند.
ماتریس embedding ایجاد می کند و کد روی واژگان تکرار می شود و بررسی می کند که آیا کلمه در دیکشنری FastText وجود دارد یا خیر. اگر چنین باشد، بردار مربوطه به ماتریس embedding در ایندکس آن کلمه به واژگان اضافه می شود.
۳) لیستی از مدل های به کارگرفته شده و هدف و تعداد پارامتر آنها و ارزیابی مدل ها

مدل های به کارگرفته شده ۴ نوع عبارتند از:

۱) شبکه عصبی ساده: هدف از آن استفاده از یک مدل دیپ لرنینگ اما با تعداد لایه های کم (۴) لایه برای بررسی نحوه عملکرد داده بود چون شبکه های عصبی CNN,RNN قبل از ظهور ترنسفورمرها بهترین مدل های بروز بودند از آن استفاده کرده ام و برای اکتیویشن از softmax,relu و از بهینه ساز adam برای آنها استفاده کرده ام که چون ساده بود ۳۵ ایپاک هم طی شده که در نتیجه ارزیابی مدل :

```
loss: 1.3344 - accuracy: 0.4734 - val_loss: 1.3332 -
val_accuracy: 0.4504
```

این مدل بسیار ساده است برای همین با اینکه سریع است اما دقت کم است

۲) مدل های GaussianNB & MultinomialNB:

هدف از آنها هم دوباره بررسی نحوه عملکرد روش ۱ و ۲ با عوض شدن مدل بود که رویکرد اول نتیجه بهتری داشت و متوجه اثر مدل بر دقت شدیم و پارامتر های زیادی برای این مدل وجود ندارد و با رویکرد اول نتیجه بهتری را می دهد.

```
Precision: 0.49
Recall: 0.32
F1-score: 0.35
```

این مدل برای این سوال به خوبی عمل نمی کند ولی در حالت کلی خوب است

۳) مدل DecisionTreeClassifier:

هدف استفاده از این مدل ، این بود که بتوان هایپرپاراماتر های آن را عوض کرد و به بهتر شدن نتیجه پی برد و مدل نسبتا قوی ای هم هست و بهترین پارامترهای آن به شکل زیر هستند و عملکرد زیر را دارد:

```
Best hyperparameters: {'criterion': 'entropy',
'max_depth': 8, 'min_samples_leaf': 2,
'min_samples_split': 5}
Precision: 0.44
Recall: 0.50
F1-score: 0.38
```

این مدل نقطه ضعف و قوت خاصی ندارد و مدل خوب و سریعی است

۴) LSTM + CNN

برای توکن کردن داده ها از keras tokenizer **استفاده شده.**

روش دیگری که استفاده شده countvectorizer **است. سپس، با استفاده از تابع** pad_sequences **، داده های متنی توکنیزه شده اضافه می شوند. بردارهای** FastText **می خواند و یک ماتریس** embedding **برای واژگان توکنایزر ایجاد می کند.**

خود مدل شامل یک لایه ورودی، یک لایه embedding **، یک لایه** dropout**، یک لایه**LSTM**، یک لایه** dropout **، دو لایه کانولوشنال با نرمال‌سازی، یک لایه** maxpooling **، یک لایه** dropout **، یک لایه متراکم با فعال‌سازی**ReLU**، یک لایه** dropout **دیگر و در نهایت یک لایه خروجی متراکم با فعال سازی**softmax

مدل با بهینه سازهای مختلف فیت شده و برای loss **هم از آنتروپی استفاده شده است. سپس بر روی داده های آموزشی ۱۵ بار با آموزش داده می شود. سپس از مدل آموزش دیده برای پیش بینی داده**

های آزمون استفاده می شود. پیش بینی ها با استفاده از تابع argmax به برچسب های کلاس تبدیل می شوند و در یک فایل CSV ذخیره می شوند.

برای توضیح بیشتر مدل و لایه های آن می توان گفت:

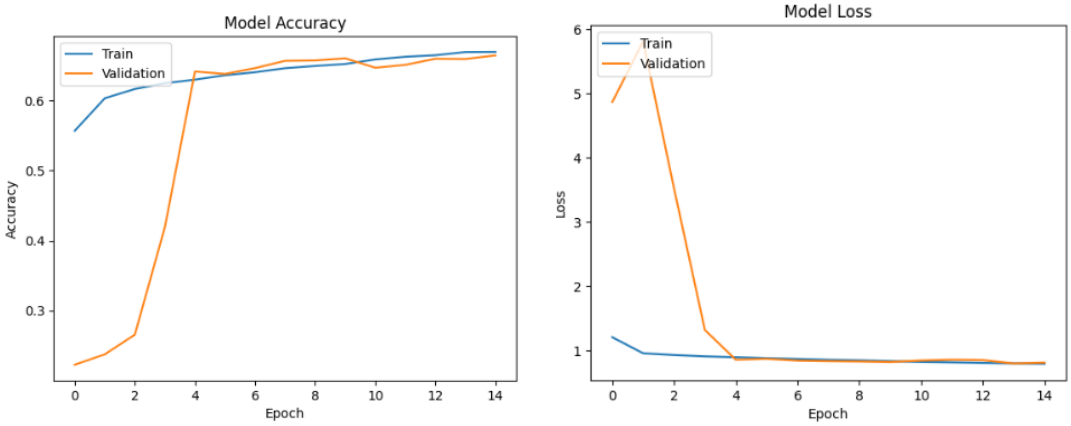
لایه dropout به طور تصادفی تعدادی از داده های ورودی را حذف می کند که باعث جلوگیری از overfitting می شود.

لایه LSTM داده را از جلو و عقب پردازش کرده و اطلاعات در مورد داده ها و پترن ها را بهتر بررسی می کند.

لایه کانولوشن مجموعه ای از فیلترها را روی داده اعمال می کند تا ویژگی های بیشتری را پیدا کند و نرمال سازی هم در این لایه رخ داده تا آموزش سریعتر شود.

در لایه متراکم هم در نهایت یک توزیع احتمال را در نهایت ارایه که ماکسیسمم آنها را برمیگزنیم.

عملکرد بسیار خوبی داشته :



نقطه ضعف این مدل کند بودن آن و امکان اورفیت کردن است که باید مراقب باشیم در دادن لایه ها و ایپاک زیاده روی نکنیم ولی دقت بالا دارد.

نتیجه گیری:

همانطور که در ابتدای گزارش آورده شد، به دلیل اینکه دیتاست داده شده به یک سری موارد که رویکرد اول به آن حساس نیست ، حساس است در مدل های شبکه عصبی پیشرفته به خوبی کار

نکرده ولی در مدل های ساده تر مثل رگرسیون یا درخت تصمیم بهتر است و انتخاب رویکرد وابسته به نحوه پیاده سازی و مدل نهایی است.

در تلاش های شکست خورده می توان دقت کم مدل ها را گفت و اینکه استفاده از countvectorizer **کمک زیادی نکرد و استفاده از** keras **نتیجه بهتری داشت و** fasttext **چون می تواند شامل داده منفی هم شود، برای نیو بیز بهتر است با رویکرد دوم از** gaussian **استفاده کرده ولی در رویکرد اول از** MultinomialNB **هم می توان استفاده کرد. شبکه عصبی ساده اولیه هم که استفاده شد به دلیل اینکه تعداد لایه ها کم بود و از** LSTM **استفاده نشده بود مدل خوبی نبود چون نتوانسته بود به بهترین صورت از پترن ها استفاده کند.**

همچنین نامتعادل بودن دسته ها هم بررسی شد و در مدل های ساده تر باعث افزایش دقت می شد (روش over sampling**) ولی در مدل های پیچیده تر خیلی باعث بهبود نمی شد. همچنین ارزیابی شبکه عصبی بیشتر با** loss,accuracy,confusion matrix **بوده و مدل های دیگر اما چون مدل های** sklearn **بودند با روش های** F1 score **و** precision Recall **هم به دست آمده اند**