دانشگاه صنعتی امیرکبیر

دانشکده‌ی علوم‌کامپیوتر

گردآورنده:

شیده هاشمیان

شماره دانشجویی:

۹۶۱۳۴۲۹

پروژه‌ی نهایی درس پردازش زبان طبیعی

عنوان : بررسی عملکرد روش های مختلف برای تحلیل احساسات

استاد درس: دکتر اکبری

پاییز ۹۹

صورت مسئله

به‌طور کلی مسئله‌ای که در این پروژه مورد بررسی قرار گرفته است، تحلیل احساسات متن‌های تولید شده در میکرولاگ‌ها است. امروزه اکثرا افراد در شبکه های اجتماعی فعالیت داشته و همواره درحال تولید محتوای جدید در این شبکه های هستند. یکی از محتوا های کاربردی که این افراد منتشر می کنند نظراتشان راجع به مسائل متفاوت است که با بررسی این متن ها و دسته بندی آن ها از نظر محتوای پیام از نظر احساسی می توان به اطلاعات مهمی دست یافت. برای مثال شرکتی می تواند باببررسی و تحلیل مجموعه متن هایی که کاربران درمورد آن شرکت یا به طور خاص، یکی از محصولات ارائه شده توسط آن شرکت نوشته‌اند می تواند میزان رضایت کاربران و یا به طور جزئی تر یافتن و بررسی نظرات منفی برای تصمیم‌گیری و برنامه‌ریزی‌های آینده ی خود استفاده کنند.(Gautam, 2014) (Go, 2009) باتوجه به این موضوع، در این پروژه قصد بر این است که به تحلیل احساسات داده‌های متنی منتشر شده در میکروبلاگ توییتر پرداخته شود و متن توییت‌ها را به دو دسته‌ی مثبت و منفی دسته بندی کرد.

چکیده

به منظور بررسی احساسات و نیت متن‌های تولید شده در میکروبلاگ توییتر، از دو رویکرد متفاوت استفاه خواهد شد، اولین روی‌کرد بردارسازی داده‌ها با مدل‌های پیچیده‌تر با عملکردی بهتر که همان `transformers` هستند و سپس دسته‌بندی توسط `SVM` که یکی از بهترین دسته‌بندهای خطی است. در دومین روی‌کرد تماما از یک `transformer(BERT)` برای دسته‌بندی استفاده می‌شود و نهایتا عملکرد این دو دسته‌بند مقایسه می‌شود.

مجموعه داده

مجموعه داده‌ی معرفی شده در سایت `Kaggle` تحت عنوان `sentiment140` استفاده شده است. دلیل ابتدایی انتخابی این مجموعه داده بالا بودن نسبی داده‌های این مجموعه داده که برابر ۱۶۰۰۰۰۰ داده‌ی متنی توییتر است که با استفاده از `twitter API` به‌دست آمده است.(Go, 2009) همچنین لازم بذکر است که داده‌های متنی این مجموعه داده توسط روش‌های یادگیری ماشین برچسپ‌گذاری شده است. به‌طور جزئی‌تر این مجموعه داده در فرمت یک فایل `csv` با اینکدینگ `"ISO-8859-1"` معرفی شده‌است و ستون‌های این مجموعه داده به شرح زیر است:

- هدف[۱]: در این ستون دو مقدار ۰ و ۴ ظاهر شده است که صفر به معنای داشتن احساس و نیت[۲] منفی است و چهار به معنای نیت و احساس مثبت است.
- شناسه[۳]: یک عدد خاص که شناسه‌ی آن توییت است در آن قرار دارد.

[۱] target

[۲] sentiment

[۳] id

- تاریخ[۴]: تاریخ گذاشته شدن توییت در این ستون قرار دارد.
- پرچم[۵]: درصورت داشتن کوئری، متن کوئری و در غیر این صورت برابر با `NO_QUERY` است. (توضیحات بیشتری در مورد این ستون آورده نشده است و باتوجه به عدم استفاده از این ستون، دلیل وجود این ستون بررسی نشده است.)
- کاربر[۶]: کاربر توییتری که توییت را منتشر کرده است.
- متن[۷]: متن توییت انتشار شده در این ستون آمده است.

توابع و کلاس‌ها

همچنین برای آسانی پیدا کردن تابع مربوط به هر قسمت در کد، متناسب با قسمت‌بندی‌های درون کد توضیحات داده می‌شوند. (متناسب با تیترهای آمده در فایل `notebook` کد مربوطه، هر قسمت توضیح داده شده است.)

- نصب (`installation`): باتوجه به بهینه بودن سرعت اجرایی با استفاده از `GPU`، این کد برای اجرا در `GoogleColab` نوشته شده است و به ان منظور کتاب‌خوانه‌های استفاده شده در این پروژه در این قطعه کد نصب می‌شوند.
- `imports`: در این سلول از کد، تمام کتاب‌خوانه‌های استفاده شده در این پروژه را `import` می‌کنیم.
- ثابت‌ها (`constants`): در این سلول از کد، تمامی نام‌ها و مسیرهای ثابت مانند مسیری که داده‌های آموزش و آزمایش در آن قرار دارند و یا مسیری که د رآن مدل‌ها دسته‌بند ذخیره می‌شوند و دیگر چیزهایی شبیه به این قرار دارند. (از آن‌جایی که نام‌های مرتبط برای متغییرهای این سلول انتخاب شده است، از تمامی آن‌ها نام برده نشده است.)
- پیش‌پردازنده‌ی متن (`text preprocessing`): در این سلول یک تابع وجود دارد که به‌شرح زیر است.
  - تابع `text_preprocessor(text)`: این تابع یک متن به‌صورت رشته را ورودی می‌گیرد، سپس ابتدا کوچک‌سازی‌های موجود رد متن را حذف کرده، مانند تغییر ایجاد شده در مثال زیر.

`Can't → can not` `haven't → have not`

سپس نام‌های کاربرهایی که در هر توییت آورده شده است را حذف می‌کند. (`mention` شدن کاربران دیگر در متن پیام‌ها) پس از آن حذف `URL`های آورده شده در متن و سپس حذف علائم نگارشی و ایست‌واژه‌ها بغیر از کلمه‌ی `not` (به دلیل معنادار بودن آن در مسئله‌ی مورد نظر). سپس رشته‌ی تولیدی پس از این تغییرات را در غالب یک رشته خروجی می‌دهد.

نمونه‌ای از عملکرد این تابع نیز در این سلول به‌عنوان مثال آورده شده است که نتیجه‌ی آن برای دو توییت انتخابی به شکل زیر است:

---

[۴] date
[۵] flag
[۶] user
[۷] text

raw tweet	the next school year is the year for exams can't think about that- #school #exams #hate
Processed tweet	next school year year exams not think - school exams hate
raw tweet	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
Processed tweet	awww bummer shoulda got david carr third day ;D

- تقسیم داده به دو دسته‌ی آموزش و آزمایش به نسبت ۸۰، ۲۰:

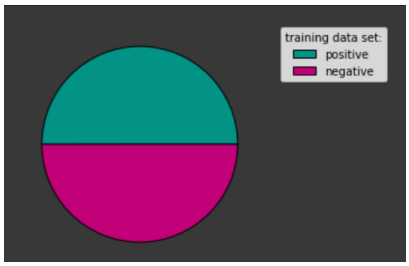
در این سلول یک تابع با نام زیر وجود دارد که در زیر شرح داده شده است.

- تابع `split_data(csv_dat_path)`:

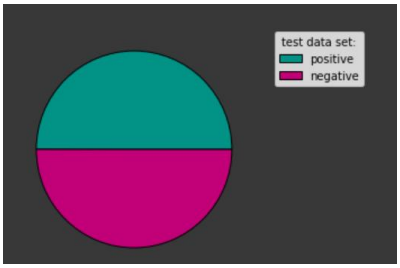
این تابع مسیر (شامل نام فایل) داده‌ی اولیه که فرمت `csv` دارد را ورودی گرفته، ابتدا تنها ستون‌های مورد استفاده که ستون هدف و متن است را با پیش‌پردازش متن با استفاده از تابع `text_preprocessor` (به دلیل بالا بودن تعداد توییت‌ها، تنها متون پردازش شده را ذخیره می‌کنیم، زیرا در ادامه این رشته‌های پیش‌پردازش شده هستند که برای آموزش و آزمایش استفاده می‌شوند.) در لغت‌نامه‌ای ذخیره می‌کند، سپس با استفاده از تابع `train_test_split` موجود در کاتب‌خانه‌ی `sklearn.model_selection`، داده را به نسبت ۸۰ به ۲۰ (۸۰ درصد داده‌ی آموزش و ۲۰ درصد داده‌ی آزمایش) به صورت تصادفی تقسیم می‌کند، سپس هر یک را به‌صورت لغت‌نامه‌ای به صورت زیر، در یک فایل در مسیر داده شده به‌عنوان `data_path` در سلول مربوط به ثابت‌ها ذخیره می‌کند.

```
{'label': <list of 0 and 4 which represent sentiment labels>,
 'text': <list of strings which each represent a tweet in dataset>
}
```

سپس در انتها در سلولی دیگر، توضیع داده‌های مثبت و منفی را در دو دسته‌ی آموزش و آزمایش به‌صورت نمودار دایره‌ای نمایش می‌دهیم. (شکل زیر برای تقسیم داده‌ی استفاده شده در ادامه‌ی پروژه است.)



شکل۱ توضیع داده‌ی آموزش (داده‌ی منفی: ۶۳۹۹۴۹ , داده‌ی مثبت: ۶۴۰۰۵۱)



شکل۲ توضیع داده‌ی آزمایش (داده‌ی منفی: ۱۶۰۰۵۱ , داده‌ی مثبت: ۱۵۹۹۴۹)

- دسته‌بند خطی/ تعریف دسته‌بند
- کلاس `LinearClassifier`:

این کلاس به‌منظور ساخت یک دسته‌بند خطی که متناسبا برای آن توابعی به‌منظور آموزش مدل، آزمایش مدل و گرفتن دسته‌ی یک توییت از مدل پیاده‌سازی شده است، ساخته شده است که در زیر شرح توابع این کلاس آورده شده است. همچنین در این پروژه از دسته‌بند خطی **SVM(Support Vector Machine)** استفاده شده است.

  - تابع `__init__(self,train_file_path,test_file_path ,model_path= linear_model_path)`:

این تابع دو مسیر کامل فایل‌های **pickle** آموزش و آزمایش را که پس از اجرای تابع **split_data** ذخیره شده‌اند را ورودی می‌گیرد و این‌دو را در دو متغییر از کلاس با نام‌های مرتبط ذخیره می‌کند و مسیری که در آن مدل خطی پس از آموزش ذخیره می‌شود را نیز می‌تواند ورودی بگیرد و در صورت نبود ورودی‌یا متنار با این متغیر، مقدار آن را برابر با مسیر معرفی شده برای این مدل در سلول ثابت‌ها می‌ذارد و آن را در متغیر **save_model_path** این کلاس ذخیره می‌کند.

همچنین مقدار **MAX_LEN** را که برای حذف داده‌هایی با طول زیاد در زمان آموزش است را نیز با مقدار متغییری با همین نام در سلول کد ثابت‌ها برابر قرار می‌دهد. (این مقدار با بررسی میانگین تعداد توکن‌های توییت‌ها که برابر ۳۹۰ بود و با آزمایش عددهای مختلف برای بررسی این که چقدر از داده را پوشش می‌دهند، به نتیجه‌ی ۶۴ توکن رسید که از میان ۱۶۰۰۰۰۰۰ داده، ۱۵۹۹۵۱ داده را شامل می‌شود.)

همین‌طور در زمان بردار سازی داده‌های آموزش و آزمایش، باتوجه به بالا بودن تعداد داده‌ها، امکان همزمانی بردار سازی و آموزش و آزمایش میسر نبود (به دلیل محدودیت حافظه) به همین منظور این دو داده را به تعدادی زیردسته (**chunk**) تقسیم کرده و آن‌ها را بردار و به‌صورت فایل ذخیره کرده، تعداد دسته‌های داده‌ی آموزش و آزمایش نیز در این‌جا با متغییرهای **train_chunk_num** و **test_chunk_num** تعیین شده است. و نهایتا مسیری که در آن هریک از دسته داده‌های بردار شده‌ی داده‌های آموزش و آزمای در آن ذخیره می‌شوند را در متغییری با نام مرتبط در کلاس ذخیره می‌کند. (مسیرها همرا با نام فایل نیز هستند.)

  - تابع `vectorizer(self,data_dict= None,query= None,train= True)`:

ابتدا در دسترس بودن **GPU** را بررسی می‌کند و متناسب با پردازنده‌ی دردسترس (در صورت دردسترس نبودن **GPU**، از **CPU** استفاده می‌شود) و یک نمونه از کلاس مدل انتخابی برای بردارسازی را می‌سازد. در این‌جا از کتاب‌خانه‌ی معرفی شده توسط UKPLab با نام **sentence_transformer** استفاده شده است و باتوجه به نتایج ارائه شده برای مدل‌های مختلف موجود در این کتاب‌خانه، مدل **DistilBERT** باتوجه به سرعت و عملکردی که دارد برای بردارسازی هر توییت به‌عنوان یک جمله استفاده شده است. حال باتوجه به نوع ورودی، حالت‌های زیر که ممکن است توضیح داده می‌شود.

- حالت `data_dict= dict, query= None, train= True`:

این حالت تابع برای بردارسازی داده‌های آموزش صدا زده شده است، پس با استفاده از `train_chunk_num` اندازه‌ی هر دسته را پیدا کرده، سپس با استفاده از تابع `encode` مدل انتخابی برای بردارسازی، هر توییت را بردار کرده در قالب زیر ذخیره کرده.

```
{‘vec’: <list of each tweet vector>,
 ‘label’: <list of each tweet label>
}
```

پس از رسیدن تعداد داده‌های بردارسازی شده به تعداد یک دسته (`chunk`) آن‌ها را در مسیر مناسب ذخیره می‌کند.

- حالت `data_dict= dict, query= None, train= False`:

در این حالت این تابع برای بردارسازی داده‌های آزمایش صدا زده شده استو در آن مانند حالت عمل می‌کند با این تفاوت که تعداد داده‌ی موجود در هر دسته را با استفاده از `test_chunk_num` بدست می‌آورد و داده‌های بردار شده را در مسیر مناسب برای داده‌های آزمایش ذخیره می‌کند.

- حالت `data_dict= None, query= string, train= False`:

در این حالت تابع برای تک جمله (توییت) صدا زده شده است و در آن با استفاده از تابع `encode` مدل انتخابی برای بردارسازی، بردار متناظر با آن جمله را به‌عنوان خروجی بازمی‌گرداند.

- تابع `define_model(self)`:

این تابع یک نمونه از کلاس `SGDClassifier` موجود در `sklearn.linear_model` ساخته که یک دسته‌بند خطی مبتنی بر `SVM` است و آن را به‌عنوان خروجی بازمی‌گرداند.

- تابع `train(self)`:

ابتدا داده‌های آموزش را در ۵ حالت متفاوت به نسبت ۸۰ -۲۰ برای آموزش و ارزیابی مدل تقسیم کرده (`cross validation`)، هر بار باتوجه به تقسیم بندی، ابتدا یک مدل دسته‌بند خطی با استفاده از تابع `define_model` ساخته، سپس دسته‌هایی که برای قسمت آموزش انتخاب شده اند را به نوبت وارد حافظه کرده و با استفاده از تابع `partial_fit` مدل تعریف شده در یک حلقه‌ی ۱۰۰۰ تایی (پیشنهاد شده در داکیومنت موجود برای این کلاس) مدل را برای آن دسته از داده آموزش داده و نهایتا پس از آموزش مدل برای دسته‌های انتخاب شده برای آموزش، مدل را بر اساس دسته‌های باقی‌مانده برای ارزیابی، با استفاده از تابع `predict` مدل را ارزیابی کرده و با محاسبه‌ی میانگین دقت برای هر دسته، درصورتی که این میانگین دقت از میانگین دقت مدل پیشین بهتر باشد، آن را در مسیر تعیین شده برای این مدل (`save_model_path`) ذخیره می‌کند. (درصورتی که فایل‌های مربوط به بردارهای داده‌ی آموزش موجود نباشد، ابتدا باتوجه به مسیر داده‌شده برای داده‌های آموزش (`train_path`) و استفاده از لغت‌نامه‌ی موجود در آن، تابع `vectorizer` را برای بردارسازی داده‌های آموزش صدا زده و سپس تابع `train` را برای آموزش دوباره صدا می‌زند.)

- تابع evaluate(self):

این تابع با استفاده از فایل‌های موجود برای بردارهای آزمایش، فایل موجود برای مدل آموزش دیده شده و تابع **predict** مدل، برای هر دسته (**chunk**) از بردارها، دسته‌ی تخمین زده شده برای هر توییت توسط مدل و دسته‌ی واقعی آن داده را در دو لیست ذخیره می‌کند. نهایتا از این دو لیست و توابع **classification_report** و **confusion_matrix** از **sklearn.metrics** برای ارزیابی مدل استفاده می‌کند و نتایج ارزیابی را خروجی می‌دهد. (در صورتی که فایل متناظر با مدل موجود نباشد، ابتدا تابع **train** را برای آموزش مدل صدا زده و سپس تابع **evaluate** را دوباره صدا می‌زند و مانند تابع **train** درصورتی که فایل‌های مربوط به بردارهای داده‌ی آزمایش موجود نباشد، ابتدا باتوجه به مسیر داده شده برای داده‌های آزمایش (**test_path**) و استفاده از لغت‌نامه‌ی موجود در آن، تابع **vectorizer** را برای بردارسازی داده‌های آزمایش صدا زده و سپس تابع **evaluate** را برای آزمایش دوباره صدا می‌زند.)

- تابع query(self, query_text):

در این تابع با گرفتن یک رشته به عنوان یک توییت، ابتدا با استفاده از تابع **text_preprocessor** متن پیش‌پردازش شده‌ی توییت را بدست آورده و سپس آن را به‌عنوان ورودی **query** تابع **vectorizer** در حالت **train= False** داده و بردار متناظر با آن توییت را گرفته و با استفاده از فایل موجود برای مدل آموزش دیده شده، آن مدل را وارد حافظه کرده و با استفاد از تابع **predict** مدل، دسته‌ی پیش‌بینی شده برای این توییت را خروجی می‌ده. (مثبت یا منفی بودن احساس توییت)

- دسته‌بند خطی/ آموزش و آزمایش

در این قسمت در سلول‌هایی نمونه‌ای از ساخت کلاس **LinearClassifier** و عملکرد توابع اصلی آن است.

- دسته‌بند BERT / تعریف دسته‌بند

این قسمت شامل دو کلاس است که در زیر هر یک را شرح می‌دهیم.

- کلاس DataSequence(tf.keras.utils.Sequence):

این کلاس یک کلاس ارث برده از کلاس **tf.keras.utils.Sequence** است که برای دسته کردن (تبدیل داده‌ها به تعدادی **batch**) داده‌ها برای دادن به‌عنوان ورودی به مدل دسته‌بند **BERT** که در کلاس بعدی معرفی می‌شود استفاده می‌شود.

- تابع __init__(self, x_set, y_set, batch_size):

دو آرایه **x_set** و **y_set** است که در آن‌ها به‌ترتیب رشته‌ی هر توییت و کلاس متناظر با آن قرار دارد و **batch_size** نیز اندازه‌ی هر دسته است ورودی‌های این تابع هستند که آن‌ها در متغییرهایی با نام مرتبط ذخیره می‌کند.

- تابع __len__(self):

این تابع تعداد دسته‌هایی که پس از دسته‌بندی داده‌ی ورودی خواهیم داست را محاسبه کرده و خروجی می‌دهد.

- تابع `__getitem__(self, idx)`:

این تابع با گرفتن یک عدد صحیح `idx` به عنوان اندیس دسته‌ی خواسته شده، تمام اعضای دسته‌ی متناظر با آن را به عنوان یک زوج دوتایی به صورت `(x_batch_idx, y_batch_idx)` خروجی می‌دهد.

- کلاس `BertClassifier`:

این کلاس به‌منظور ساخت یک مدل برای دسته‌بندی و تعریف توابع مختلف برای آن مدل ساخته شده است. در این پروژه از یکی از مدل‌های `expert bert` معرفی شده توسط `TensorFlow Lab` به اسم `wiki_books/sst2` استفاده شده است و دلیل انتخاب این مدل این است که این مدل برای تحلیل احساسات بر دیگر مدل‌های خبره توسط معرفان آن ارجحیت داده شده است.

- تابع `__init__(self,train_file_path,test_file_path,saved_model_path = transformer_model_path)`:

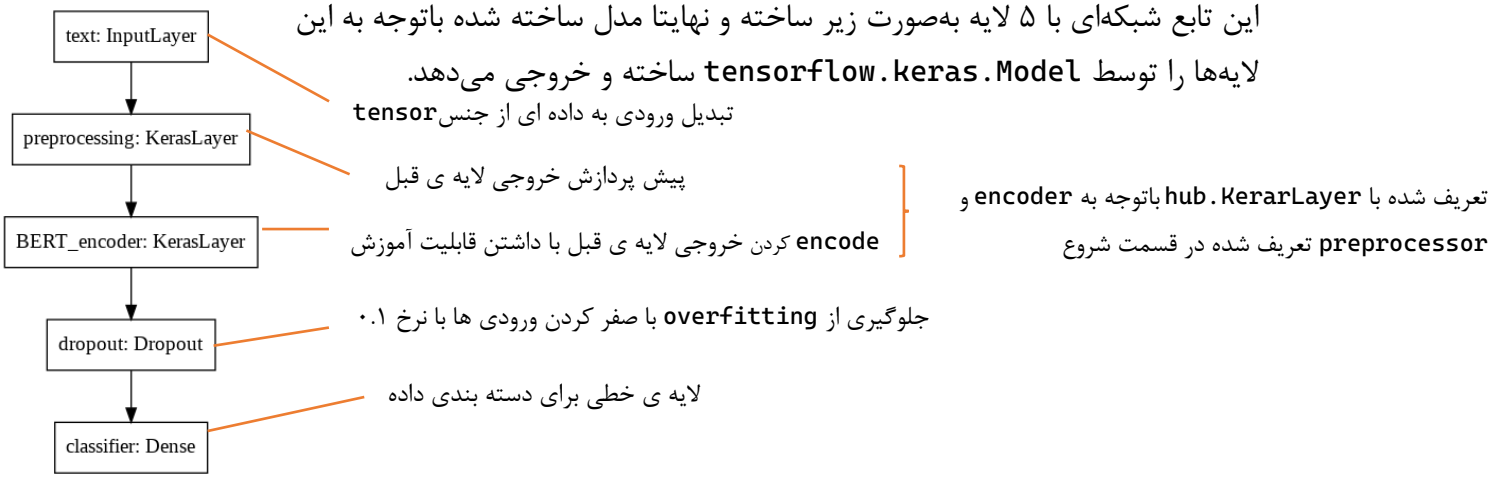
این تابع مسیر (همراه با نام فایل) دو فایل آموش و آزمایش گرفته و درمتغییری با نامی مناسب ذخیره کرده، همچنین با گرفتن نام مسیر (شامل نام پوشه‌ی مدل) پوشه‌ای که متغیرهای این مدل در آن ذخیره می‌شوند را در متغیری با نام مناسب ذخیره می‌کند (درصورت عدم مقدار دهی این ورودی، مسیر تعریف شده در متغیر `transformer_model_path` را به‌عنوان مقدار در نظر می‌گیرد.)

همچنین متغیرهایی با نام‌های `train_size`، `val_size` و `test_size` را باتوجه به متغیری با همین نام در ثابت‌ها مقدار دهی می‌کند که به‌ترتیب تعداد داده‌های درنظر گرفته شده برای آموزش، ارزیابی (`validation`) و آزمایش (`evaluation`) است. (به دلیل بسیار زیاد بودن حجم داده و محدودیت زمانی، برروی تمامی داده‌ی دراختیار مدل آموزش داده نشده است و تنها بر روی مقداری از آن آموزش و آزمایش شده است.)

علاوه بر این‌ها، لینک `encoder` و `preprocessor` مدل انتخابی را باتوجه به لغت‌نامه‌ی تمامی لینک‌ها که در سلول ثابت‌ها معرفی شده است را نیز در متغییری با نام مناسب ذخیره می‌کند و در نهایت یک متغیر با نام `model` می‌سازد که قبل از آموزش مدل یا خواندن فایل آن، برابر با `None` است.

- تابع `build_classifier_model(self)`:

این تابع شبکه‌ای با ۵ لایه به‌صورت زیر ساخته و نهایتا مدل ساخته شده باتوجه به این لایه‌ها را توسط `tensorflow.keras.Model` ساخته و خروجی می‌دهد.



- تابع `train(self, batch_size= 32, epochs= 5)`:

این تابع اندازه‌ی دسته و تعداد تکرار آموزش را ورودی می‌گیرد (در صورت عدم مقدار دهی این دو در زمان صدا زدن این تابع، این دو مقدار به‌ترتیب برابر ۳۲ و ۵ قرار می‌گیرند). ابتدا با استفاده از مسیر فایل داده‌ها آموزش و مقدار `val_size` و `train_size` از داده‌ی آموزش جدا کرده و شی‌ای از کلاس `DataSequence` برای داده‌ی آموزش و ارزیابی (باتوجه به اندازه‌های تعیین شده و سایز دسته که ورودی تابع `train` است) ساخته تا در ادامه به‌عنوان ورودی برای آموزش به مدل دهد. پیش از آن تابع `loss`را با استفاده از تابع `BinaryCrossentropy` کلاس `tensorflow.keras.losses` در حالت `from_logits= True` (به دلیل عملکرد بهتری که توسط سازندگان آن گفته شده است دارد) ساخته (که در این حالت لزوم برچسپ‌ها باید ۱ و ۰ باشند) و متغیر `metrics` را برابر با `tf.metrics.BinaryAccuracy()` تعریف کرده و سپس `optimizer` ای از نوع `adamw` می‌سازیم. حال مقدار مدل تعریف شده در تابع شروع این کلاس را برابر با خروجی تابع `build_classifier_model` قرار می‌دهیم و سپس باتوجه به `loss` ، `metrics` و `optimizer` تعریف شده به عنوان ورودی تابع `compile` این مدل، آن را صدا می‌زنیم. نهایتا با صدا زدن تابع `fit` مدل تعریف شده، و ورودی دادن تعداد دفعات آموزش ورودی (`epochs`) و دو شی ساخته شده به‌عنوان داده‌ی آموزش و ارزیابی به این تابع مدل را آموزش می‌دهیم.
پس از اتمام فرایند آموزش مدل را در مسیر داده شده در زمان ساخت این کلاس ذخیره می‌کنیم. (با استفاده از تابع `save` مدل)

- تابع `evaluate(self)`:

ابتدا در صورت `None` بودن مقدار مدل تعریف شده برای کلاس، اول با استفاده از مسیر ذخیره شدن مدل، سعی در خواندن آن از روی آن فایل کرده و در صورت عدم وجود آن، تابع `train` را صدا می‌زند (در این صورت مطمئنا این متغییر مقدار دارد). سپس باتوجه به مسیر داده شده به‌عنوان محل داده‌های آزمایش، ابتدا آن‌را وارد حافظه کرده و با استفاده از `test_size` تعریف شده در تابع شروع این کلاس، به این اندازه از داده‌های آزمایش را انتخاب کرده، برای هر یک، از مدل آموزش دیده یک عدد بین ۰و ۱ را گرفته که در صورتی که عدد کمتر از ۰.۵ باشد این داده به کلاس منفی و در غیر این صورت به کلاس مثبت تعلق دارد. در این حین، دو لیست که در یکی کلاس انتخابی مدل و در دیگری کلاس حقیقی هر یک از داده‌ها وجود دارد را می‌سازیم، نهایتا مانند ارزیابی مدل خطی در گذشته مدل را ارزیابی کرده و نتایج را خروجی می‌دهیم. (`report` و `confusion matrix`)

- تابع `query(self, tweet)`:

این تابع یک رشته (متن توییت) را به‌عنوان ورودی می‌گیرد و ابتدا با استفاده از تابع `text_preprocessor` متن آن را پیش‌پردازش کرده و سپس با استفاده از مدل تعریف شده (در صورت مقدار`None` داشتن آن مانند تابع قبل عمل‌کرده) یکد عدد بین ۰ و ۱ گرفته و دسته‌ی پیش‌بینی شده برای رشته‌ی ورودی همراه با احتمال آن را خروجی می‌دهد.

- دسته‌بند BERT / آموزش و آزمایش:

در این قسمت در سلول‌هایی نمونه‌ای از ساخت کلاس `BertClassifier` و عملکرد توابع اصلی آن است.

ارزیابی و نتایج

در این‌جا نتایج ارزیابی هر دو مدل همراه عملکرد آن‌ها برروی چند نمونه متن توییتر آمورده شده است.

model	classes	precision	recall	f1-score	accuracy	macro avg	weighted avg	confusion matrix
SVM	0	0.74	0.70	0.72	0.73	0.73	0.73	112393 47658 39063 120886
	4	0.72	0.76	0.74				
BERT	0	0.78	0.81	0.79	0.78	0.78	0.78	8230 1987 2353 7430
	4	0.79	0.76	0.77				

در زیر نیز نمونه‌ای از عملکرد این دو مدل برروی چند نمونه از داده‌های توییتر را می‌توان دید.

Tweet	SVM	BERT
@PrinceDavey aww no invite?? lol jk. coolness for the day off!	positive	negative (0.4803)
Note to you all: don't go to the choclate bar @ schiphol!! it is passengers only	positive	negative (0.4059)
eally let down by gossip girl...it's all I have to make my Mondays good and all they give are reruns...	positive	negative (0.4240)
@garretjiroux do u write back on twitter? i miss ya garee... x	positive	negative (0.4677)
@phlaimeaux where are you?	positive	negative (0.4912)
it was not that bad	negative	positive (0.9393)
No TravoRadio this morning. BlipFM is down.	positive	negative (0.4678)

همان‌طور که انتظار می‌رفت مدل `BERT` عملکرد بهتری از دسته‌بند خطی `SVM` دارد با این که بردارسازی آن توسط `transformers` انجام شده است و بر روی مجموعه داده‌ی بیشتری آموزش دیده است.

همچنین نکته‌ی قابل توجه‌ای که وجود دارد این است که باتوجه به داده‌ی اولیه که خود نیز توسط الگوریتم‌های یادگیری ماشین برچپ‌گذاری شده است، ممکن است مدل‌ها را به سمتی برده که برای برخی از جملات برچسپی تعلق داده که توسط یا انسان، برچسپ معقولی نباشد.

مراجع

Gautam, G. a. (٢٠١٤). Sentiment analysis of twitter data using machine learning approaches and semantic analysis. In ٢٠١٤ *Seventh International Conference on Contemporary Computing (IC٣)* (pp. ٤٣٧–٤٤٢). IEEE.

Go, A. a. (٢٠٠٩). Twitter sentiment classification using distant supervision. *CS٢٢٤N project report, Stanford*, ٢٠٠٩.