گزارش پروژه سوم یادگیری عمیق
(Word2vec)

18 دی ماه 1399

غزاله محمودی

96522249

منابع :

Another Twitter sentiment analysis with Python — Part 11（CNN + Word2Vec）

Machine Learning — Word Embedding & Sentiment Classification using Keras

models.keyedvectors – Store and query word vectors — gensim

لینک کدهای اجرایی:

- https://colab.research.google.com/drive/1Jlb9nGzbDag2fanKvTz9e-icGOdxxhjt?usp=sharing
- https://colab.research.google.com/drive/1Z8TcFIrr6Flp2qhsdx1-xZq2ZCS4yvBR?usp=sharing

بخش اول

در این بخش قصد داریم با استفاده از ماژول gensim بردار word 2 vec را برای هر کلمه حساب کنیم.
ابتدا لازم است دیتاست را برای استفاده آماده کنیم. به کمک pandas و نظرات موجود را استخراج می کنیم. در نهایت DataFrame به صورت زیر حاصل می شود.

در ادامه به ازای تمام نظرات(جملات) موجود در دیتاست کلمات موجود در آن ها را استخراج می کنیم. این کار با استفاده از tokenizer ماژول HAZM انجام می دهیم.

```
[ ]  1  # Storing comments in list
     2  comments = [comment for comment in df.comment]
```

```
[ ]  1  # converting each sentence to list of words and inserting in sents
     2  sents = [word_tokenize(comment) for comment in comments]
```

```
     1  # example
     2  sents[0]
```

```
['لطفا', 'رنگ', 'سفید', 'رو', 'هم', 'موجود', 'کنید']
```

برای به دست آوردن word2vev به کمک Gensim **تعدادی پارامتر قابل تنظیم** دارد که در ادامه به بررسی آن ها می پردازم.

- **Size**
  - این پارامتر تعیین کننده سایز vector برای نمایش هر word یا token است. هر چه دیتاست محدود تر و کوچکتر باشد این عدد نیز کوچک تر در نظر گرفته می شود و هر چه دیتاست بزرگتر باشد(کلمات unique بیشتری داشته باشد) باید اندازه vector بزرگتر در نظر گرفته شود. تجربه نشان داده اندازه بین 100 تا 150 برای دیتاست های بزرگ مقدار مناسبی است.
- **Windows**
  - این پارامتر تعیین کننده بیشترین فاصله مابین کلمه اصلی و همسایه های آن می باشد.از لحاظ تئوری هر چه این سایز کوچکتر باشد کلماتی که بیشتر ارتباط را به یکدیگر دارند به عنوان خروجی برمی گرداند. اگر تعداد داده به اندازه کافی بزرگ باشد سایز پنجره اهمیت زیادی ندارد اما باید این نکته را در نظر گرفت که این سایز نباید خیلی بزرگ یا بیش از حد کوچک باشد. اگر درباره انتخاب آن اطمینان نداریم بهتر است از مقدار پیش فرض استفاده کنیم.
- **Min_count**
  - این پارامتر حداقل تکرار کلمه در دیتاست را نشان می دهد که در صورتی که کلمه ای به این تعداد تکرار شود در word embedding مورد توجه قرار می گیرد و در غیر این صورت کنار گذاشته می شود. تعیین این عدد در دیتاست های بزرگ برای کنار گذاشتن کلمات کم اهمیت که غالبا کم تکرار می شوند مناسب است. همچنین در مصرف بهینه مموری و حافظه هم تاثیر دارد.
- **Workers**
  - این پارامتر تعداد thread هایی که در عملیات اجرا مورد استفاده قرار می گیرد را مشخص می کند. برای عملیات بهینه سازی و افزایش سرعت اجرا در سیستم هایی که قابلیت پردازش موازی دارند.
- **Iter**
  - تعداد epoch یا دفعاتی که الگوریتم اجرا می شود و مدل آموزش می بیند.

- **seed**
  - برای مقدار دهی رندوم اولیه استفاده می شود.
- **sg**
  - به به صورت skip Gram , CBOW انجام می شود.

در این روش به ازای هر کلمه برداری به طول Size آموزش داده می شود که نشان دهنده ویژگی های کلمه مورد نظر است.

نتایج و تحلیل ها :

- مشاهده می شود هر چه min_count را افزایش دهیم بردار های با معنی تری تولید می شود. همچنین تعداد vocab یا کلماتی که برای آن ها word embedding به دست می آوریم کاهش می یابد. این کاهش نشان دهنده این است که تنها کلمات مهم و با معنی که به تعداد خوبی در متن هستند را مورد بررسی قرا داده است.
- هر چه کلمه پر تکرار تر باشد مدل مفهوم آن را بهتر درک می کند.

در ادامه به بررسی چند مثال و انجام تحلیل می پردازیم.

- به عنوان مثال برای model زیر 15 تا از مشابه ترین کلمات با واژگانی را مورد بررسی قرار دادم.
  - در این مدل اندازه size = 64 , min_count=1

size = 64 /min count = 1

```
[ ]  1  model = Word2Vec(sentences=sents, size=64, window=10, min_count=5, workers=5)
```

```
[ ]  1  len(model.wv.vocab)
```

```
32492
```

برای کلمات 'عالی' , '15 'S4 کلمه مشابه که بر اثر learning یاد گرفته شده را در آوردم.

```
[ ]  1  # Another example
     2  model.most_similar('عالی', topn=15)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.p

[('0.8146510124206543', 'خوب'),
 ('0.7441064119338989', 'عالیی'),
 ('0.7411764860153198', 'عالیه'),
 ('0.7372027635574341', 'محشر'),
 ('0.7327728271484375', 'بینظیر'),
 ('0.6945533156394958', 'باکیفیت'),
 ('0.6856202483177185', 'روان'),
 ('0.6829501390457153', 'فوقالعاده'),
 ('0.682911217212677', 'عاااالی'),
 ('0.6690618395805359', 'عالیش'),
 ('0.6622269153594971', 'عالییی'),
 ('0.6500575542449951', 'افتضاح'),
 ('0.6399007439613342', 'خوبی'),
 ('0.636671781539917', 'بینظیره'),
 ('0.6346702575683594', 'محشره')]
```

همانطور که ملاحظه می شود کلمات خوبی در آمده اما کلمه ای به مانند افتضاح نیز در میان کلمات دیده می شود.

حال برای همین دو کلمه 15 تا از مشابه ترین کلمات را برای مدل جدید با min_count = 500 , size = 64 در آوردم.

size = 64 / min count = 500

```
[ ]  1  model_1 = Word2Vec(sentences=sents, size=500, window=10, min_count=45, workers=5)
```

```
[ ]  1  len(model_1.wv.vocab)
```

```
8772
```

لازم به ذکر است طبق پیش بینی ها vocab size کاهش یافته است. زیرا تنها کلماتی مورد بررسی قرار گرفته که بیش از 500 بار در دیتاست تکرار شده اند.

همانطور که مشاهده می شود کلمات مشابه کیفیت بالاتری دارند و اشتباه دفعه پیش (کلمه افتضاح) رخ نداده است که این خود نشانی از بهبود یافتن کیفیت embedding و فیچر ها و معانی های استخراج شده از از کلمات است. شاید عدد 500 برای min_count عدد به نسبت بزرگی است اما در این مثال به وضوح تاثیر آن بر بهترین فهمیدن کلمه و جلوگیری از اشتباهات مشهود است.

```
[ ]  1  # Another example
     2  model_1.most_similar('S4', topn=15)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:

[('S3', 0.8089423179626465),
 ('S5', 0.7992942929267883),
 ('SIII', 0.7799493074417114),
 ('Samsung', 0.7087080478668213),
 ('GALAXY', 0.7049573659896851),
 ('S', 0.6980941295623779),
 ('Mini', 0.6956166625022888),
 ('Galaxy', 0.6939382553100586),
 ('S6', 0.6883549690246582),
 ('note3', 0.6860137581825256),
 ('s4', 0.679043173789978),
 ('III', 0.6780843734741211),
 ('s2', 0.6754888296127319),
 ('S7', 0.6732395887374878),
 ('A5', 0.6699981689453125)]
```

در رابطه با این کلمه اگر در دو مدل به مقایسه بپردازیم به نتایج جالبی دست می یابیم. در مدل دوم که قبل تر ذکر کردم که بهتر معنا مفهوم کلمات را درک کرده مشاهده می شود کلماتی که به عنوان کلمات مشابه S4 آوردند همگی اکثرا (به جز 2 مورد) جز برند سامسونگ می باشند. می توان نتیجه گرفت این مدل علاوه بر این که فرا گرفته این کلمه گوشی موبایل است، اینکه به کارخانه سامسونگ نیز مربوط می شود را فراگرفته و این خود نکته بسیار جالبی است.

به بررسی مثالی دیگر می پردازیم.

```
result = model.similar_by_word('یخچال')
most_similar_key, similarity = result[0]  # look at the first match
print(f"{most_similar_key}: {similarity:.4f}")
```

```
لباسشویی: 0.8549
```

در این مثال از مدل کلمه مرتبط با یخچال را خواستیم که لباسشویی را بازگردانده و میزان شباهت 0.8 است که عدد قابل توجهی است و نشان دهنده کیفیت مدل است.

در ادامه به بررسی دو کلمه مرتبط و غیر مرتبط می پردازیم.

اعداد نمایانگر این است که مدل کلمات مرتبط و غیر مرتبط را به خوبی می تواند از هم تفکیک و این نشان دهنده کیفیت مدل است.

```
# Check similarity
model.similarity('گوشی', 'مشکی')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launc

-0.0522618
```

البته مدل نسبت به کلماتی که به تعداد کافی ندیده نتایج مناسبی نمی دهد. که خب امری بدیهی است. همچنین کم تکرار بودن این کلمه در مدل با توجه به میزان شباهت شبیه ترین کلمه ها مشخص است.

و البته مدل در مقابل کلمات پرتکرار بسلر خوب مفهوم را یاد میگیرد.

نتیجه کلی اینکه مدل هایی که در این بخش مورد بررسی قرار دادم معنا و مفهوم کلمات را به حد خوبی فرا گرفته اند. یک مدل 64 تا 150 تایی برای یادگیری مفهوم کفایت می کند. با تنظیم مقدار min_count فیچر های پیدا شده با معنا تر هستند.

همچنین تعداد تکرار کلمه در متن برای یادگیری مناسب مدل امری ضروری است.

بخش دوم

در این قسمت هدف این است به کمک embedding کلمات بر روی دیتاست دیجی کالا sentiment Analysis را انجام دهیم.

سپس با توجه به اینکه برچسب 1 به معنی نظر مثبت و -1 به معنی نظر منفی می باشد و صفر به معنا بدون لیبل گذاری است. بنابراین تصمیم گرفتم تنها label های 1 و -1 را نگه دارم و لیبل 0 را کنار بگذارم. و بدین ترتیب به ازای هر جمله ورودی به آن لیبل 1 یا -1 به معنای مثبت یا منفی بودن نظر می دهم. بنابراین با یک تسک binary classification روبرو هستیم.

در این قسمت لیبل های صفر را حذف کردم.

```
1  df = df[df.score != 0]
2  df.head()
```

	score	comment
5	1	... سلام من سه هفتس این گوشی رو خریدم واقعا عالیه
6	1	...دیجی جون خدایی موجودش کن خیلی وقته منتظریم.. خ
12	1	همه چیزش معرکه ست تو این رنج قیمت مخصوصا (فورجیش)
17	1	...خیلی عالیه بنظر من اگه این رو بخری خیلی سود کر
20	1	...بابا تو رو خدا شگفت انگیزش کنین قبل عید این گو

در ادامه با دیتاست را به نسبت 20 به 80 به داده های تست و آموزشی تقسیم می کنیم. این کار را به کمک train_test_split کتابخانه sklearn انجام دادم.

در ادامه باید text ورودی را برای ورودی شبکه عصبی مناسب کنیم.
بنابر این مراحل زیر را بر روی داده های لغوی ورودی انجام می دهیم تا ورودی مناسب برای شبکه عصبی تولید شود.

- Tokenizer
  - توکن سازی فرآیند تبدیل داده های حساس به داده های غیر حساس به نام "توکن" است که می تواند در یک پایگاه داده یا سیستم داخلی مورد استفاده قرار گیرد بدون اینکه آنها را در دامنه خود قرار دهد. با جایگزینی داده های اصلی با مقدار نامرتبط با همان طول و قالب ، از رمزگذاری می توان برای ایمن سازی داده های حساس استفاده کرد. سپس نشانه ها برای استفاده به سیستم های داخلی سازمان ارسال می شوند و داده های اصلی در یک دیکشنری به ذخیره می شوند.
- Fit_to_text
  - در این قسمت به هر کلمه یک مقدار یکتای integer نسبت می دهیم. اعدادی که در این قسمت به vocab ها نسبت می دهیم همگی عددی بزرگتر از یک می باشند. عدد صفر هم برای padding در نظر میگیریم.
- Text_to_sequence
  - جمله ورودی را به دنباله ای از اعداد تبدیل می کند. این اعداد در واقع بع دیکشنری که در قسمت قبل تولید شده مپ می شوند. به جای ورود خوده کلمات به مدل عددی به نمایندگی آن ها به مدل وارد می شود.
- Pad_sequence
  - می دانیم ورودی شبکه عصبی طول ثابتی دارد. اما جملات طول متفاوتی دارند و این تناقضی به وجود می آورد. بنابراین برای جلوگیری از مشکلاتی که در ادامه به وجود می آید باید طول همه جملات یکسان باشد و بدین منظور به اول یا انتها جمله padding اضافه می کنیم که برابر مقدار صفر است و صرفا برای هم سایز شدن همه جملات است. طول padding را به حدی می گیریم که سایز جمله برابر سایز ماکزیمم جمله موجود در دیتاست شود.2 نوع padding به صورت های post , pre داریم که اولی به آخر جمله صفر اضافه می شود و در دومی به ابتدا جمله صفر اضافه می شود. برای یادآوری لازم به ذکر است که صفر همان padding است.

مراحلی که به اختصار در قسمت قبل توضیح دادم به صورت زیر پیاده سازی می شوند.
ابتدا برای جدا کردن لغات هر جمله با ابزار های موجود آن ها را tokenize می کنیم. سپس متن را بر روی دیتاست موجود fit می کنیم و در ادامه در تلاشیم هر جمله را به sequence مپ کنیم. در انتها باید طول sequence ها یکسان کنیم.

ماکزیمم اندازه بردار را برابر اندازه بزرگترین جمله در نظر گرفتم.
به عنوان مثال برای جمله 'گوشی خوبیه' بردار متناظر مرحله text_to_sequence به صورت زیر است.

```
گوشیه خوبیه
[51, 52]
```

در نهایت چون باید سایز ورودی ها یکسان باشد، با پدینگ این اختلاف سایز را جبران می کنیم. دو نوع padding post , pre موجود است. در اینجا از 'pre' استفاده کردم.

```
گوشیه خوبیه
[51, 52]
[ 0  0  0 ...  0 51 52]
```

حال sequence تولید شده را به عنوان ورودی به شبکه دادم. این ورودی مناسب شبکه عصبی می باشد.

در اینجا طول بردار درون نگاری برابر با 100 در نظر گرفته شده است. ماکزیمم طول جملات را 2000 فرض کردم(با بررسی های انجام شده بلند ترین نظر طولی در حدودی 2600 دارد و من عدد 2000 را به عنوان بزرگترین طول در نظر گرفتم. جملات بزرگتر از آن قسمتی تز خود را از دست می دهند. جملات کوچکتر نیز تا رسیدن به طول 2000 پدینگ میگیرند)
همچنین در شبکه عصبی batch_size = 2048 گرفتم بدین معنی که هر 2048 داده یکبار وزن ها آپدیت می شود.
تعداد epoch = 4 می باشد بدین معنی که آموزش بر روی کل داده های آموزشی 4 بار تکرار می شود.

چون تسک binary classification است loss = 'binary_crossantropy' در نظر گرفتم. متریک را دقت و اپتیمایزر adam انتخاب شد.

- مدل

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history = model.fit(train_sequences_matrix, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(test_sequences_matrix, y_test))
```

```
Epoch 1/4
313/313 [==============================] - 50s 156ms/step - loss: 2.4383 - accuracy: 0.9201
Epoch 2/4
313/313 [==============================] - 48s 155ms/step - loss: 2.5404 - accuracy: 0.9167
Epoch 3/4
313/313 [==============================] - 49s 155ms/step - loss: 2.4456 - accuracy: 0.9198
Epoch 4/4
313/313 [==============================] - 49s 155ms/step - loss: 2.6272 - accuracy: 0.9139
```

- نتایج

.

نمودار بالا که بر روی داده های تست رسم شده بالانس نبودن داده ها را به وضوح نشان می دهد

در هر دو مدل به تقریب های خوبی رسیدیم. البته باید به نکته ای توجه کرد که با یک دیتاست unbalance روبرو هستیم. در این مجموعه ها معمولا به جای دقت معیار های دیگر بهتر از دقت نشان دهنده میزان پیشرفت هستند. همان طور که واضح است loss مقادیر مرتبط تری نشان می دهد.

شبکه با همین تعداد اپیاک به نتایج مناسبی رسید. برای بهبود شبکه لازم است دیتا را بالانس کنیم به طوری داده های لیبل +1 و -1 به تناسب هم باشند.