دانشگاه صنعتی امیرکبیر

دانشکده‌ی علوم‌کامپیوتر

گردآورنده:

شیده هاشمیان

شماره دانشجویی:

۹۶۱۳۴۲۹

تمرین چهارم درس پردازش زبان طبیعی

عنوان : پرسش و پاسخ

استاد درس: دکتر اکبری

پاییز ۹۹

این پیاده‌سازی متشکل از یک فایل ipynb. به نام NLP_Assignment_4.ipynb است. که هریک از توابع موجود در کلاس آن در زیر توضیح داده شده است.

برای اجرای برنامه، ابتدا یک کپی از فایل داده در google drive خود ایجاد کرده و آدرس جایی که در آن قرار دارد را در سلولی که برای unzip کردن است قرار دهید (copied_data_path). همچنین مسیر data_path (محلی که داده‌های آموزش و آزمون درآن قرار دارند) و documents_path (محلی که داده‌هایی که در حین اجرا در آن ذخیره می‌شوند) را درصورت لزوم به آدرس مطلوب تغییر دهید.

لازم به ذکر است این کد تنها پیاده‌سازی بخش یک تمرین است.

۱. یافتن جمله حاوی جواب صحیح (کلاس Answer_Sentence_Detector):

- تابع __init__(train_data_path, k) :

این تابع با گرفتن مسیری که داده‌های آموزش در آن قرار دارند و k که عددی برای حد بالای جملات پاراگراف است، (باتوجه به این که به عنوان hyperparameter نمی‌توان در نظر گرفت، چون لزوما با بزرگ بودن آن، نتیجه بهتر خواهد بود، با دیدن میانگین جملات تمام پاراگراف‌ها و بررسی نسبتی از داده‌ها که از این شرط عبور می‌کنند، ۷ برای این مقدار در نظر گرفته شده است.) ابتدا داده‌های مورد نیاز برای این کلاس را از داده استخراج می‌کند و در ساختاری به شکل زیر برای کلاس در نظر می‌گیرد.

```
- self.paragraph_dict = {par_id: par_val, …}
- self.quesrion_answer_dict = {‘answer’: [{‘answer_sart’: int_start,’text’: ‘ans1’}                                                    ,…],
                               ‘question’: [‘q1’, …]}
```

سپس با استفاده از تابع preprocess ساختار آن‌ها را به شکل مطلوب تغییر داده.

لازم به ذکر است تمامی متغییرها با نامی مشابه با paragraph_dict و question_answer_dict ساختاری ی مشابه ساختار معرفی شده را دارند. (تنها در زمان پاسخ کوئری، کلید پاسخی وجود ندارد و مابقی تماما با همین ساختار است)

- تابع preprocess(k) :

با گرفتن K که همان عدد ورودی در تابع قبل است را ورودی گرفته و ابتدا با استفاده از تابع sent_tokenize پاراگراف‌هایی که جملت بیشتری از حد مجاز را داشته حذف می‌کند، سپس برای لیست سوالات، لیست خروجی تابع POS_tagger (با دادن لیست اولیه‌ی سوالات به‌عنوان ورودی تابع) را جای‌گزین می‌کند.

- تابع POS_tagger(questions_list) :

این تابع لیستی از رشته‌ها (در این جا سوالات است) را ورودی گرفته، و برای هر سوال تنها توکن‌هایی که یکی از نقش‌های آمده در لیست POS_taggs (که در سلول ثابت‌ها آمده است) را دارند در لیستی جدید نگهداری می‌کند و در انتها لیست جدید رشته‌ها را خروجی می‌دهد. برای تشخیص جز کلام از CoreNLPCliet ارائه شده در کتاب‌خانه‌ی stanza که توسط دانشگاه استنفرد معرفی شده است، استفاده شده است.

- تابع sent_tokenize(paragraph) :

این تابع یک پاراگراف را ورودی می‌گیرد، و با تشخیص علامت‌های نقطه و علامت سوال (نشانه‌های پایان جمله)، رشته را از آن محل‌ها جدا کرده و لیستی از رشته‌های جملات را خروجی می‌دهد.

- تابع `sent_vectorize(paragraph_dict, question_answer_dict, train= True)` :

این تابع با گرفتن لغت‌نامه‌ای از پاراگراف‌ها، لغت‌نامه‌ای از سوالات و پاسخ‌ها و متغییر **Boolean** با نام **train**(که در صورتی که این تابع در قسمتی جز آموزش صدا زده شود مقدار **false** دارد) را ورودی گرفته، و برای هر پاراگراف، با استفاده از تابع **sent_tokennize** ابتدا لیستی از جملات پاراگراف گرفته، سپس با استفاده از تابع **encode** شی ساخته شده از کلاس **SentenceTransformer**، برای هر جمله یک بردار ارائه می‌دهد. همین کار را برای سوالات نیز انجام می‌دهد و تمامی بردارها را با ساختاری مشابه ساختار زیر نگهداری می‌کند.

```
- {'doc_vec':{‘par_id’: [‘sen_1_vec’, … ], … },
  'question_vec':[‘q1_vec’, …]
  }
```

درصورتی که **train= True** باشد، این ساختار را با نامی مناسب در مسیر **documents_path** ذخیره می‌کند. در نهایت این ساختار را خروجی می‌دهد.

همچنین شئ ساخته شده از کلاس **SentenceTransformer**، بر پایه‌ی مدل **DistilBERT** جملات را بردار می‌کند. دلیل انتخاب این مدل سرعت و عملکرد آن بوده است. (لینک داده‌های مرتبط بیشتر در **notebook** آمده است)

- تابع `calculate_vector_distances(self, type, vectorized_data, question_answer_dict, train= True)` :

این تابع با گرفتن نوع (یک عدد که نشان‌دهنده‌ی روش محاسبه است ۱: کسینوسی، ۲: اقلیدسی و ۳: ضرب نقطه‌ای است) داده‌ی بردار شده با ساختاری مشابه خروجی تابع قبل، لغت‌نامه‌ی سوالات و پاسخ‌ها و متغییر **Boolean** با نام **train**(که در صورتی که این تابع در قسمتی جز آموزش صدا زده شود مقدار **false** دارد)، برای هر سوال و جمله‌های پاراگراف متناظر با آن سوال، با توجه به روش ورودی، فاصله را محاسبه می‌کند و برای تمامی جفت سوال-جمله ها این مقدار را در متغیری با ساختار زیر نگهداری می‌کند.

```
- {'par_id':[
                ['first_question/first_sentence distance',
                'first_question/second_sentence distance',
                ...]
                ['second_question/first_sentence distance',
                'second_question/second_sentence distance', ...]
                ,...
              ]
        ,...
    }
```

در انتها، درصورتی که **train= True** باشد این ساختار را با نامی مناسب در مسیر **documents_path** ذخیره می‌کند و در نهایت این ساختار را خروجی می‌دهد.

- تابع `compare_roots(self, paragraph_dict, question_answer_dict, train= True)` :

این تابع با گرفتن لغت‌نامه‌ای از پاراگراف‌ها، لغت‌نامه‌ای از سوالات و پاسخ‌ها و متغییر **Boolean** با نام **train**(که در صورتی که این تابع در قسمتی جز آموزش صدا زده شود مقدار **false** دارد)، برای هر سوال و جمله‌های پاراگراف متناظر با آن سوال، با استفاده از **CoreNLPClient**، با تجزیه‌ی درخت وابستگی، ریشه‌ی درخت هر دو

(جمله و سوال) را یافته و با استفاده از lemma تعریف شده در خود CoreNLPClient، ریشه‌ی هر دو کلمه که ریشه‌ی درخت بودن را می‌یابد و درصورت برابر بودن ۱ و در غیر این صورت صفر را برای این جفت در متغییری با ساختار زیر در نظر می‌گیرد.

```
- {'doc_id':[
                    ['first_question/first_sentence root_sim',
                    'first_question/second_sentence root_sim',
                    ...]
                    ['second_question/first_sentence root_sim',
                    'second_question/second_sentence root_sim',
                     ...]
                    ,...
                ]
            ,...
        }
```

در انتها، درصورتی که train= True باشد این ساختار را با نامی مناسب در مسیر documents_path ذخیره می‌کند و در نهایت این ساختار را خروجی می‌دهد.

- تابع check_wh_presence(self,question_answer_dict, train= True) :

این تابع لیستی از سوالات و متغییر Boolean با نام train(که در صورتی که این تابع در قسمتی جز آموزش صدا زده شود مقدار false دارد) را ورودی گرفته و تعداد ظهور کلماتی که به عنوان کلید در لغت‌نامه‌ی wh_terms در سلول ثابت‌ها آمده است را برای هر سوال شمرده و لیت متناظر با هر سوال را به‌ترتیب در لیستی ذخیره می‌کند.

در انتها، درصورتی که train= True باشد این بیست نهایی را با نامی مناسب در مسیر documents_path ذخیره می‌کند و در نهایت آن را خروجی می‌دهد.

- تابع extract_linguistic_features(self,paragraph_dict, question_answer_dict, train= True) :

این تابع با گرفتن لغت‌نامه‌ای از پاراگراف‌ها، لغت‌نامه‌ای از سوالات و پاسخ‌ها و متغییر Boolean با نام train(که در صورتی که این تابع در قسمتی جز آموزش صدا زده شود مقدار false دارد)، با استفاده از CoreNLPClient، برای هر جمله در پاراگراف و سوالات NER های موجود در آن را پیدا کرده، آن‌هایی که کلیدهای موجود در لغت‌نامه‌ی linguistic_features در سلول ثابت‌ها وجود دارند را با توجه به مقدار آن کلید، مشابه در نظر گرفته و تعداد آن‌ها را در لیستی متناظر با آن جمله می‌شمارد. و این لیست را برای هر جمله در ساختاری به‌شکل زیر نگهداری می‌کند.

```
- {'paragraph':{‘par_id’: [‘sen_1_feature’, …], …},
                'question': [‘q1_feature’, …]
    }
```

در انتها، درصورتی که train= True باشد این بیست نهایی را با نامی مناسب در مسیر documents_path ذخیره می‌کند و در نهایت آن را خروجی می‌دهد.

- تابع `transform_data(self, sentence_linguistic_feature= None, cosine_similarity_dict = None, euclidean_distance_dict = None, dot_product_dict = None, root_comparision_dict = None, wh_presence_list = None, question_linguistic_feature = None, train= True)` :

در این تابع در حالتی که برای سنجش مدل (**evaluate**) و یا پرسش (**query**) استفاده شود (که **train= False** نشان ذهنده‌ی آن است)، خروجی توابعی که ویژگی متناظر با نام این متغیرها را تولید می‌کنند را ورودی می‌گیرد اما در حالتی که برای حالت آموزش باشد (که **train= True** نشان ذهنده‌ی آن است) به‌دلیل این که فایل متناظر با آن‌ها باید موجود باشد (به دلیل زمان‌بر بودن استخراج ویژگی برای داده‌ی آموزش، ذخیره‌ی آن‌ها در حافظه‌ی دائم پس از یک بار استخراج و وارد حافظه‌ی موقت کردن آن‌ها در صورت نیاز بهینه‌تر است) این ویژگی‌ها را ورودی نمی‌گیرد.

پس در صورتی که **train= True** باشد، سعی در خواندن داده‌ی متناظر هر ویژگی شده و در صورت موجود نبودن آن فایل، تابع متناظر با استخراج آن ویژگی را صدا زده و در نهایت تمام ویژگی‌ها برای داده‌های آموزش را در حافظه آورده.

حال با دراختیار داشتن تمام داده‌های ویژگی، آن‌ها را در ساختاری به شکل زیر در کنار هم قرار می‌دهیم.

```
- {<paragraph_id>: [
                    [ <first question,first sentence features>,
                      <first question,second sentence features>,...
                     ],
                     [ <second question,first sentence features>,
                       <second question,second sentence features>,...
                      ], ...
                    ],...
  }
```

که هرکدام از ویژگی زوج جمله-سوال به ترتیب حاوی داده‌های زیر است.

```
- <sentence_linguistic_feature>: a list of size 3
- <cosine similarity>: a float number
- <Euclidean distance>: a float number
- <dot product>: a float number
- <root comparision>: an int (0 or 1)
- <wh_presence>: a list of size 5
- <questoin_linguistic_feature>: a list of size 3
```

لازم به ذکر است که برای ویژگی‌هایی که به صورت لیست هستند، تنها مقادیر آن‌ها را در نظر گرفته، پس نهایتا برای هر جفت جمله-سوال آرایه‌ای ۱۵ عضوی از ویژگی خواهیم داشت.

در انتها، درصورتی که **train= True** باشد این بیست نهایی را با نامی مناسب در مسیر **documents_path** ذخیره می‌کند و در نهایت آن را خروجی می‌دهد.

- تابع `extract_label(self, paragraph_dict, question_answer_dict)` :

این تابع لغت‌نامه‌ی پاراگراف‌ها و لغت‌نامه‌ی سوالات و پاسخ‌ها را رودی گرفته و برای هر پاراگراف و پاسخ، با استفاده از **answer_star**، وجود و یا عدم وجود پاسخ در هر یک از جملات پاراگرف را بررسی کرده و در ساختاری به شکل زیر ذخیره می‌کنیم.

```
- {<paragraph_id>: [
                    [ <first answer, first sentence label>,
                      <first answer, second sentence label>, ...
                     ],
                     [ <second answer, first sentence label>,
                       <second answer, second sentence label>, ...
                      ], ...
                    ],...
  }
```

در نهایت این ساختار را خروجی می‌دهیم.

- تابع `prepare_data_for_classifier(self, transformed_data, label)`:

این تابع با گرفتن دو لغت‌نامه متناسب با ساختار خروجی تابع‌های متناسب با آن‌ها را ورودی گرفته و آن‌ها را برای ورودی دادن به دسته‌بند آماده می‌کند به این صورت که داده‌ها را به صورت جفت جمله-پرسش به ترتیب در یک لیست قرار می‌دهد، لیست بدست آمده از این تبدیل روی **transformed_data** مربوط به X مدل و لیست بدست آمده از این تبدیل روی **label** مربوط به Y مدل است. نهایتا این دو لیست را به‌ترتیب به عنوان یک زوج خروجی می‌دهد.

- تابع `fit(self, transformed_data_dict)`:

این تابع با گرفتن لغت‌نامه‌ی **transformed_data_dict** و با خروجی حاصل از صدا زدن تابع **extract_label** ، این دو لغت‌نامه را به تابع **prepare_data_for_classifier** داده و خروج آن را برای آموزش به شیئای از مدل **GaussianNB** از کلاس **sklearn.naive_bayes** داده و نهایتا مدل حاصل از آموزش را با نامی مناسب ذخیره می‌کند.

- تابع `evaluate(self, evaluation_json_path)`:

این تابع مسیر کامل داده‌ی آزمایش را ورودی می‌گیرد سپس داده‌های آن را مانند ساختاری که داده‌های آموزش را استخراج کرده، استخراج می‌کند. سپس خروجی تابع **POS_tagger** را برای سوالات، به جای سوالات قبل بذخیره کرده. حال تمام ویژگی‌های لازم را با صدا زدن توابع متناسب با آن‌ها در حالت غیر آموزش (با **train= False**) استخراج کرده و نهایتا با صدا زدن تابع **transform_data** (باز هم در حالت غیر آموزش) خروجی آن را در متغییری ذخیره می‌کند. سپس خروجی تابع **extract_label** را هم گرفته و این دو را به تابع **prepare_data_for_classifier** داده و خروجی آن را برای دادن به مدل در متغیری نگهداری می‌کند. سپس با توجه به فایل مدل ذخیره شده (در صورت وجود نداشتن مدل، تابع **fit** را برای ساخت مدل صدا می‌زند)، X خروجی تابع **prepare_data_for_classifier** را به تابع **predict** مدل داده و با استفاده از خروجی این تابع و Y تابع **prepare_data_for_classifier** ، دقت را برای مدل محاسبه کرده و خروجی می‌دهد.

دقت بدست آمده برابر با **0.7974** است

- تابع `query(self, question, paragraph)`:

این تابع ابتدا پاراگراف ورودی و سوال را به ساختار زیر در می‌آورد.

سپس با استخراج ویژگی‌های لازم (مانند حالت ارزیابی تا انتهای صدا دن تابع `transform_data`)، لیستی از لیست‌های موجود در خروجی تابع `transform_data` را برای جفت جمله-پرسش را به عنوان ورودی تابع `predict_proba` مدل داده و لیستی از احتاملات وجود داشتن یا نداشتن پاسخ در هر جمله را خروجی گرفته، سپس جمله‌ای که بیشترین احتمال قرارگیری در کلاس ۱ (که به معنی وجود جواب در آن جمله است) را دارد به عنوان خروجی می‌دهد.

همچنین لازم به ذکر است که در صورت عدم وجود مدل، مانند تابع `evaluate` تابع `fit` را برای ساخت آن صدا می‌زند.