دانشگاه صنعتی امیرکبیر

دانشکده‌ی علوم‌کامپیوتر

گردآورنده:

شیده هاشمیان

شماره دانشجویی:

۹۶۱۳۴۲۹

تمرین اول درس پردازش زبان طبیعی

عنوان : آموزش مدل های زبانی

پاییز ۹۹

این پیاده‌سازی متشکل از چهار فایل py. است که هریک از آن‌ها و توابع موجود در آن‌ها در زیر توضیح داده شده است.
و برای اجرای برنامه، آدرس محل train.json و valid.json را در given_data_root_path که در فایل constant.py وجود دارد وارد کنید.

۱. **ثابت‌ها (constant.py):**

این فایل شامل متغییرهایی است که در دیگر فایل‌ها مورد استفاده قرار می‌گیرند و در میان آن‌ها یکسان است که شامل آدرس پیکره‌های اولیه، آدرسی که برنامه فایل‌هایی که در طول اجرا تولید می‌کند در آن آدرس ذخیره کند هست.
همچین مجموعه‌ای از علائم نگارشی و علائم غیر الفبایی (به‌غیر از نقطه، علامت سوال و اعداد) که در پیکره‌ی آموزش موجود بود هست که در مرحله‌ی نرمال‌سازی متن از آن‌ها استفاده شود. علاوه بر این‌ها شامل ثابت‌های عددی که در دیگر بخش‌ها مورد استفاده قرار می‌گیرند مانند حدی که برای اعداد پرکاربد و بررسی برقراری توزیع power law در صورت تمرین داده شده است و میانگینی از طول جملات که در قسمت ارزیابی مدل مورد استفاده قرار می‌گیرد است.

۲. **ابزارهای پردازش زبان(LP_toolkits.py):**

این فایل متشکل از سه تابع است.

- تابع sub_alphabets که برگرفته شده از تابعی با همین نام در پکیج parsivar هست باتوجه به نیاز در این برنامه در برخی از قسمت‌ها عوض شده است که با گرفتن یک رشته در آن تمام حروفی که در این برنامه برای ما معنی‌دار هستند را به یک مجموعه حروف مشخص map می‌کند تا کلماتی که یک نگارش دارند یکسان شناسایی شوند.
- تابع normalizer نرمال‌سازی ابتدایی که شامل اجرای تابع sub_alphabets بر روی رشته‌های ورودی، حذف علائم نگارشی و علائم غیر الفبایی (به‌غیر از نقطه، علامت سوال و اعداد) و تغییر اعداد به N را انجام داده و سپس رشته‌ی نهایی را به عنوان خروجی بازمی‌گرداند.
- تابع tokenizer رشته‌ی ورودی را دریافت کرده و آرایه‌ای از کلمات اآن رشته را برمی‌گرداند.

۳. **آماده‌سازی و شناخت داده (train_data_prep.py):**

- تابع create_vocabulary_json_file(v) یک لغت‌نامه به عنوان ورودی می‌گیرد و آن را در فایل jsonای ذخیره می‌کند. از آن زمان خواندن پیکره‌ی آموزش برای یافتن تمام لغات آموزش و تعداد تکرار آن‌ها در این پیکره استفاده می‌شود.
- تابع report_token_numnber() با استفاده از لغت‌نامه‌ی ذخیره شده، تعداد کل توکن‌ها و توکن‌های یکتا را در خروجی چاپ می‌کند.
- تابع most_frequent_words() با استفاده از لغت‌نامه‌ی ذخیره شده، ۱۰۰۰۰ لغتی که بیشترین تکرار را داشته‌اند در یک فایل متنی با نام most_frequent.txt ذخیره‌سازی می‌کند.
- تابع most_frequent_words_percentage_among_all() درصد ۱۰۰۰۰ لغتی که بیتشرین تکرار را داشته‌اند به کل لغت‌ها در خروجی چاپ می‌کند.

- تابع reformat_least_frequent_words(train_doc_array) آرایه‌ای از متون موجود در پیکره‌ی آموزش را که نرمال‌سازی اولیه (آن‌هایی که توسط تابع normalizer اعمال می‌شوند) دریافت کرده سپس لغاتی که در most_frequent.txt حضور ندارند را با UKN جایگزین کرده، سپس با توجه به محل وقوع تقطه و علامت سوال جمله‌ها را جدا کرده و در فابل متنی train_sentences.txt ذخیره می‌کند.
- تابع read_training_data(raw_train_corpus_dir) آدرس فایل آموزش را دریافت کرده، با خواندن هر داده‌ی آن با استفاده از تابع normalizer آن‌ها را نرمال کرده و در آرایه‌ای ذخیره کرده و همزمان از آن برای ساختن لغت‌نامه‌ی کلمات موجود در داده استفاده می‌کند. پس از آن توابع create_vocabulary_json_file(v) ، most_frequent_words() و reformat_least_frequent_words(train_doc_array) را اجرا می‌کند.

۴. مدل زبانی (langiage_model.py):

این فایل شامل یک تابع و یک کلاس است.

- تابع evaluation_corpus_prep(raw_valid_corpus_dir) آدرس پیکره‌ی اعتبار سنجی را دریافت کرده و جمله‌های نرمال‌شده‌ی آن را در فایل متنی valid_sentences.txt ذخیره می‌کند.
- کلاس LanguageModel

- توابع unigram_constructor(self)، bigram_constructor(self) و trigram_constructor(self) مشابه نامشان، با استفاده از فایل train_sentences.txt ساختار unigram، bigram و trigram را ساخته و در فایل json با همین نام ذخیره کرده، همچنین برای هر یک کلمه‌ای بیشترین تکرار را دارد در فایلی دیگر(مانند bigram_prob.json) ذخیره می‌کنیم به عنوان مثال برای bigram به‌ازای هر کلمه، کلمه‌ای که در داده‌ها بیشترین بار بعد از این کلمه آمده را ذخیره می‌کنیم. علاوه بر این برای روش kneser-ney در دو حالت bigram و trigram داده‌ساختاری برای محاسبه‌ی continuationcount در زمان محاسبه‌ی احتمال با استفاده از این روش smoothing ذخیره‌سازی می‌کنیم. (فایل‌هایی با نام‌های trainig_bigram_continuation.json و training_trigram_continuation.json)
- تابع train(self) که متناسب با مدل زبانی، با استفاده از توابع ذکر شده در مورد قبل ساختارهای مورد نیاز برای محاسبه‌ی احتمالات را ذخیره می‌کنیم.
- توابع unigram_prob(self,sentence)، bigram_prob(self,sentence) و trigram_prob(self,sentence) مشابه نامشان احتمال رخداد هر جمله را باتوجه به فایل‌های ذخیره شده در قسمت قبل محاسبه و در خروجی چاپ می‌کنند.
- تابع prob(self,sentence) باتوجه به نوع آموزش مدل زبانی، از توابع ذکر شده در مورد قبل استفاده می‌کند.
- توابع unigram_generator(self,sentence)، bigram_generator(self,sentence) و trigram_generator(self,sentence) با استفاده از فایل‌ها bigram_prob.json و مشابه آن، با توجه به جمله‌ی ورودی کلمه‌ای که احتمال آمدن آن در ادامه‌ی جمله‌ی ورودی بیشترین است را برمی‌گرداند. (زیرا در محاسبه‌ی احتمال‌ها، باتوجه به ثابت بودن مخرج کسرها کلمه‌ای که صورت را بیشینه کند احتمال بیشینه را دارد)

- تابع generate(self,sentence) باتوجه به نوع آموزش مدل زبانی، از توابع ذکر شده در مورد قبل استفاده می‌کند و در صورت وجودکلمه‌ی پیشنهادی، آن را در ادامه‌ی جمله چاپ کرده و در غیر این صورت پیامی مناسب می‌دهد.
- تابع evaluation(self,valid_corpus_dir) با بررسی جمله به جمله‌ی فایل داده شده، کلمه‌ی اول جمله را درنظر گرفته، در صورت وجود آن در لغت‌نامه‌ی مدل، آن را (باتوجه به نوع آموزش مدل زبانی) به یکی از توابع unigram_generator(self,sentence)، bigram_generator(self,sentence) و trigram_generator(self,sentence) داده (در صورت عدم وجود در لغت‌نامه با نشان ابتدای جمله شروع کرده) تا کلمه‌ی بعدی را تولید کنند و این روند را تا ظهور نشان اتمام جمله یا رسیدن به طول جمله‌ی متوسط (ذخیره شده در فایلconstant.py) ادامه می‌دهد سپس wer آن را محاسبه کرده. پس از محاسبه‌یwer برای تمام جملات و جمع آن‌ها بر تعداد جملات تقسیم کرده و آن را در خروجی چاپ می‌کند. که این مقدار برای انواع آموزش مدل به‌صورت زیر است

- Unigram: 1.77
- Bigram: 1.35
- Tringram: 1.109

که مطابق انتظار باتوجه به پیچیده‌تر شدن نوع آموزش مدل، خطا کمتر شده است.