مستندات

علی علی‌محمّدی ۹۶۱۳۰۲۷

برای استفاده از برنامه لازم است ابتدا یک instance از کلاس LanguageModel بسازیم. به این منظور باید مقادیر زیر را به عنوان پارامترهای مدل، تعیین نماییم:

- آدرس فولدری را که دو فایل train.json و valid.json در آن قرار دارند، به عنوان ورودی اول (corpus_dir)
- عدد N به عنوان مشخص‌کننده‌ی مدل (Unigram/Bigram)
- یک متغیر از جنس Boolean به عنوان smoothing، که اگر True باشد، ***Laplace*** Smoothing اعمال می‌شود و اگر False باشد، بدون Smoothing خواهد بود. *مقدار پیشفرض این پارامتر، **False** است.*

پس از آن تابع constructor این کلاس فراخوانی می‌شود که در آن اعمال زیر به ترتیب انجام می‌شوند:

- ✓ هر دو فایل ورودی مسئله، در مدل بارگزاری می‌شود.
- ✓ کاراکترهای اضافی حذف می‌شوند. به این منظور، حروف الفبای فارسی، اعداد، علامت سؤال و نقطه را درون یک dict قرار می‌دهیم تا بتوان در O(1) به آن‌ها دسترسی داشت و مشخص کرد که آیا کاراکتری به این مجموعه تعلق دارد یا خیر. به قبل و بعد نقطه‌ها و علامت‌های سؤال، یک space اضافه می‌کنیم تا در ادامه که متن را به توکن‌ها تجزیه می‌کنیم، در توکن‌ها لحاظ شوند و به کلمه‌ی قبل خود نچسبند.
- ✓ متن را به توکن‌ها تجزیه می‌شود.
- ✓ توکن‌های عددی با حرف N جایگزین می‌شود.
- ✓ پیشنیاز مدل‌های n-gram، حتماً شامل Unigram می‌شود. به این منظور، ابتدا Unigram را آموزش می‌دهیم. به این ترتیب، کلمات پر تکرار قابل شناسایی می‌شوند. کلمات را بر اساس تعداد تکرار مرتب می‌کنیم.
- ✓ به جای مابقی توکن‌های خارج از ده‌هزار توکن اول، UNK قرار می‌دهیم.
- ✓ کلمات پرتکرار در فایل most_frequent.txt ذخیره می‌شوند.
- ✓ مقادیر مطلوب مسئله گزارش می‌شود.

- ✓ جملات در فایلی جداگانه به نام sentences.txt ذخیره می‌شوند و میانگین تعداد کلمات درون جملات محاسبه و گزارش می‌شود تا در ادامه به عنوان مقدار حد طول جمله مورد استفاده قرار گیرد.
- ✓ نمودار توزیع قانون توانی رسم می‌شود. خط رگرسیون را برای درک بهتر نمودار به آن اضافه می‌شود.

به طور مشابه، تابع آموزش Bigram نیز پیاده‌سازی می‌شود.

در تابع prob،که ورودی آن یک جمله است؛ با توجه به نوع مدل، احتمال وقوع آن جمله محاسبه می‌شود. دقت شود که smoothing برای Unigram معنای خاصی ندارد و در صورت نیاز، برای Bigram از *Laplace* Smoothing استفاده می‌شود.

در تابع generate، با توجه به نوع مدل، با استفاده از تعداد رخدادهای هر توکن که از قبل محاسبه شده بود، کلمات جمله انتخاب می‌شوند و یک جمله تولید می‌شود. ورودی این تابع، کلمه‌ی اول جمله و حد طول جمله است. در حالت پیشفرض، این حد برابر میانگین طول جملات در نظر گرفته شده است که برابر ۳۶ کلمه است.

در تابع evaluate، با استفاده از داده‌های درون valid.json، اعتبار جملات تولید شده توسط مدل سنجیده می‌شود. هر جمله‌ی این فایل، با جمله‌ی متناظر تولید شده توسط مدلِ آموزش‌دیده که کلمه‌ی ابتدایی آن‌ها یکسان است، مقایسه می‌شود. با استفاده از متریک فاصله‌ی *Levenshtein*، نرخ خطای کلمه (WER) محاسبه می‌شود. دقت شود که این معیار بر اساس مقدار اولیه‌ی N که مشخص کننده‌ی نوع مدل است گزارش داده می‌شود؛ یعنی اگر مدل Bigram باشد، این نرخ مربوط به Bigram خواهد بود. در ادامه میانگین این نرخ، با توجه به تعداد کل جملات محاسبه می‌شود و به عنوان خروجی تابع بازگردانده می‌شود.

- ❖ دقت شود که این تابع تنها دارای یک ورودی است که مشخص می‌کند که جملات تولید شده برای مقایسه، چه طولی خواهند داشت. به طور پیشفرض، این مقدار برابر میانگین تعداد کلمات جملات (۳۶) است.
- ✓ پیشنهاد می‌شود عدد کوچک‌تری برای این پارامتر در نظر گرفته شود؛ مثلاً ۱۰.
- ❖ توجه کنید که تعداد جملات درون valid.json خیلی زیاد است و این تابع از نظر محاسباتی، هزینه‌ی زیادی دارد.
- ✓ به این منظور پیشنهاد می‌شود که در حلقه‌ی for این تابع، مقدار i محدودتر شود؛ مثلاً:

```
for i in range(100):
```

به این ترتیب، زمان اجرای برنامه کاهش می‌یابد.

در پایان، Driver Codeای نوشته شده است که اجرای توابع را ممکن می‌سازد.

در صورت وجود هرگونه سؤال و یا ابهام، لطفاً با ایمیل alialimohammadi@aut.ac.ir در تماس باشید.