فایل های ضمیمه ی گزارش:

test.py
train.py
hmm.py
ex02.ipynb
در پوشه‌ی output: result1.txt, result2.txt
در پوشه‌ی models: model_01.txt , model_02.txt, model_03.txt, model_04.txt, model_05.txt

از روی فایل model_init اطلاعات ماتریس A و B و بردار احتمالات pi خوانده می‌شود. این مقادیر به عنوان مقدار اولیه به هریک از ۵ مدل داده می‌شوند. کار خواندن این پارامترها از روی فایل را تابع read_model انجام می‌دهد که هم در test.py و هم در train.py پیاده سازی شده. این تابع یک رشته به عنوان ادرس در ورودی می‌گیرد و سپس محتویات آن را خوانده و در قالب ماتریس ها و بردار A,B,pi خروجی می‌دهد. با استفاده از این اطلاعات خوانده شده از model_init یک مدل اولیه HMM تعریف می‌شود.

در train.py ادرس یک فایل داده های اموزشی مثل seq_model_0x.txt داده می‌شود. خط های این فایل به صورت لیستی از دنباله هایی که هر کدام نظیر یک خط داده‌ی آموزشی هست را تولید می‌کند. این لیست به تابع train_model از کلاس HMM داده می‌شود تا آموزش مدل صورت بگیرد. مدت زمانی برای اینکه کل دیتای آموزشی خوانده و آموزش انجام شود طول می‌کشد. این زمان در پایان نشان داده می‌شود.

بعد از اینکه مدل آماده شد با استفاده از تابع write_models_to_files در پوشه ای که ادرس ان به صورت آرگومان داده شده با همان نام ذکرشده به فرمت پارامترهایی که در model_init نوشته شده بودند نوشته و ذخیره می‌شود.

مدل ها با استفاده از کلاس HMM ساخته و آموزش داده می‌شوند. این کلاس در فایل hmm.py به طور کامل به شرح زیر پیاده سازی شده.

در این کلاس هر مدل علاوه بر ۳ پارامتر ذکر شده برای راحتی محاسبات یک لیست dic از حروف استفاده شده در مدل‌ها (حروف بزرگ ۱ تا ۶ الفبای انگلیسی) و یک عدد N که برابر تعداد وضعیت‌هاست دارد. در این کلاس ۴ تابع پیاده سازی شده.

توابع forward و backward که هر کدام به ازای دنباله ی ورودی داده شده کار می‌کنند و با مدلی که از آن فراخوانی می‌شوند مقدار p(observation|model) را محاسبه می‌کنند. تابع forward با محاسبه ی

ماتریس آلفا جلو می رود و در نهایت خود ماتریس آلفا و مقدار مجموع سطر آخر آن را به عنوان مقدار p(observation|model) برمی‌گرداند. تابع backward با محاسبه ی ماتریس بتا جلو می‌رود. و در نهایت حاصل مجموع ضرب درایه به درایه ی سطر اول بتا در pi را به همراه ماتریس بتا برمی‌گرداند. هر دو این توابع از برنامه نویسی پویا برای محاسبه ی احتمال یک دنباله مشاهده ی ورودی به ازای مدل مشخص استفاده می‌کنند.

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j)$$

فرمول

$$\beta[T][i] = 1$$

$$\beta[t][i] = \sum_{j=1}^{N} A[i][j]\, B[o_{t+1}][j]\, \beta[t+1][j]$$

پیاده‌سازی

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i\, \beta_0(i) = \sum_{i=1}^{N} p_i[i]\, \beta[0][i]$$

تخمین احتمال

$$\alpha_0(i) = \pi_i$$

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i)\, a_{ij} \right) b_j(o_{t+1})$$

فرمول

$$\alpha[0][i] = p_i[i]$$

$$\alpha[t+1][j] = \left(\sum_{i=1}^{N} \alpha[t][i]\, A[i][j] \right) \times B[o_{t+1}][j]$$

پیاده‌سازی

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{i=1}^{N} \alpha[T][i]$$

تخمین احتمال

تابع دیگری که در HMM پیاده سازی شده تابع train_model هست که محتوای فایل آموزشی یا بخشی از آن را به صورت یک لیست ورودی می‌گیرد و عمل آموزش مدل را به ازی آن روی مدل انجام می‌دهد. در هر گام از آموزش یک عنصر لیست معادل یک خط فایل داده های seq_model_0x.txt خوانده می شود. با استفاده از تابع baum_welch مقادیر به دست آمده برای پارامترهای مدل محاسبه می‌شود. دوتا ماتریس برگشتی مقادیر به صورت دوتایی هایی هستند که اگر بر هم تقسیم کنیم مقادیر جدید A وB را می‌دهند. ولی این مقادیر به صورت همین ماتریس دوتایی ها نگهداری می‌شود تا آخرین خط داده ی آموزشی هم خوانده شود. کل دوتایی های به دست آمده از تمام خط ها باهم جمع می‌شوند و در نهایت با تقسیم عنصر اول بر دوم این دوتایی ها در هر سطر و ستون که هستند درایه ی نظیر آن سطر و ستون در ماتریس های A NxN و B به دست می‌آید. برای بردار اختمالات pi هم کافی ست تمام اختمالات به دست آمده در هر خروجی baum_welch را باهم جمع زده و در آخر بر تعداد گام های حلقه تقسیم شود (میانگین) . در آخر مقدارهای new_A و new_B و new_pi جایگزین مقادیر A , B, pi در مدل می‌شوند و آموزش روی داده های ورودی به پایان می‌رسد. به ازای تعداد ایتریشن هایی که به عنوان آرگومان اول از ورودی سیستم می‌خواند، از روی داده های آموزشی می‌گذرد و پارامترهای مدل را آپدیت می‌کند.

تابع پیاده شده ی baum_welch به ازای یک خط ورودی مشخص الگوریتم باوم-ولش را پیاده سازی می‌کند. از برنامه نویسی پویا استفاده شده و با کمک ماتریس های alpha و beta که از الگوریتم های forward و backward به ازای همان ورودی آموزشی گرفته و با پارامترهای فعلی مدل مقادیر ماتریس های gamma و xi را محاسبه می‌کند. به ازای ورودی داده شده، سطر اول گاما به عنوان pi جدید و به ازای دو ماتریس A و B جدید هم دو ماتریس NxN که هر درایه اش یک دوتایی هست که بالا توضیح دادم نتیجه می‌دهد. برای A جدید به ازای کل سطرهای xi و gamma مقادیر را باید باهم جمع زد و به ترتیب در درایه های اول و دوم دوتایی ها قرار داد. برای ماتریس B جدید باید برای تمام سطرها مقادیر gamma نظیر ستون‌هایشان جمع شوند تا درایه دوم به دست بیاید. در صورتی که سطر مورد نظر نظیر مشاهده ای در ورودی باشد که مقدار مشخص نظیر سطرB را داشته باشد، این ها مجموعgamma های نظیرشان به عنوان درایه اول دوتایی ماتریس B محاسبه می‌شود.

$$\gamma_t(i) = \frac{\alpha_t(i)\,\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\,\beta_t(j)}$$

فرمول

$$\gamma[t][i] = \frac{\alpha[t][i]\,\beta[t][i]}{\sum_{j=1}^{N} \alpha[t][j]\,\beta[t][j]}$$

پیاده‌سازی

$$\xi_t(i,j) = \frac{\alpha_t(i)\,a_{ij}\,b_j(o_{t+1})\,\beta_{t+1}(j)}{\sum_{k=1}^{N} \alpha_t(k)\,\beta_t(k)}$$

فرمول

$$\xi_i[t][i][j] = \frac{\alpha[t][i]\,A[i][j]\,B[o_{t+1}][j]\,\beta[t+1][j]}{\sum_{k=1}^{N} \alpha[t][k]\,\beta[t][k]}$$

پیاده‌سازی

$$pi^* = \gamma_1$$

$$A[i][j]^* = \frac{\sum_{t=1}^{T} \xi_i[t][i][j]}{\sum_{t=1}^{T} \gamma[t][i]} = \frac{\text{از } i \text{ به } j \text{ برو}}{\text{از } i \text{ به هر جا برو}}$$

$$B[i][j]^{*} = \frac{\sum_{t=1}^{T} \gamma[t][j] \;:\; \text{if } O_t = v_i}{\sum_{t=1}^{T} \gamma[t][j]}$$

تخمین پارامتر

در فایل test.py مدلی که ادرس آن داده شده از فایل متنی با تابع read_model خوانده می‌شود. فایل داده های تست هم از ادرس داده شده خط به خط خوانده و به صورت لیستی از دنباله ها اماده میشود. مقدار p(o|model) برای هر خط به ازای هر ۵ مدل محاسبه می‌شود. این محاسبه را measure_p_o_lambda انجام می‌دهد. در write_results_to_file بهترین نتیجه یعنی بیشترین اختمالی که محاسبه شده انتخاب می‌شود و مدلی که آن نتیجه را به دست داده و همراه اندازه ی احتمال مخاسبه شده روی فایلی به نام و ادرس تعین شده نوشته می‌شود.

فایلهایی که با اجرای این فایلها به دست آمده در پوشه ی output هست. این فایل‌ها با دستورات زیر تولید شده اند. نتیجه ها با ۵ ایتریشن حاصل شده :

```
python train.py 5 ./hmm_data/model_init.txt ./hmm_data/seq_model_01.txt ./models/model_01.txt
python train.py 5 ./hmm_data/model_init.txt ./hmm_data/seq_model_02.txt ./models/model_02.txt
python train.py 5 ./hmm_data/model_init.txt ./hmm_data/seq_model_03.txt ./models/model_03.txt
python train.py 5 ./hmm_data/model_init.txt ./hmm_data/seq_model_04.txt ./models/model_04.txt
python train.py 5 ./hmm_data/model_init.txt ./hmm_data/seq_model_05.txt ./models/model_05.txt
```

و برای تولید نتایج و ارزیابی مدل روی دو مجموعه ی تست :

```
python test.py ./hmm_data/modellist.txt ./hmm_data/testing_data1.txt ./output/result1.txt
python test.py ./hmm_data/modellist.txt ./hmm_data/testing_data2.txt ./output/result2.txt
```

که نتیجه ی این اجراها به ترتیب فایل های پوشه ی models و output هستند.

قبل از این که این فایل ها تهیه شوند تمام این کد ها برای هر ۵ مدل و ساخت و ارزیابی شان در ex02.ipynb نوشته شده بود. در این فایل ۵ مدل به صورت dictionary با نام models در دسترس بود. تمامی توابع توضیح داده شده در کلاس HMM روی این عینا موجود هستند. توابع فایل train.py و test.py با اندکی تغیر اینجا وجود دارند و تغیر هم مربوز به این است که ادرس فایل ها را از ورودی سیستم دریافت شود به علاوه اینکه در هر اجرا فقط یه مدل آموزش داده شود.

در این نوت بوک یک تابع به نام measure_precision_on_testing_set1 هست که دقت نهایی result1.txt را بر حسب جواب های فایل testing_answer.txt شمارش می‌کند. هر ۵ مدل را با ۵ ایتریشن آموزش دادم و دقت را محاسبه کردم. این دقت برابر هست با ۰,۷۵۶