Lst勉強会4回目

始めに

- NPB-DAAを理解するために

pyhsmm/exampleにあるhsmm.pyの流れに沿って理解する

- 今回は右の画像の40～45行目までを読解

```
39  # Build the true HSMM model
40  truemodel = pyhsmm.models.HSMM(
41              alpha=6.,gamma=6.,
42              init_state_concentration=10.,
43              obs_distns=true_obs_distns,
44              dur_distns=true_dur_distns)
45
46  # Sample data from the true model
47  data, labels = truemodel.generate(T)
48
49  # Plot the truth
50  plt.figure()
51  truemodel.plot()
52  plt.gcf().suptitle('True HSMM')
53
```

pyhsmm/example/hsmm.py
40～45行目

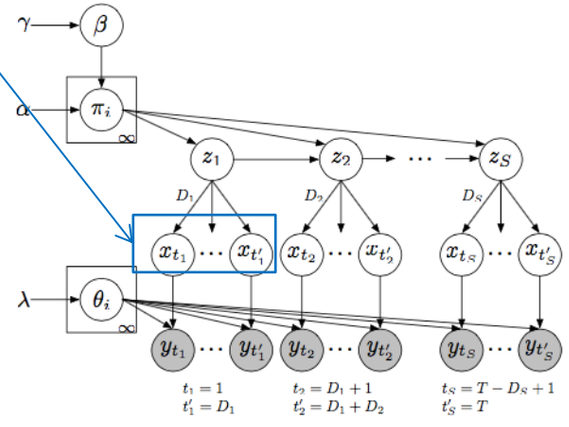
```
39  # Build the true HSMM model
40  truemodel = pyhsmm.models.HSMM(
41              alpha=6.,gamma=6.,
42              init_state_concentration=10.,
43              obs_distns=true_obs_distns,
44              dur_distns=true_dur_distns)
45
46  # Sample data from the true model
47  data, labels = truemodel.generate(T)
48
49  # Plot the truth
50  plt.figure()
51  truemodel.plot()
52  plt.gcf().suptitle('True HSMM')
53
```

- **推定のための真のHSMMモデルの生成を行う**
  - **ハイパーパラメータ**
  - **集中度パラメータ？？？**
  - **観測分布**
  - **持続時間分布**

pyhsmm/model.py>>HSMM class 439～462

```python
class HSMM(HMM, ModelGibbsSampling, ModelEM, ModelMAPEM):
    _states_class = states.HSMMStatesPython
    _trans_class = transitions.HDPHSMMTransitions
    _trans_class_conc_class = transitions.HDPHSMMTransitionsConcResampling
    _init_steady_state_class = initial_state.HSMMSteadyState

    def __init__(self,dur_distns,**kwargs):

        self.dur_distns = dur_distns

        super(HSMM,self).__init__(**kwargs)

        if isinstance(self.init_state_distn,self._init_steady_state_class):
            self.left_censoring_init_state_distn = self.init_state_distn
        else:
            self.left_censoring_init_state_distn = self._init_steady_state_class(self)

    @property
    def stateseqs_norep(self):
        return [s.stateseq_norep for s in self.states_list]

    @property
    def durations(self):
        return [s.durations for s in self.states_list]
```

- オブジェクトの生成
- HMMクラスの_init_の実行
- 打ち切り時間の設定？
- 状態列の受け渡し



- 状態持続時間の受け渡し

pyhsmm/model.py>>HSMM class 464～473

```
464     def add_data(self,data,stateseq=None,trunc=None,right_censoring=True,left_censoring=False,
465                  **kwargs):
466         self.states_list.append(self._states_class(
467             model=self,
468             data=np.asarray(data),
469             stateseq=stateseq,
470             right_censoring=right_censoring,
471             left_censoring=left_censoring,
472             trunc=trunc,
473             **kwargs))
```

- states_listに_states_classを追加

- **状態列**
- **右側の打ち切り**
- **左側の打ち切り**
- **打ち切り？**

pyhsmm/model.py>>HSMM class 475～487

```
475     def log_likelihood(self,data=None,trunc=None,**kwargs):
476         # NOTE: this only works with iid emissions
477         if data is not None:
478             self.add_data(data=data,trunc=trunc,stateseq=np.zeros(len(data)),**kwargs)
479             s = self.states_list.pop()
480             _, betastarl = s.messages_backwards()
481             return np.logaddexp.reduce(np.log(s.pi_0) + betastarl[0])
482         else:
483             if hasattr(self,'_last_resample_used_temp') and self._last_resample_used_temp:
484                 self._clear_caches()
485             initials = np.vstack([
486                 s.messages_backwards()[1][0] + np.log(s.pi_0) for s in self.states_list])
487             return np.logaddexp.reduce(initials,axis=1).sum()
```

- **尤度の獲得　（互いに独立で同一の分布に従う場合のみ）**
  - messages_backwards()**から返ってきた**betal, betastarl**を**_, betastarl**に代入**
  - pi_0:**重み**(initial_states.py)
  - log(pi_0**の総和**)**を返す**
  - log−likelihood**の式**

$$p(y_1,\ldots,y_n|\eta,\phi) = \prod_{i=1}^{n} p(y_i|\eta_i,\phi)$$

$$= \prod_{i=1}^{n} \exp\left(L(y_i|\eta_i,\phi)\right),$$

  - Lは個々の観測データの尤度

log_likelihood**の初期値を決める**

pyhsmm/internals/states.py
>>class HSMMStatesPython 457~477

```
457     def messages_backwards(self):
458         if self._betal is not None and self._betastarl is not None:
459             return self._betal, self._betastarl
460
461         aDl, aDsl, Al = self.aDl, self.aDsl, np.log(self.trans_matrix)
462         T,state_dim = aDl.shape
463         trunc = self.trunc if self.trunc is not None else T
464
465         betal = np.zeros((T,state_dim),dtype=np.float64)
466         betastarl = np.zeros((T,state_dim),dtype=np.float64)
467
468         for t in xrange(T-1,-1,-1):
469             np.logaddexp.reduce(betal[t:t+trunc] + self.cumulative_likelihoods(t,t+trunc) + aDl[:min(trunc,T-t)],axis=0, out=betastarl[t])
470             if T-t-1 < trunc and self.right_censoring:
471                 np.logaddexp(betastarl[t], self.likelihood_block(t,None) + aDsl[T-t -1], betastarl[t])
472             np.logaddexp.reduce(betastarl[t] + Al,axis=1,out=betal[t-1])
473         betal[-1] = 0.
474
475         self._betal, self._betastarl = betal, betastarl
476
477         return betal, betastarl
```

$$
\begin{aligned}
B_t(i) &:= p(y_{t+1:T} | x_t = i, F_t = 1) \\
&= \sum_j B_t^*(j) p(x_{t+1} = j | x_t = i), \\
B_t^*(i) &:= p(y_{t+1:T} | x_{t+1} = i, F_t = 1) \\
&= \sum_{d=1}^{T-t} B_{t+d}(i) \underbrace{p(D_{t+1} = d | x_{t+1} = i)}_{\text{duration prior term}} \cdot \underbrace{p(y_{t+1:t+d} | x_{t+1} = i, D_{t+1} = d)}_{\text{likelihood term}} \\
&\quad + \underbrace{p(D_{t+1} > T - t | x_{t+1} = i) p(y_{t+1:T} | x_{t+1} = i, D_{t+1} > T - t)}_{\text{censoring term}}, \\
B_T(i) &:= 1,
\end{aligned}
$$

$y_{k_1:k_2}$ denote $(y_{k_1}, \ldots, y_{k_2})$

- 右の式を計算してる？？

pyhsmm/internals/states.py
>>class HSMMStatesPython 479~493

```
479     def cumulative_likelihoods(self,start,stop):
480         out = np.cumsum(self.aBl[start:stop],axis=0)
481         return out if self.temp is None else out/self.temp
482
483     def cumulative_likelihood_state(self,start,stop,state):
484         out = np.cumsum(self.aBl[start:stop,state])
485         return out if self.temp is None else out/self.temp
486
487     def likelihood_block(self,start,stop):
488         out = np.sum(self.aBl[start:stop],axis=0)
489         return out if self.temp is None else out/self.temp
490
491     def likelihood_block_state(self,start,stop,state):
492         out = np.sum(self.aBl[start:stop,state])
493         return out if self.temp is None else out/self.temp
```

- **尤度の配列全体を累加する**
  (aBl[start]~aBl[stop])
- **尤度の配列全体を累加する**
  (aBl[start]~aBl[stop],aBl[state])?
- **尤度の配列の要素の総和をとる**
  (aBl[start]~aBl[stop])
- **尤度の配列の要素の総和をとる**
  (aBl[start]~aBl[stop],aBl[state])?

pyhsmm/model.py>>HSMM class 491～493

```
491     def generate(self,T,keep=True,**kwargs):
492         tempstates = self._states_class(model=self,T=T,initialize_from_prior=True,**kwargs)
493         return self._generate(tempstates,keep)
```

pyhsmm/model.py>>HMM class 148～162

```
148     ### generation
149
150     def generate(self,T,keep=True,**kwargs):
151         tempstates = self._states_class(model=self,T=T,initialize_from_prior=True,**kwargs)
152         return self._generate(tempstates,keep)
153
154     def _generate(self,tempstates,keep):
155         obs,labels = tempstates.generate_obs(), tempstates.stateseq
156
157         if keep:
158             tempstates.added_with_generate = True
159             tempstates.data = obs
160             self.states_list.append(tempstates)
161
162         return obs, labels
```

Keepが**Trueならば実行**

- HMM classの_generate()に値を受け渡す

- *def _generate(self,tempdata,keep)*
  - obs:観測データ
  - labels:ラベル
  - states_listに挿入
  - obsとlabelsを返す

これらはGibbs samplingで用いる

pyhsmm/model.py>>HSMM class 497～516

```
495     ### Gibbs sampling
496
497     def resample_model(self,**kwargs):
498         self.resample_dur_distns()
499         super(HSMM,self).resample_model(**kwargs)
500
501     def resample_dur_distns(self):
502         # TODO TODO get rid of logical indexing
503         for state, distn in enumerate(self.dur_distns):
504             distn.resample_with_truncations(
505                     data=
506                     [s.durations_censored[s.untrunc_slice][s.stateseq_norep[s.untrunc_slice] == state]
507                         for s in self.states_list],
508                     truncated_data=
509                     [s.durations_censored[s.trunc_slice][s.stateseq_norep[s.trunc_slice] == state]
510                         for s in self.states_list])
511         self._clear_caches()
512
513     def copy_sample(self):
514         new = super(HSMM,self).copy_sample()
515         new.dur_distns = [d.copy_sample() for d in self.dur_distns]
516         return new
```

- *def* resample_model(*self*,***kwags*)
  - resample_dur_distnsの呼び出し
  - Class HMMのresample_dur_distnsの呼び出し
- *def* resample_dur_distns(*self*)
  - 持続分布のリサンプリング
- *def* copy_sample(*self*)
  - オブジェクトコピー用？
  - Class HMMのcopy_sampleの呼び出し

```
176     ### Gibbs sampling
177
178     def resample_model(self,temp=None):
179         self._last_resample_used_temp = temp is not None and temp != 1
180         self.resample_obs_distns()
181         self.resample_trans_distn()
182         self.resample_init_state_distn()
183         self.resample_states(temp=temp)
184
185     def resample_obs_distns(self):
186         # TODO TODO get rid of logical indexing! it copies data!
187         for state, distn in enumerate(self.obs_distns):
188             distn.resample([s.data[s.stateseq == state] for s in self.states_list])
189         self._clear_caches()
190
191     def resample_trans_distn(self):
192         self.trans_distn.resample([s.stateseq for s in self.states_list])
193         self._clear_caches()
194
195     def resample_init_state_distn(self):
196         self.init_state_distn.resample([s.stateseq[:1] for s in self.states_list])
197         self._clear_caches()
198
199     def resample_states(self,temp=None):
200         for s in self.states_list:
201             s.resample(temp=temp)
202
203     def copy_sample(self):
204         new = copy.copy(self)
205         new.obs_distns = [o.copy_sample() for o in self.obs_distns]
206         new.trans_distn = self.trans_distn.copy_sample()
207         new.init_state_distn = self.init_state_distn.copy_sample()
208         new.states_list = [s.copy_sample(new) for s in self.states_list]
209         return new
```

pyhsmm/model.py>>HMM class 179～209

- *def resample_dur_distns(self)*
  - **持続分布のリサンプリング**
- *def* resample_trans_distn(*self*)
  - **遷移確率のリサンプリング**
- *def* resample_init_states_distns(*self*)
  - **初期状態のリサンプリング**
- def resample_states(self)
  - **状態のリサンプリング**
- def copy_sample(self)
  - **オブジェクトのコピー**

pyhsmm/model.py>>HSMM class 520～526

- **リサンプルの並列処理？**

```
518     ### parallel
519
520     def resample_model_parallel(self,*args,**kwargs):
521         self.resample_dur_distns()
522         super(HSMM,self).resample_model_parallel(*args,**kwargs)
523
524     def _get_parallel_kwargss(self,states_objs):
525         return [dict(trunc=s.trunc,left_censoring=s.left_censoring,
526                      right_censoring=s.right_censoring) for s in states_objs]
```

pyhsmm/model.py>>HSMM class 528～552

```
528     ### EM
529
530     def EM_step(self):
531         super(HSMM,self).EM_step()
532
533         # M step for duration distributions
534         for state, distn in enumerate(self.dur_distns):
535             distn.max_likelihood(
536                     None, # placeholder, "should" be [np.arange(s.T) for s in self.states_list]
537                     [s.expectations[:,state] for s in self.states_list])
538
539     def Viterbi_EM_step(self):
540         super(HSMM,self).Viterbi_EM_step()
541
542         # M step for duration distributions
543         for state, distn in enumerate(self.dur_distns):
544             # TODO TODO get rid of logical indexing
545             distn.max_likelihood(
546                     [s.durations[s.stateseq_norep == state] for s in self.states_list])
547
548     @property
549     def num_parameters(self):
550         return sum(o.num_parameters() for o in self.obs_distns) ¥
551                 + sum(d.num_parameters() for d in self.dur_distns) ¥
552                 + self.state_dim**2 - self.state_dim
```

- HMM classのEM_step()の呼び出し
- M stepの計算
- HMM classのViterbi_EM_step()の呼び出し

EMアルゴリズムで用いる

```
282     def EM_step(self):
283         assert len(self.states_list) > 0, 'Must have data to run EM'
284         self._clear_caches()
285
286         ## E step
287         for s in self.states_list:
288             s.E_step()
289
290         ## M step
291         # observation distribution parameters
292         for state, distn in enumerate(self.obs_distns):
293             distn.max_likelihood([s.data for s in self.states_list],
294                     [s.expectations[:,state] for s in self.states_list])
295
296         # initial distribution parameters
297         self.init_state_distn.max_likelihood(
298                 None, # placeholder, "should" be np.arange(self.state_dim)
299                 [s.expectations[0] for s in self.states_list])
300
301         # transition parameters (requiring more than just the marginal expectations)
302         self.trans_distn.max_likelihood(None,[(s.alphal,s.betal,s.aBl) for s in self.states_list])
```

pyhsmm/model.py>>HMMclass 166～170

```
166         def _clear_caches(self):
167             for s in self.states_list:
168                 s.clear_caches()
169             if hasattr(self.init_state_distn,'clear_caches'):
170                 self.init_state_distn.clear_caches()
```

pyhsmm/model.py>>HMM class 282～302

- cachesの削除
- E stepの計算
- M stepの計算

- 初期分布のパラメータ

- 遷移パラメータ

pyhsmm/model.py>>HMM class 304～333

```
304     def Viterbi_EM_fit(self, tol=0.1, maxiter=20):
305         return self.MAP_EM_fit(tol, maxiter)
306
307     def MAP_EM_step(self):
308         return self.Viterbi_EM_step()
309
310     def Viterbi_EM_step(self):
311         assert len(self.states_list) > 0, 'Must have data to run Viterbi EM'
312         self._clear_caches()
313
314         ## Viterbi step
315         for s in self.states_list:
316             s.Viterbi()
317
318         ## M step
319         # observation distribution parameters
320         for state, distn in enumerate(self.obs_distns):
321             # TODO TODO get rid of logical indexing
322             distn.max_likelihood([s.data[s.stateseq == state] for s in self.states_list])
323
324         # initial distribution parameters
325         self.init_state_distn.max_likelihood(
326                 np.array([s.stateseq[0] for s in self.states_list]))
327
328         # transition parameters (requiring more than just the marginal expectations)
329         self.trans_distn.max_likelihood([s.stateseq for s in self.states_list])
330
331     @property
332     def num_parameters(self):
333         return sum(o.num_parameters() for o in self.obs_distns) + self.state_dim**2
```

- cachesの削除
- Viterbi stepの計算
- M stepの計算

- **初期分布のパラメータ**

- **遷移パラメータ**

Plotで用いる

```
554     ### plotting
555
556     def plot_durations(self,colors=None,states_objs=None):
557         if colors is None:
558             colors = self._get_colors()
559         if states_objs is None:
560             states_objs = self.states_list
561
562         cmap = cm.get_cmap()
563         used_states = self._get_used_states(states_objs)
564         for state,d in enumerate(self.dur_distns):
565             if state in used_states:
566                 d.plot(color=cmap(colors[state]),
567                        data=[s.durations[s.stateseq_norep == state]
568                              for s in states_objs])
569         plt.title('Durations')
570
571     def plot(self,color=None):
572         plt.gcf() #.set_size_inches((10,10))
573         colors = self._get_colors()
574
575         num_subfig_cols = len(self.states_list)
576         for subfig_idx,s in enumerate(self.states_list):
577             plt.subplot(3,num_subfig_cols,1+subfig_idx)
578             self.plot_observations(colors=colors,states_objs=[s])
579
580             plt.subplot(3,num_subfig_cols,1+num_subfig_cols+subfig_idx)
581             s.plot(colors_dict=colors)
582
583             plt.subplot(3,num_subfig_cols,1+2*num_subfig_cols+subfig_idx)
584             self.plot_durations(colors=colors,states_objs=[s])
585
586     def plot_summary(self,color=None):
587         # if there are too many state sequences in states_list, make an
588         # alternative plot that isn't so big
589         raise NotImplementedError # TODO
```

pyhsmm/model.py>>HMM

class 554～589