HW1-MSiA-420

Kungang Zhang

Jan 17, 2017

1 Prob 1:

```
1  public int sampleZ(int doc, int layer, int pos, boolean useWord, boolean sub)
       {
2      MarkovBlanketContainer mbc = getMarkovBlanket(doc, layer, pos, useWord);
3
4      int curZ = _c._z[doc][layer].get(pos);
5
6      //throw new Exception("Implement me.");
7      //Given the sampler-state's counts in the MarkovBlanketContainer,
8      //compute the sampling distribution for the variable at the given pos/
       layer/doc.
9      //product vector into a distribution, and return a sample from it.
10     //Of the arguments to this function, you will only need to use "sub"
       which says
11     //whether or not to subtract the count corresponding to the current
       sampler state,
12     //which is set to the variable curZ.
13     //The number of states at this layer is in the variable _NUMSTATES[layer
       ].
14     //The marginal counts for a given state are _marginal[layer][i]
15     //You should not need to use any other class fields or methods.
16     double[] sample_dist = new double[_NUMSTATES[layer]];
17     double sum=0;//sum must be double, otherwise the sum will be zero during
       running.
18
19     for(int i=0;i<_NUMSTATES[layer];i++){
20       sample_dist[i] = 1;
21       for(int j=0;j<mbc.sbts.length;j++){
22         if (mbc.sbts[j] != null){
23         //Have to catch the case when mbc.sbts[j] is null and pass it
24           if (j == 0){
25             //If j==0, we don't normalize
26             if (sub == true && i == curZ){
27               //If i==curZ, we subtract 1 from both numerator and denomenator
```

```
28                sample_dist[i]*=(mbc.sbts[j].get(i)+_alpha/_NUMSTATES[layer]-1)
    ;
29              }else{
30                sample_dist[i]*=(mbc.sbts[j].get(i)+_alpha/_NUMSTATES[layer]);
31              }
32            }else{
33              //If j!=0, we normalize
34              if (sub == true && i == curZ){
35                sample_dist[i]*=(mbc.sbts[j].get(i)+_alpha/_NUMSTATES[layer]-1)
    /(_marginal[layer][i]+_alpha-1);
36              }else{
37                sample_dist[i]*=(mbc.sbts[j].get(i)+_alpha/_NUMSTATES[layer])/(
    _marginal[layer][i]+_alpha);
38              }
39            }
40          }
41        }
42        sum+=sample_dist[i];
43      }
44
45      for(int i=0;i<_NUMSTATES[layer];i++){
46        sample_dist[i]/=sum;
47      }
48
49      double u = _r.nextDouble();
50      double temp = 0;
51
52      for(int i=0;i<_NUMSTATES[layer];i++){
53        temp+=sample_dist[i];
54        if(u<=temp) return i;
55      }
56      return -1;
57    }
```

Listing 1: Code for sampleZ.

2 Prob 2:

Configuration	Perplexity
1-layer	428.6
4-layer	582.2

Table 1: Perplexity of 1-layer and 4-layer models.

The perplexity of 4-layer model is larger than the 1-layer model (and the log likelihood of 4-layer model is less than 1-layer model), which means the 4-layer model is worse than 1-layer model. However, the 4-layer model with much more flexibility should be better.

3 Prob 3:

```
int _NUMLAYERS = 2; //number of layers of latent states in the sequence model
  int [] _NUMSTATES = new int [] {20, 20}; //number of latent state values at
     each layer
  boolean [][] _transitionTemplate = new boolean[][] {{true, false},{false,
    true}}; //dims: MUST BE _NUMLAYERS x _NUMLAYERS
  boolean [] _productionTemplate = new boolean[] {false, true}; //dims: MUST
    BE _NUMLAYERS
```

Listing 2: Two layers model.

Some states in *layer 1* are often mapped to different states in *layer 0*, for example, $\{3, 18, 0, 18, 0, 5, 14, 0, 15, 8, 6, 0, 8, 6, 2, 11\}$ for *layer 0* and $\{0, 11, 6, 6, 6, 4, 9, 15, 7, 14, 15, 2, 19, 10, 12, 1\}$ for *layer 1*, which means many states in *layer 0* have overlap with each other in generating states in *layer 1*. Then, this means we shouldn't have so many states in *layer 0*.

4 Prob 4:

If the model were a Markov Net instead of a Bayes Net, we have to formulate features and assign a weight to each feature. After going over all the data and all weights (all cliques in the net), we can get the normalize constant Z as a function of weights w_i. Then, we can form a likelihood for the given data set, by calculating the conditional probability table for each clique. Then, maximize that likelihood in terms of those weights (using some optimization method, like gradient descent). After getting weights, we can get any conditional probability table and sample any node given its Markov Blanket. Basically, in Bayes Net, we can obtain the MLE for conditional probability tables on the fly (the MLE can be obtained with local information), while in, Markov Net, the conditional probability tables have to be obtained as a whole before the sampling procedure starts.

5 Prob 5:

The reason that we want to deduct one when $z = curZ$ is that we don't want to over-estimated the rare cases. It would be more trustworthy to relay on the other data and smooth techniques to estimate conditional probability. Then, I tested that if I don't use this technique, whether the result would be significantly different.

- Without the technique of deducting 1 when $z = curZ$:

1000:33626[34150,34717,35295,35909,36446,37008,37572,38138,38658,39240,39795,40369,40947,41513,42067]time:54changes:36757skipWords:0seconddocsamples(layer0):7, 1, 0, 0, 0, 4, 6, 10, 5, 1, 10, 2, 11, 0, 0, 3marginal[0]:[85581.0,148750.0,103600.0,42055.0,79379.0,63082.0,58137.0,24026.0,46710.0,52555.0,129988.0,70873.0,24853.0]

- With the technique:

  1000:33626[34150,34717,35295,35909,36446,37008,37572,38138,38658,39240,39795,40369,40947,41513,42067]time:61changes:38705skipWords:0seconddocsamples(layer0):11, 8, 10, 10, 10, 7, 0, 6, 2, 9, 6, 12, 2, 10, 10, 5marginal[0]:[75014.0,27184.0,114864.0,43269.0,41303.0,42062.0,133380.0,64988.0,39542.0,126362.0,86535.0,37702.0,97384.0]

Even though the most probable paths of states in two cases have different index, but the patterns are very similar. This means, in this case, whether or not to use this technique is not very clear.