持之以恒

Every thing that has a beginning has an end.

Reading Note : Parameter estimation for text analysis 暨LDA学习小结

发表日期：**2013** 年 **3** 月 **5** 日
分类：　**Academics**　标签：　**Gibbs Sampling**, **LDA**, **MCMC**　作者：　恒.　　401 views

伟大的Parameter estimation for text analysis！当把这篇看的差不多的时候，也就到了LDA基础知识终结的时刻了，意味着LDA基础模型的基本了解完成了。所以对该模型的学习告一段落，下一阶段就是了解LDA无穷无尽的变种，不过那些不是很有用了，因为LDA已经被人水遍了各大"论坛"……

总结一下学习过程：
1.概率的基本概念：CDF、PDF、Bayes'rule、各种简单的分布Bernoulli，binomial，multinomial、包括对prior、likelihood、postprior的理解（PRML1.2）
2.共轭：为何Beta Distribution与Bernoulli共轭？ 狄利克雷分布 Dirichlet Distribution
3.概率图模型 Probabilistic Graphical Models: PRML Chapter 8 基本概念即可
4.采样算法：Basic Sampling，Sampling Methods（PRML Chapter 11），马尔科夫蒙特卡洛 MCMC，Gibbs Sampling
5.原始论文阅读记录：【JMLR】LDA
6.进阶资料：《Gibbs Sampling for the Uninitiated》、本文

——————————————— 伟大的分割线！PETA!
————————————————

一、前面无关部分

关于ML、MAP、Bayesian inference

二、模型进一步记忆
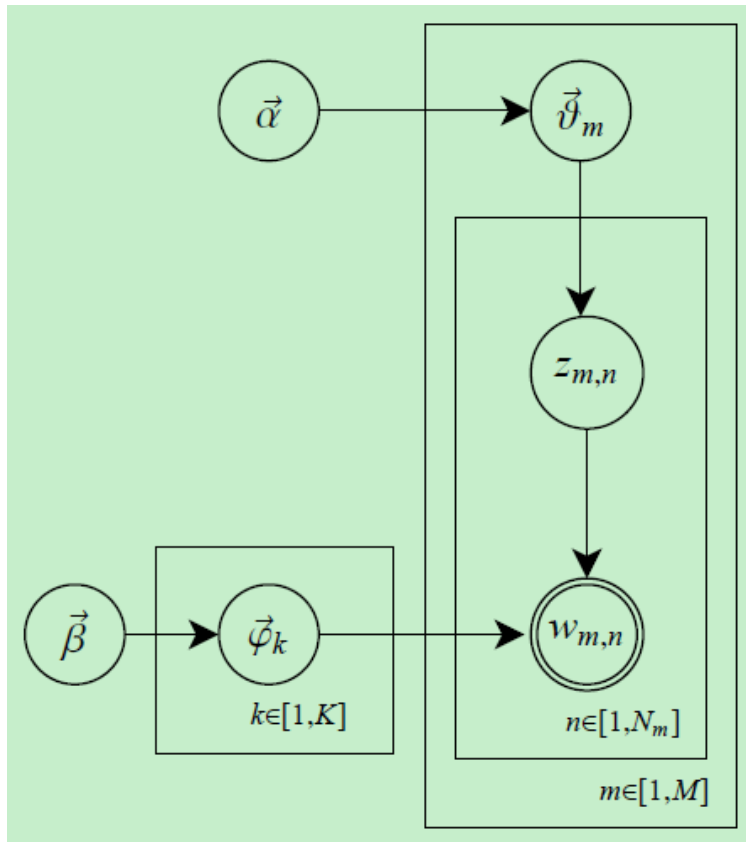
从本图来看，需要记住：

1.θ_m是每一个document单独一个θ，所以M个doc共有M个θ_m，整个θ是一个 M*K的矩阵（M个doc，每个doc一个K维topic分布向量）。

2.φ_k总共只有K个，对于每一个topic，有一个φ_k，这些参数是独立于文档的，也就是对于整个corpus只sample一次。不像θ_m那样每一个都对应一个文档，每个文档都不同，φ_k对于所有文档都相同，是一个K*V的矩阵（K个topic，每个topic一个V维从topic产生词的概率分布）。

就这些了。

三、推导

公式（39）：$P(p|\alpha) = Dir(p|\alpha)$意思是从参数为$\alpha$的狄利克雷分布，采样一个多项分布参数p的概率是多少，概率是标准狄利克雷PDF。这里 Dirichlet delta function为：

$$\Delta(\vec{\alpha}) = \frac{\Gamma(\alpha_1)*\Gamma(\alpha_2)*\ldots*\Gamma(\alpha_k)}{\Gamma(\sum_1^K \alpha_k)}$$

这个function要记住，下面一溜烟全是这个。

公式（43）是一元语言模型的likelihood，意思是如果提供了语料库W，知道了W里面每个词的个数，那么使用最大似然估计最大化L就可以估计出参数多项分布p。

公式（44）是考虑了先验的情形，假如已知语料库W和参数α，那么他们产生多项分布参数p的概率是$Dir(p|\alpha + n)$，这个推导我记得在PRML2.1中有解释，抛开复杂的数学证明，只要参考标准狄利克雷分布的归一化项，很容易想出式（46）的归一化项就是$\Delta(\alpha + n)$。这时如果要通过W估计参数p，那么就要使用贝叶斯推断，用这个狄利克雷pdf输出一个p的期望即可。

最关键的推导（63）-（78）：从63-73的目标是要求出整个LDA的联合概率表达式，这样（63）就可以被用在Gibbs Sampler的分子上。首先（63）把联合概率拆成相互独立的两部分$p(w|z,\beta)$和$p(z|\alpha)$，然后分别对这两部分布求表达式。式（64）、（65）首先不考虑超参数β，而是假设已知参数Φ。这个Φ就是那个K*V维矩阵，表示从每一个topic产生词的概率。然后（66）要把Φ积分掉，这样就可以求出第一部分$p(w|z,\beta)$为表达式（68）。从66-68的积分过程一直在套用狄利克雷积分的结果，反正整篇文章套来套去始终就是这么一个狄利克雷积分。$\vec{n}_z$是一个V维的向量，对于topic z，代表每一个词在这个topic里面有几个。从69到72的道理其实和64-68一模一样了。$\vec{n}_m$是一个K维向量，对于文档m，代表每一个topic在这个文档里有几个词。

最后（78）求出了Gibbs Sampler所需要的条件概率表达式。这个表达式还是要贴出来的，为了和代码里面对应：

具体选择下一个新topic的方法是：通过计算每一个topic的新的产生概率 $p(z_i = k | z_{\neg i}, w)$ 也就是代码中的p[k]产生一个新topic。比如有三个topic，算出来产生新的p的概率值为{0.3,0.2,0.4}，注意这个条件概率加起来并不一定是一。然后我为了按照这个概率产生一个新topic，我用random函数从 uniform distribution产生一个0至0.9的随机数r。如果0<=r<0.3，则新topic赋值为1，如果0.3<=r<0.5，则新topic赋值为2，如果0.5<=r<0.9，那么新topic赋值为3。

四、代码

view plain　copy to clipboard　print　?

```
01.  /*
02.   * (C) Copyright 2005, Gregor Heinrich (gregor :: arbylon : net)
03.   * LdaGibbsSampler is free software; you can redistribute it and/or modify it
04.   * under the terms of the GNU General Public License as published by the Free
05.   * Software Foundation; either version 2 of the License, or (at your option) a
      ny
06.   * later version.
07.   * LdaGibbsSampler is distributed in the hope that it will be useful, but
08.   * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
      or
09.   * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for mo
      re
10.   * details.
11.   * You should have received a copy of the GNU General Public License along wit
      h
12.   * this program; if not, write to the Free Software Foundation, Inc., 59 Templ
      e
13.   * Place, Suite 330, Boston, MA 02111-1307 USA
14.   */
15.  import java.text.DecimalFormat;
16.  import java.text.NumberFormat;
17.
18.  public class LdaGibbsSampler {
19.      /**
20.       * document data (term lists)
21.       */
22.      int[][] documents;
23.      /**
24.       * vocabulary size
25.       */
26.      int V;
27.      /**
28.       * number of topics
29.       */
30.      int K;
31.      /**
32.       * Dirichlet parameter (document--topic associations)
33.       */
34.      double alpha;
35.      /**
36.       * Dirichlet parameter (topic--term associations)
37.       */
38.      double beta;
39.      /**
40.       * topic assignments for each word.
41.       * N * M 维，第一维是文档，第二维是word
42.       */
43.      int z[][];
```

```
44.      /**
45.       * nw[i][j] number of instances of word i (term?) assigned to topic j.
46.       */
47.      int[][] nw;
48.      /**
49.       * nd[i][j] number of words in document i assigned to topic j.
50.       */
51.      int[][] nd;
52.      /**
53.       * nwsum[j] total number of words assigned to topic j.
54.       */
55.      int[] nwsum;
56.      /**
57.       * nasum[i] total number of words in document i.
58.       */
59.      int[] ndsum;
60.      /**
61.       * cumulative statistics of theta
62.       */
63.      double[][] thetasum;
64.      /**
65.       * cumulative statistics of phi
66.       */
67.      double[][] phisum;
68.      /**
69.       * size of statistics
70.       */
71.      int numstats;
72.      /**
73.       * sampling lag (?)
74.       */
75.      private static int THIN_INTERVAL = 20;
76.
77.      /**
78.       * burn-in period
79.       */
80.      private static int BURN_IN = 100;
81.
82.      /**
83.       * max iterations
84.       */
85.      private static int ITERATIONS = 1000;
86.
87.      /**
88.       * sample lag (if -1 only one sample taken)
89.       */
90.      private static int SAMPLE_LAG;
91.
92.      private static int dispcol = 0;
93.
94.      /**
95.       * Initialise the Gibbs sampler with data.
96.       *
97.       * @param V
98.       *            vocabulary size
99.       * @param data
100.      */
101.     public LdaGibbsSampler(int[][] documents, int V) {
102.
103.         this.documents = documents;
104.         this.V = V;
105.     }
106.
107.     /**
108.      * Initialisation: Must start with an assignment of observations to topics ?
109.      * Many alternatives are possible, I chose to perform random assignments
110.      * with equal probabilities
111.      *
112.      * @param K
```

```java
113.        *              number of topics
114.        * @return z assignment of topics to words
115.        */
116.   public void initialState(int K) {
117.       int i;
118.
119.       int M = documents.length;
120.
121.       // initialise count variables.
122.       nw = new int[V][K];
123.       nd = new int[M][K];
124.       nwsum = new int[K];
125.       ndsum = new int[M];
126.
127.       // The z_i are are initialised to values in [1,K] to determine the
128.       // initial state of the Markov chain.
129.       // 为了方便，他没用从狄利克雷参数采样，而是随机初始化了！
130.
131.       z = new int[M][];
132.       for (int m = 0; m < M; m++) {
133.           int N = documents[m].length;
134.           z[m] = new int[N];
135.           for (int n = 0; n < N; n++) {
136.               //随机初始化！
137.               int topic = (int) (Math.random() * K);
138.               z[m][n] = topic;
139.               // number of instances of word i assigned to topic j
140.               // documents[m][n] 是第m个doc中的第n个词
141.               nw[documents[m][n]][topic]++;
142.               // number of words in document i assigned to topic j.
143.               nd[m][topic]++;
144.               // total number of words assigned to topic j.
145.               nwsum[topic]++;
146.           }
147.           // total number of words in document i
148.           ndsum[m] = N;
149.       }
150.   }
151.
152.   /**
153.    * Main method: Select initial state ? Repeat a large number of times: 1.
154.    * Select an element 2. Update conditional on other elements. If
155.    * appropriate, output summary for each run.
156.    *
157.    * @param K
158.    *             number of topics
159.    * @param alpha
160.    *             symmetric prior parameter on document--topic associations
161.    * @param beta
162.    *             symmetric prior parameter on topic--term associations
163.    */
164.   private void gibbs(int K, double alpha, double beta) {
165.       this.K = K;
166.       this.alpha = alpha;
167.       this.beta = beta;
168.
169.       // init sampler statistics
170.       if (SAMPLE_LAG > 0) {
171.           thetasum = new double[documents.length][K];
172.           phisum = new double[K][V];
173.           numstats = 0;
174.       }
175.
176.       // initial state of the Markov chain:
177.       //启动马尔科夫链需要一个起始状态
178.       initialState(K);
179.
180.       //每一轮sample
181.       for (int i = 0; i < ITERATIONS; i++) {
182.
```

```
183.              // for all z_i
184.              for (int m = 0; m < z.length; m++) {
185.                  for (int n = 0; n < z[m].length; n++) {
186.
187.                      // (z_i = z[m][n])
188.                      // sample from p(z_i|z_-i, w)
189.                      //核心步骤，通过论文中表达式（78）为文档m中的第n个词采样新的
      topic
190.                      int topic = sampleFullConditional(m, n);
191.                      z[m][n] = topic;
192.                  }
193.              }
194.
195.              // get statistics after burn-in
196.              //如果当前迭代轮数已经超过 burn-in的限制，并且正好达到 sample lag间隔
197.              //则当前的这个状态是要计入总的输出参数的，否则的话忽略当前状态，继续
      sample
198.              if ((i > BURN_IN) && (SAMPLE_LAG > 0) && (i % SAMPLE_LAG == 0)) {
199.                  updateParams();
200.              }
201.          }
202.      }
203.
204.      /**
205.       * Sample a topic z_i from the full conditional distribution: p(z_i = j |
206.       * z_-i, w) = (n_-i,j(w_i) + beta)/(n_-i,j(.) + W * beta) * (n_-
      i,j(d_i) +
207.       * alpha)/(n_-i,.(d_i) + K * alpha)
208.       *
209.       * @param m
210.       *            document
211.       * @param n
212.       *            word
213.       */
214.      private int sampleFullConditional(int m, int n) {
215.
216.          // remove z_i from the count variables
217.          //这里首先要把原先的topic z(m,n)从当前状态中移除
218.          int topic = z[m][n];
219.          nw[documents[m][n]][topic]--;
220.          nd[m][topic]--;
221.          nwsum[topic]--;
222.          ndsum[m]--;
223.
224.          // do multinomial sampling via cumulative method:
225.          double[] p = new double[K];
226.          for (int k = 0; k < K; k++) {
227.              //nw 是第i个word被赋予第j个topic的个数
228.              //在下式中，documents[m][n]是word id，k为第k个topic
229.              //nd 为第m个文档中被赋予topic k的词的个数
230.              p[k] = (nw[documents[m][n]][k] + beta) / (nwsum[k] + V * beta)
231.                      * (nd[m][k] + alpha) / (ndsum[m] + K * alpha);
232.          }
233.          // cumulate multinomial parameters
234.          for (int k = 1; k < p.length; k++) {
235.              p[k] += p[k - 1];
236.          }
237.          // scaled sample because of unnormalised p[]
238.          double u = Math.random() * p[K - 1];
239.          for (topic = 0; topic < p.length; topic++) {
240.              if (u < p[topic])
241.                  break;
242.          }
243.
244.          // add newly estimated z_i to count variables
245.          nw[documents[m][n]][topic]++;
246.          nd[m][topic]++;
247.          nwsum[topic]++;
248.          ndsum[m]++;
```

```
249.
250.            return topic;
251.        }
252.
253.        /**
254.         * Add to the statistics the values of theta and phi for the current state
        .
255.         */
256.        private void updateParams() {
257.            for (int m = 0; m < documents.length; m++) {
258.                for (int k = 0; k < K; k++) {
259.                    thetasum[m][k] += (nd[m]
    [k] + alpha) / (ndsum[m] + K * alpha);
260.                }
261.            }
262.            for (int k = 0; k < K; k++) {
263.                for (int w = 0; w < V; w++) {
264.                    phisum[k][w] += (nw[w][k] + beta) / (nwsum[k] + V * beta);
265.                }
266.            }
267.            numstats++;
268.        }
269.
270.        /**
271.         * Retrieve estimated document--
    topic associations. If sample lag > 0 then
272.         * the mean value of all sampled statistics for theta[][] is taken.
273.         *
274.         * @return theta multinomial mixture of document topics (M x K)
275.         */
276.        public double[][] getTheta() {
277.            double[][] theta = new double[documents.length][K];
278.
279.            if (SAMPLE_LAG > 0) {
280.                for (int m = 0; m < documents.length; m++) {
281.                    for (int k = 0; k < K; k++) {
282.                        theta[m][k] = thetasum[m][k] / numstats;
283.                    }
284.                }
285.
286.            } else {
287.                for (int m = 0; m < documents.length; m++) {
288.                    for (int k = 0; k < K; k++) {
289.                        theta[m][k] = (nd[m]
    [k] + alpha) / (ndsum[m] + K * alpha);
290.                    }
291.                }
292.            }
293.
294.            return theta;
295.        }
296.
297.        /**
298.         * Retrieve estimated topic--
    word associations. If sample lag > 0 then the
299.         * mean value of all sampled statistics for phi[][] is taken.
300.         *
301.         * @return phi multinomial mixture of topic words (K x V)
302.         */
303.        public double[][] getPhi() {
304.            double[][] phi = new double[K][V];
305.            if (SAMPLE_LAG > 0) {
306.                for (int k = 0; k < K; k++) {
307.                    for (int w = 0; w < V; w++) {
308.                        phi[k][w] = phisum[k][w] / numstats;
309.                    }
310.                }
311.            } else {
312.                for (int k = 0; k < K; k++) {
313.                    for (int w = 0; w < V; w++) {
```

```java
314.                    phi[k][w] = (nw[w][k] + beta) / (nwsum[k] + V * beta);
315.                }
316.            }
317.        }
318.        return phi;
319.    }
320.
321.    /**
322.     * Configure the gibbs sampler
323.     *
324.     * @param iterations
325.     *            number of total iterations
326.     * @param burnIn
327.     *            number of burn-in iterations
328.     * @param thinInterval
329.     *            update statistics interval
330.     * @param sampleLag
331.     *            sample interval (-1 for just one sample at the end)
332.     */
333.    public void configure(int iterations, int burnIn, int thinInterval,
334.        int sampleLag) {
335.        ITERATIONS = iterations;
336.        BURN_IN = burnIn;
337.        THIN_INTERVAL = thinInterval;
338.        SAMPLE_LAG = sampleLag;
339.    }
340.
341.    /**
342.     * Driver with example data.
343.     *
344.     * @param args
345.     */
346.    public static void main(String[] args) {
347.        // words in documents
348.        int[]
[] documents = { {1, 4, 3, 2, 3, 1, 4, 3, 2, 3, 1, 4, 3, 2, 3, 6},
349.            {2, 2, 4, 2, 4, 2, 2, 2, 2, 4, 2, 2},
350.            {1, 6, 5, 6, 0, 1, 6, 5, 6, 0, 1, 6, 5, 6, 0, 0},
351.            {5, 6, 6, 2, 3, 3, 6, 5, 6, 2, 2, 6, 5, 6, 6, 6, 0},
352.            {2, 2, 4, 4, 4, 4, 1, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 0},
353.            {5, 4, 2, 3, 4, 5, 6, 6, 5, 4, 3, 2}};
354.        // vocabulary
355.        int V = 7;
356.        int M = documents.length;
357.        // # topics
358.        int K = 2;
359.        // good values alpha = 2, beta = .5
360.        double alpha = 2;
361.        double beta = .5;
362.
363.        LdaGibbsSampler lda = new LdaGibbsSampler(documents, V);
364.
365.        //设定sample参数，采样运行10000轮，burn-in 2000轮，第三个参数没用，是为了显
示
366.        //第四个参数是sample lag，这个很重要，因为马尔科夫链前后状态
conditional dependent，所以要跳过几个采样
367.        lda.configure(10000, 2000, 100, 10);
368.
369.        //跑一个！走起！
370.        lda.gibbs(K, alpha, beta);
371.
372.        //输出模型参数，论文中式 （81）与（82）
373.        double[][] theta = lda.getTheta();
374.        double[][] phi = lda.getPhi();
375.    }
376. }
```

分类： **Academics**   标签：   **Gibbs Sampling**, **LDA**, **MCMC**   作者： 恒.   401 views