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

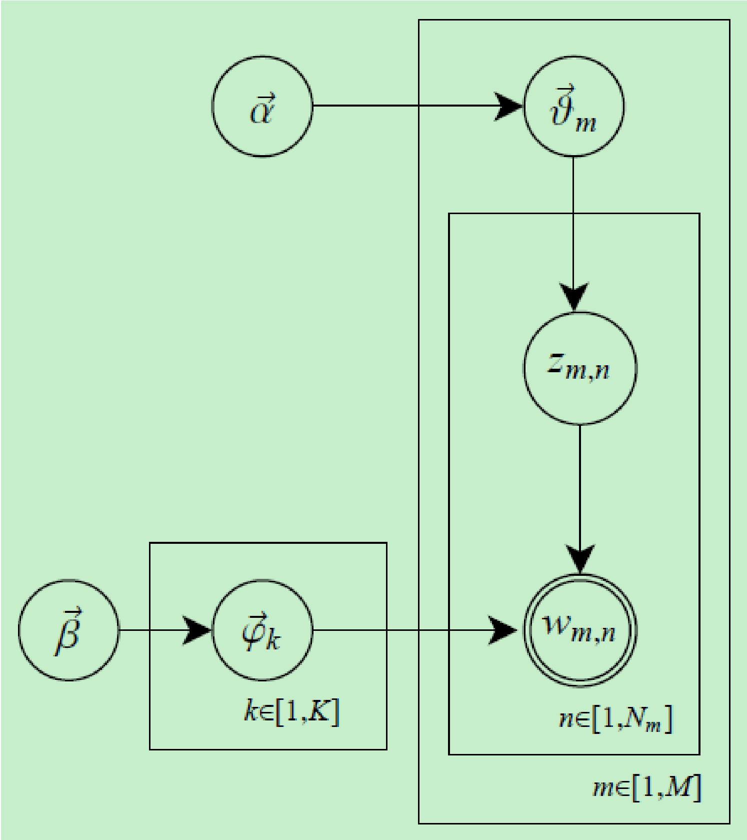
——————————————— 伟大的分割线 ！ PETA!

———————————————

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：

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

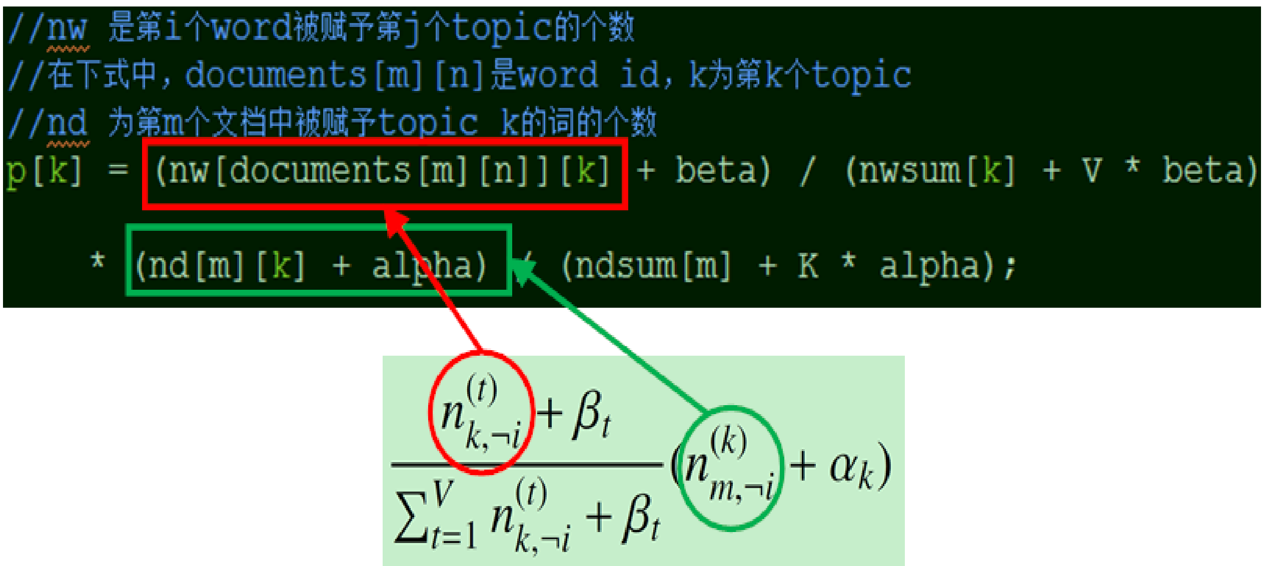
这个function要记住，下面一溜烟全是这个。

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 ?

```java
/*
 * (C) Copyright 2005, Gregor Heinrich (gregor :: arbylon : net)
 * LdaGibbsSampler is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option) any
 * later version.
 * LdaGibbsSampler is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
 * details.
 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 59 Temple
 * Place, Suite 330, Boston, MA 02111-1307 USA
 */
import java.text.DecimalFormat;
import java.text.NumberFormat;

public class LdaGibbsSampler {
    /**
     * document data (term lists)
     */
    int[][] documents;
    /**
     * vocabulary size
     */
    int V;
    /**
     * number of topics
     */
    int K;
    /**
     * Dirichlet parameter (document--topic associations)
     */
    double alpha;
    /**
     * Dirichlet parameter (topic--term associations)
     */
    double beta;
    /**
     * topic assignments for each word.
     * N * M 维，第一维是文档，第二维是word
     */
    int z[][];
```

```
    /**
     * nw[i][j] number of instances of word i (term?) assigned to topic j.
     */
    int[][] nw;
    /**
     * nd[i][j] number of words in document i assigned to topic j.
     */
    int[][] nd;
    /**
     * nwsum[j] total number of words assigned to topic j.
     */
    int[] nwsum;
    /**
     * nasum[i] total number of words in document i.
     */
    int[] ndsum;
    /**
     * cumulative statistics of theta
     */
    double[][] thetasum;
    /**
     * cumulative statistics of phi
     */
    double[][] phisum;
    /**
     * size of statistics
     */
    int numstats;
    /**
     * sampling lag (?)
     */
    private static int THIN_INTERVAL = 20;

    /**
     * burn-in period
     */
    private static int BURN_IN = 100;

    /**
     * max iterations
     */
    private static int ITERATIONS = 1000;

    /**
     * sample lag (if -1 only one sample taken)
     */
    private static int SAMPLE_LAG;

    private static int dispcol = 0;

    /**
     * Initialise the Gibbs sampler with data.
     *
     * @param V
     *            vocabulary size
     * @param data
     */
    public LdaGibbsSampler(int[][] documents, int V) {

        this.documents = documents;
        this.V = V;
    }

    /**
     * Initialisation: Must start with an assignment of observations to topics
 ?
     * Many alternatives are possible, I chose to perform random assignments
     * with equal probabilities
     *
     * @param K
```

```
     *             number of topics
     * @return z assignment of topics to words
     */
    public void initialState(int K) {
        int i;

        int M = documents.length;

        // initialise count variables.
        nw = new int[V][K];
        nd = new int[M][K];
        nwsum = new int[K];
        ndsum = new int[M];

        // The z_i are are initialised to values in [1,K] to determine the
        // initial state of the Markov chain.
        // 为了方便，他没用从狄利克雷参数采样，而是随机初始化了！

        z = new int[M][];
        for (int m = 0; m < M; m++) {
            int N = documents[m].length;
            z[m] = new int[N];
            for (int n = 0; n < N; n++) {
                //随机初始化！
                int topic = (int) (Math.random() * K);
                z[m][n] = topic;
                // number of instances of word i assigned to topic j
                // documents[m][n] 是第m个doc中的第n个词
                nw[documents[m][n]][topic]++;
                // number of words in document i assigned to topic j.
                nd[m][topic]++;
                // total number of words assigned to topic j.
                nwsum[topic]++;
            }
            // total number of words in document i
            ndsum[m] = N;
        }
    }

    /**
     * Main method: Select initial state ? Repeat a large number of times: 1.
     * Select an element 2. Update conditional on other elements. If
     * appropriate, output summary for each run.
     *
     * @param K
     *             number of topics
     * @param alpha
     *             symmetric prior parameter on document--topic associations
     * @param beta
     *             symmetric prior parameter on topic--term associations
     */
    private void gibbs(int K, double alpha, double beta) {
        this.K = K;
        this.alpha = alpha;
        this.beta = beta;

        // init sampler statistics
        if (SAMPLE_LAG > 0) {
            thetasum = new double[documents.length][K];
            phisum = new double[K][V];
            numstats = 0;
        }

        // initial state of the Markov chain:
        //启动马尔科夫链需要一个起始状态
        initialState(K);

        //每一轮sample
        for (int i = 0; i < ITERATIONS; i++) {

```

```java
            // for all z_i
            for (int m = 0; m < z.length; m++) {
                for (int n = 0; n < z[m].length; n++) {

                    // (z_i = z[m][n])
                    // sample from p(z_i|z_-i, w)
                    //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
                    int topic = sampleFullConditional(m, n);
                    z[m][n] = topic;
                }
            }

            // get statistics after burn-in
            //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
            //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
            if ((i > BURN_IN) && (SAMPLE_LAG > 0) && (i % SAMPLE_LAG == 0)) {

                updateParams();
            }
        }
    }

    /**
     * Sample a topic z_i from the full conditional distribution: p(z_i = j |
     * z_-i, w) = (n_-i,j(w_i) + beta)/(n_-i,j(.) + W * beta) * (n_-i,j(d_i) +
     * alpha)/(n_-i,.(d_i) + K * alpha)
     *
     * @param m
     *            document
     * @param n
     *            word
     */
    private int sampleFullConditional(int m, int n) {

        // remove z_i from the count variables
        //这里首先要把原先的topic z(m,n)从当前状态中移除
        int topic = z[m][n];
        nw[documents[m][n]][topic]--;
        nd[m][topic]--;
        nwsum[topic]--;
        ndsum[m]--;

        // do multinomial sampling via cumulative method:
        double[] p = new double[K];
        for (int k = 0; k < K; k++) {
            //nw 是第i个word被赋予第j个topic的个数
            //在下式中，documents[m][n]是word id，k为第k个topic
            //nd 为第m个文档中被赋予topic k的词的个数
            p[k] = (nw[documents[m][n]][k] + beta) / (nwsum[k] + V * beta)
                * (nd[m][k] + alpha) / (ndsum[m] + K * alpha);
        }
        // cumulate multinomial parameters
        for (int k = 1; k < p.length; k++) {
            p[k] += p[k - 1];
        }
        // scaled sample because of unnormalised p[]
        double u = Math.random() * p[K - 1];
        for (topic = 0; topic < p.length; topic++) {
            if (u < p[topic])
                break;
        }

        // add newly estimated z_i to count variables
        nw[documents[m][n]][topic]++;
        nd[m][topic]++;
        nwsum[topic]++;
        ndsum[m]++;
```

```java

        return topic;
    }

    /**
     * Add to the statistics the values of theta and phi for the current state.
     */
    private void updateParams() {
        for (int m = 0; m < documents.length; m++) {
            for (int k = 0; k < K; k++) {
                thetasum[m][k] += (nd[m][k] + alpha) / (ndsum[m] + K * alpha);
            }
        }
        for (int k = 0; k < K; k++) {
            for (int w = 0; w < V; w++) {
                phisum[k][w] += (nw[w][k] + beta) / (nwsum[k] + V * beta);
            }
        }
        numstats++;
    }

    /**
     * Retrieve estimated document--topic associations. If sample lag > 0 then
     * the mean value of all sampled statistics for theta[][] is taken.
     *
     * @return theta multinomial mixture of document topics (M x K)
     */
    public double[][] getTheta() {
        double[][] theta = new double[documents.length][K];

        if (SAMPLE_LAG > 0) {
            for (int m = 0; m < documents.length; m++) {
                for (int k = 0; k < K; k++) {
                    theta[m][k] = thetasum[m][k] / numstats;
                }
            }

        } else {
            for (int m = 0; m < documents.length; m++) {
                for (int k = 0; k < K; k++) {
                    theta[m][k] = (nd[m][k] + alpha) / (ndsum[m] + K * alpha);
                }
            }
        }

        return theta;
    }

    /**
     * Retrieve estimated topic--word associations. If sample lag > 0 then the
     * mean value of all sampled statistics for phi[][] is taken.
     *
     * @return phi multinomial mixture of topic words (K x V)
     */
    public double[][] getPhi() {
        double[][] phi = new double[K][V];
        if (SAMPLE_LAG > 0) {
            for (int k = 0; k < K; k++) {
                for (int w = 0; w < V; w++) {
                    phi[k][w] = phisum[k][w] / numstats;
                }
            }
        } else {
            for (int k = 0; k < K; k++) {
                for (int w = 0; w < V; w++) {
```

```
                    phi[k][w] = (nw[w][k] + beta) / (nwsum[k] + V * beta);
                }
            }
        }
        return phi;
    }

    /**
     * Configure the gibbs sampler
     *
     * @param iterations
     *            number of total iterations
     * @param burnIn
     *            number of burn-in iterations
     * @param thinInterval
     *            update statistics interval
     * @param sampleLag
     *            sample interval (-1 for just one sample at the end)
     */
    public void configure(int iterations, int burnIn, int thinInterval,
        int sampleLag) {
        ITERATIONS = iterations;
        BURN_IN = burnIn;
        THIN_INTERVAL = thinInterval;
        SAMPLE_LAG = sampleLag;
    }

    /**
     * Driver with example data.
     *
     * @param args
     */
    public static void main(String[] args) {
        // words in documents
        int[][] documents = { {1, 4, 3, 2, 3, 1, 4, 3, 2, 3, 1, 4, 3, 2, 3, 6},
            {2, 2, 4, 2, 4, 2, 2, 2, 2, 4, 2, 2},
            {1, 6, 5, 6, 0, 1, 6, 5, 6, 0, 1, 6, 5, 6, 0, 0},
            {5, 6, 6, 2, 3, 3, 6, 5, 6, 2, 2, 6, 5, 6, 6, 6, 0},
            {2, 2, 4, 4, 4, 4, 1, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 0},
            {5, 4, 2, 3, 4, 5, 6, 6, 5, 4, 3, 2}};
        // vocabulary
        int V = 7;
        int M = documents.length;
        // # topics
        int K = 2;
        // good values alpha = 2, beta = .5
        double alpha = 2;
        double beta = .5;

        LdaGibbsSampler lda = new LdaGibbsSampler(documents, V);

        //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
        //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
        lda.configure(10000, 2000, 100, 10);

        //跑一个！走起！
        lda.gibbs(K, alpha, beta);

        //输出模型参数，论文中式 （81）与（82）
        double[][] theta = lda.getTheta();
        double[][] phi = lda.getPhi();
    }
}
```

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