Stat 547 — Assignment 2

Release Date:	Wednesday March 31, 2011
Due Date:	Wednesday, April 14 (possible extension TBA)

You should submit a written report under my office door (LSK 330) as well as a zipped file by email containing the "answer" and "src" folders (do *not* include the other directory to avoid email quotas problems). The report should contain your work for the written questions as well as a summary of what worked/did not work in your experiments.

1 Getting the code and data

First, make sure as early as possible that you can access the course materials.

`http://www.stat.ubc.ca/~bouchard/pri/stat547-assignment2.zip`

The authentication restrictions are due to licensing terms. The username and password have been announced in class (same as assignment 1), but if for any reason you did not get it, please let me know by email.
Unzip the downloaded file to your local working directory. It contains both the data that you will need, some evaluation code, and some harness code that will help you do the assignment.

2 Technical stuff

Use the same procedure as assignment 1 to get the code harness setup. Create a new, fresh project. You will probably not need the code from the first assignment in this assignment, but you can always copy contents from the previous assignments manually later on (if you copy files into a project, you might need to refresh it by right-clicking and selecting refresh on the project).

3 Dirichlet process (DP) mixtures

In this question, after going over some written questions about properties of Dirichlet distributions and Dirichlet processes, we will use the Dirichlet process mixture model to do density estimation.

3.1 Theory on DPs

We start by looking at some basic properties of Dirichlet processes:

Part A

Suppose $G \sim \mathrm{DP}(\alpha_0, G_0)$, and let $A, B \in \mathcal{F}_\Omega$ be disjoint measurable sets. Evaluate $\mathbb{E}[G(A)G(B)]$.

Part B

Recall that two simple lemmas on Dirichlet distributions were a key ingredient in the proof of equivalence of the stick breaking and Kolmogorov consistency definitions of DPs. Refer to lemma 2.11 and 2.12 in the lecture notes, part 2. Prove both results. Note that you should not have to use multivariate changes of variable. Hints: For 2.11, all you need is to represent the Dirichlet distributions as gamma random variables. For 2.12, use dirichlet-multinomial conjugacy and the law of total probabilities.

3.2 DP mixture models in practice

In this section, we are going to implement a Dirichlet process mixture model with a normal likelihood function. You will need R 2.2.12 installed on your machine (probably other versions work as well, but I haven't tested). Java will call R for you to create plots. Important note for windows user: the path to R should not have spaces in it. This means that if your current R distribution is in 'Program File', the simplest will be to install another R in a path without spaces (you can always erase it after doing the assignment).
Similarly, your 'workspace' folder (the one in which the project sits) should be in a path that does not contain spaces (e.g. 'My Documents' will cause problems). After installing R, set the variable 'pathToR' in the file 'src/dp/DPMixtureTest' to the executable you would call from the command line to launch R (be careful about case sensitiveness with linux/mac, e.g. r vs. R). In mac or linux, type 'which r' or 'which R' in the command line to find out (often, it will be '/usr/bin/r'); on windows, this is the file 'R.exe' usually in a subfolder bin of the R folder.

Part A

Run DPMixtureTest, which is in the directory 'src/dp/'. It generates data using a synthetic mixture distribution; if you are curious, you can look at the function that generates each data point in 'sampleSyntheticData()', but it's probably simpler to visualize in the output of the program. Each time DPMixtureTest is started, it create a new execution subfolder in 'state/execs'. When you are happy with the answer of one question, copy the x.exec folder over, renaming it into a section.subsection.part format (e.g. 3.2.A for this question). Inside the execution subfolder, open 'true-clusters.pdf'. This is the points that we

will approximate a density from. After a few seconds, more files will appear in the execution subdirectory. The files 'cluster-<iteration#>.pdf' show the current clustering of the points, and the files 'pred-<iteration#>.pdf' show the predictive density corresponding to the current sample. The results are pretty bad so far, simply because there are some parts of the code that need to completed.

The first part that needs to be completed is the core of the collapsed Gibbs sampler. Open 'DPState.java', still in the folder 'src/dp'. This class is responsible for keeping track of the current state of the sampler (one cluster indicator for each datapoint/customer), and to resample customer's tables, one at the time. Look at the function 'doSamplePoint()', which I have written for you. First, 'doSamplePoint()' takes out one customer from its table (since the sufficient statistics considered here, $\sum x_i, \sum x_i x_i^T$, are linear, this is done by subtracting the sufficient statistic of the customer from the sum of the sufficient statistic for the table). Second, 'doSamplePoint()' computes an unnormalized conditional probability for each possible operations (inserting at one existing table, or creating a new table). Finally, 'doSamplePoint()' normalizes the multinomial distribution, sample from it, and apply the selected operation to the datastructures keeping track of cluster allocations.

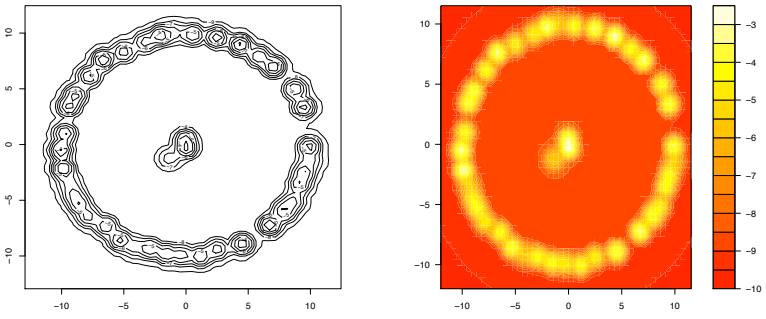
Your task will be to implement the second step, which is done by calling 'logUnnormalizedPrJoinTable()' and 'logUnnormalizedPrCreateTable()', which respectively compute the unnormalized log probability of joining a table and creating a table. You will find the following functions useful:

$$\text{marginalLogLikelihood}(y) = \int l(y|\theta) G_0(\,\mathrm{d}\theta)$$

$$\text{predictiveLogLikelihood}(y_{\text{pred}}|y_{\text{cond}}) = \int l(y_{\text{pred}}|\theta) G_0(\,\mathrm{d}\theta|y_{\text{cond}}),$$

where l is the likelihood model, and G_0 is the base measure. Again, note that points are represented by their sufficient statistics, which by definition is all we need to perform these computations.

Once you have filled these functions, you should get a predictive density in 'pred-<iteration#>.pdf' that looks like this:



This is better, but still not quite what we would like: note that a very large number of clusters is needed to get this fit. We will address this problem in the next question.

Part B

The high number of clusters in the first part is caused by a restriction in the current code for the likelihood model. Under this restriction, θ only contains a prior for the mean of normal distributions, and the variance if fixed:

$$\begin{aligned} \mu_d &\sim \mathrm{N}(0,1) \quad d \in \{0,1\} \\ \theta &= (\mu_1, \mu_2) \\ y|\theta &\sim \mathrm{N}(\theta, I), \end{aligned}$$

where I is the identity matrix.
We will use a more flexible likelihood model in this section, based on Normal-Inverse-Wishart distributions (NIW) to address the problem with the large number of clusters. NIW distributions are priors distribution on both mean vectors and covariance matrices, and are conjugate with a multivariate normal likelihood models:

$$\begin{aligned} \theta = (\mu, \Sigma) &\sim \mathrm{NIW} \\ y|\mu, \Sigma &\sim \mathrm{N}(\mu, \Sigma). \end{aligned}$$

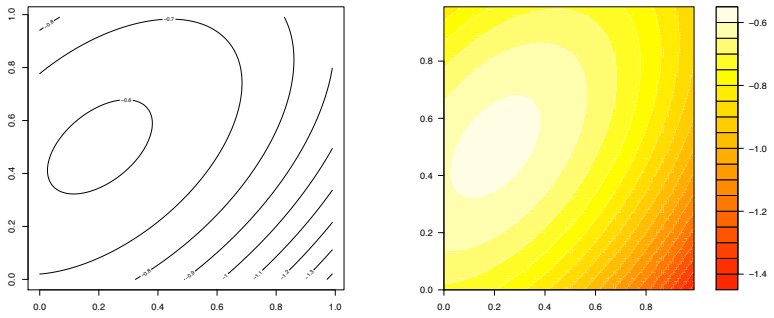
See the following references for more background information on Normal-Inverse-Wishart: Erik Sudderth's thesis, section 2.1.4:

`http://www.cs.brown.edu/~sudderth/papers/sudderthPhD.pdf`

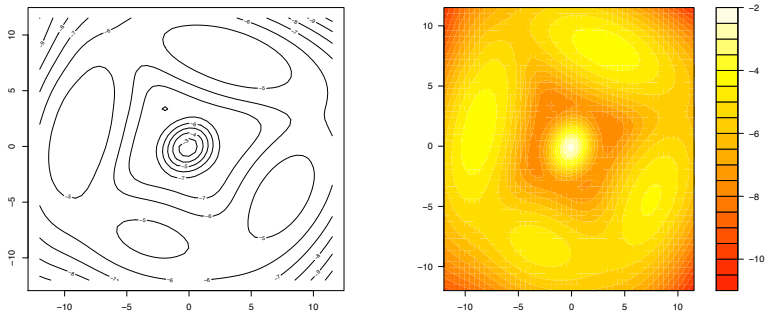
and this tech report, written by Kevin Murphy, page 19:

`http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.4603`

Your task will be to compute the marginal likelihood of (sufficient statistics of) observations, and is greatly simplified by the fact that I have written the code that computes the posterior of a Normal-Inverse-Wishart given (sufficient statistics of) observations, as well as evaluation of normal and normal-inverse-wishart densities. If you start writing complicated matrix expression, think a little bit more about a trick we covered in class! Open 'normalmodel/MargMultGaussian.java', which is responsible of computing marginals in a multivariate gaussian model. See 'margLogLikelihood()' for hints and the location where you should write this piece of code.
Once you are done with implementing this, run 'dp/TestDistrib.java'. In the execution folder, the file 'predictive-density.pdf' should look like



The final step is to re-run the DP mixture model with this likelihood model. This is done by setting the variable 'useNormalInverseWishartModel' in 'DP-MixtureTest.java' to true. You should get something like this now:



Part C (Optional)

Apply the code to a real dataset of your choice. Contact me for help on how to load the data into the DP mixture model. Some standard sources of datasets:

`http://archive.ics.uci.edu/ml/`
`http://mlcomp.org/`

4 Beyond DPs

In this question, we will look at models beyond simple Dirichlet processes. To begin with, we will revisit some of the exchangeability claims and prove them formally:

4.1 Theory

Part A

Prove directly (that is without invoking the existence of the stick breaking prior) that the Pitman-Yor (PY) version of the chinese restaurant process is exchangeable. Hint: A useful fact from group theory: any *permutation* (bijection between $\{1, \ldots, N\}$ to itself) can be written as the composition of *transpositions* (permutation that leave all the elements fixed except for two elements).

Part B

Prove that the Indian Buffet Process is exchangeable as well.

4.2 Practice

In this question, we will apply a hierarchical DP to a language modeling task. Start by running 'LanguageModelEvaluation.java' in the package hpd. It loads training and test sentences (the same as previous assignment, without using the POS information this time), trains language models on the training section, and evaluates them on the test sentences. Evaluation is done using the perplexity metric:

`http://en.wikipedia.org/wiki/Perplexity`

Also check in the execution folder, each time the code is ran, text is generated at random from the model into files called '*.generated', which can sometimes lead to funny sentences.

Part A

Open the file 'hdp/DP.java', this is where the sampler takes out customers and reinsert them at random. Note that following work by Blunsom et al, we use a sampler that maintains only an histogram over table sizes for each word (this is a sufficient statistic for the sampler at hand):

`http://www.aclweb.org/anthology/P/P09/P09-2085.pdf`

The book-keeping of this histogram is done for you in 'Restaurant.java'. Your task will be to do part of the Gibbs step. This is similar to the Gibbs step in the previous question, but simpler in some aspects (we do not have to worry about likelihood models), but slightly more complicated in other aspects (we have a hierarchy of priors now).
The missing key is 'addCustomerAtRandom()', which assumes that the customer has been taken out at random (by takeOneRandomCustomerOut()), and now needs to be reinserted. See in this function for hints.
Once it is implemented, change the variable 'addDPLanguageModel' to true in 'LanguageModelEvaluation.java'. You should get a much lower (better) perplexity now, around 158. Also look in the file 'HDPLanguageModel.generated', you will see slightly more (locally) realistic looking sentences.

Part B

Do a few experiments with the current model. Some examples: scaling the training size (can be controlled with LanguageModelEvaluation.nTrainWords), the order of the language model (lmOrder), changing the number of Gibbs scan, α_0, or size of testing set. Plot the performance as a function of at least one of these quantities. What is the main limitation(s) of this model in your opinion?

Part C (Optional)

Extend the model in one or several ways. Examples include: transforming the model into a PY process, learning or resampling the hyper-parameters (see dp.Concentration for an implementation of resampling α_0), or scaling it to a larger dataset (the data to load is controlled by pathToData—see data/POS for an example of the format (note that POSs can be filled by arbitrary strings)).