INF367: Spring 2020
Project 2: Bayesian recommendation engine

Deadline: May 15th, 23.59
Deliver here: `https://mitt.uib.no/courses/23594/assignments/29327`

Deliverables. During the project, you will do 3 warm-up tasks and complete 3 quizzes (one for each warm-up task). **Deadlines for the warm-up quizzes are 24.4., 1.5., and 8.5.**

In addition, you will complete the project. For the final project, you should return **exactly two** files:

1. A pdf file containing a report; see Section 6.3 for more details
2. A zip file containing code

Grading: Grading of the final project will be based on the following criteria:

- Correctness (your answers/code are correct and clear)
- Quality and thoroughness of experiments
- Reporting (thoroughness and clarity of the report)
- Clarity of code (documentation, naming of variables, logical formatting)

This project will count 25% towards your final grade.

Warm-up tasks are graded using pass/fail scale (The reason why they have earlier deadlines is to force everybody to start working early).

Late submission policy: All late submissions will get a deduction of 2 points. In addition, there is a 2-point deduction for every starting 12-hour period. That is, a project submitted at 00.01 on May 16th will get a 4-point deduction and a project submitted at 12.01 on the same day will get a 6-point deduction (and so on). All projects submitted on May 18th or later are automatically failed. (Executive summary: Submit your project on time.) There will be no possibility to resubmit failed projects so start working early.

If you are sick during the project period, the deadline will be extended accordingly.

1 Background

Recommender systems are widely used in online commerce. A common use-case for recommender systems is a product recommender. However, they are used also in other domains. For example, medical companies use recommender systems to speed-up development of new drugs by recommending chemical compounds to be tested in a laboratory.

There are different types of recommender systems. In this project, we will concentrate on *collaborative filtering* where we try to predict user preferences based on preferences from several other users.

Matrix factorization is an important tool in collaborative filtering. In matrix factorization, the goal is to represent a large matrix as a product of two or more smaller matrices. For example, PCA is a matrix factorization technique.

Formally, let $X \in \mathbb{R}^{n \times d}$ be a data matrix; X is often sparsely observed. In low-rank matrix factorization, the goal is to find two smaller matrices $Z \in \mathbb{R}^{n \times k}$ and $W \in \mathbb{R}^{d \times k}$ with $k \ll n, d$ such that

$$X \approx ZW^T.$$

Often, one uses maximum likelihood estimation to find W and Z. To build a Bayesian recommender system, we of course set prior distributions on W and Z and estimate the posterior distribution $P(W, Z \,|\, X)$.

We will build our recommender engine using a probabilistic programming language called Stan. Probabilistic programming languages combine a language to specify models with a general-purpose sampler. This makes it easy to get samples from the posterior.

The project is divided into three warm-up tasks and the main project. The idea of the warm-up tasks is to make sure that everybody starts working early. The deadlines for completing the warm-up tasks are 24.4., 1.5., and 8.5.

2 Warm-up I: Deterministic matrix factorization and recommender systems

The goal of this tasks is to familiarize us with recommender systems and matrix factorization in general. You can start with the following material:

- Watch the video: `https://www.youtube.com/watch?v=ZspR5PZemcs`
- Do the tutorial: `https://developers.google.com/machine-learning/recommendation` (You can skip the softmax model and neural networks)
- Read the paper: `https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf`

Task: Learn about recommender systems and complete the quiz at `https://mitt.uib.no/courses/23594/quizzes/10627`

Deadline: April 24th

3 Warm-up II: Probabilistic programming and Stan

Deriving update formulas and implementing them for every single model is a tedious task. Fortunately, we can avoid this task: These days there are several probabilistic programming languages/tools that make modelling easy by automating sampling.

In this task, we learn to use a probabilistic programming language called *Stan*[1]. Probabilistic programming languages like Stan can make modeller's life much easier. In Stan, you specify the model and the systems takes care of the rest. That is, you do not have to derive and implement update equations. Stan also provides different convergence diagnostics and warns the user if something seems to be going wrong[2]. In addition, there are different visualization tools readily available.

3.1 Hamiltonian Monte Carlo

Stan does inference using Hamiltonian Monte Carlo (HMC). You can think HMC as Metropolis-Hastings sampling with a smart (but very complicated) proposal function[3]. In standard Metropolis-Hastings sampling, the proposed states do not depend on the acceptance probability which can lead to high rejection rate. HMC, on the other hand, uses information about the shape of the distribution and tries to propose states that have high acceptance probability.

Hamiltonian Monte Carlo spends more time per sample than Gibbs sampling or Metropolis sampling. However, HMC tends to produce less correlated samples and thus one usually needs less samples.

HMC works reasonably well for many inference problems. One should note, however, that when efficient inference is critical (for example, with very large data sets), one can benefit from tailor-made samplers.

3.2 Convergence and mixing

Stan provides tools for convergence diagnostics. Stan reports potential scale reduction factor[4] $\hat{R}$ for each parameter. If the chains have converged, $\hat{R}$ should be near 1. If it seems that the chain has not converged then Stan usually gives suggestions how to tweak the sampling algorithm.

If your chain does not converge, you can try at least the following things:

- Change the parameters that Stan mentions in the warnings

[1] `https://mc-stan.org`

[2] One should remember, however, that Stan does not necessarily recognize all problems.

[3] Hamiltonian Monte Carlo is too complicated to be covered on this course. Those who are interested to know more about inner workings of HMC, can start with, for example `http://www.mcmchandbook.net/HandbookChapter5.pdf` and `https://arxiv.org/pdf/1701.02434.pdf`.

[4] `https://mc-stan.org/docs/2_18/reference-manual/notation-for-samples-chains-and-draws.html`

- Longer chain
- Different hyperparameter values (You may have too much/too little regularization)
- Different model

To diagnose mixing, Stan reports effective sample size, that is, a rough estimate on to how many independent samples the samples from the chain correspond.

3.3 Visualisation

There exists tools for visualising the results of inference. For example, the `arviz` package[5] can be useful.

3.4 Warm-up task

To prepare for the quiz, you should complete the following tasks:

1. Install Stan (Instructions can be found here: `https://pystan.readthedocs.io/en/latest/`)
2. Implement the Poisson-Gamma model (Weekly exercise 5.1) in Stan.
3. Load data `exercises5_1.txt`. Learn the posterior of the parameter λ using Stan.
4. Compare your results to the exact solution in Exercise 5.1. Does Stan give you reasonable results?

Task: Practice using Stan and complete the quiz at `https://mitt.uib.no/courses/23594/quizzes/11163`.

Deadline: May 1st

4 Bayesian matrix factorization

Before going to the final warm-up exercise and the main project, let us first define the Bayesian matrix factorization problem. Bayesian matrix factorization is an example of latent factor models.

Suppose we a sparsely observed data matrix $X \in \mathbb{R}^{n \times d}$ where n is the number of users and d is the number of items. Furthermore, let Obs denote the set of the indices of the observed values. Furthermore, let $Z \in \mathbb{R}^{n \times k}$ be a latent factor matrix and $W \in \mathbb{R}^{d \times k}$ be a weight matrix; k is a user-defined parameter.

[5] `https://arviz-devs.github.io/arviz/index.html`

Let us use Gaussian likelihood

$$P(X \mid W, Z, \beta) = \prod_{(i,j) \in Obs} N(X_{ij} \mid Z_{i\cdot} W_{j\cdot}^T, \beta^{-1})$$

where $Z_{i\cdot}$ is the ith row of Z and $W_{j\cdot}$ is the jth row of W. Furthermore, we need prior $P(W)$, $P(Z)$, and $P(\beta)$; there are lots of alternative ways to choose these priors so we use generic priors here. As usual, the goal is to infer the posterior $P(W, Z, \beta \mid X) \propto P(X \mid W, Z, \beta)P(W)P(Z)P(\beta)$.

Note that the above model has $(n + d)k + 1$ parameters whose posterior distribution we want to estimate. This means that even with a relatively small matrix, we end up with thousands of parameters.

5 Warm-up III: Simple bilinear model

As we learned in Section 2, recommender systems are often constructed using matrix factorization. Here we illustrate one challenging aspect of such models called *unidentifiability*. Intuitively, unidentifiability means that the likelihood or posterior has several modes (maxima).

Let us illustrate latent variable models with a simple bilinear model $x_i \approx w z_i$ where x_i is a 1-dimensional observation, w an unobserved weight and z_i a latent variable. We use a Gaussian likelihood $x_i \sim N(w z_i, \lambda_{noise}^{-1})$ with known precision and Gaussian prior for $w \sim N(\mu_w, \lambda_w^{-1})$ and $z_i \sim N(\mu_z, \lambda_z^{-1})$.

If $\mu_w = 0$ and $\mu_z = 0$ then it is easy to see that $P(w, z \mid x) = P(-w, -z \mid x)$. Thus, the posterior has two modes. (General matrix factorization models suffer from an additional source of unidentifiability called label switching. That is, you can shuffle the k columns of W and Z without affecting the posterior probability.)

Why is unidentifiability problematic? First challenge is that samplers typically stuck around one mode. Moving from one mode to another requires typically several unlikely proposals to be accepted and therefore one needs to have a very long chain to sample equally from several modes.

Another consequence of multimodality is that the posterior mean is a very bad estimate. In the bilinear model above, the posterior mean for both w and z is 0 (still assuming $\mu_w = 0$ and $\mu_z = 0$). So if you succeeded to get a samples from from the true posterior and decided to predict using the posterior mean, you usually get catastrophically bad results.

Task: Download the jupyter notebook `bilinear_model.ipynb`. The file contains a toy matrix factorization model where we factorize a 1×1 matrix.

In the light of above, try to get an understanding how the simple model and the sampler work.

- Try to change the value of the observation. What will happen when you move away from the zero?
- Try different priors. What happens if the priors from W and Z are not identical? What happens if at least one of them is not zero-centered?

- Try to sample different number of samples. What happens when the chain gets longer?

Once you have done the above tasks, complete the quiz at `https://mitt.uib.no/courses/23594/quizzes/11181`

Deadline: May 8th

6 Main project: Bayesian movie recommender system

The main project is to build a recommender system based on Bayesian matrix factorization. That is, we want to predict ratings given by users as well as estimate uncertainty in predictions.

We use MovieLens[6] data to build a movie recommender system; You can download the file `ml-100k.zip` from MittUiB. The zip-file consists of several files. The actual ratings can be found in file `u.data`. The data matrix is stored in a sparse format. Each row of the file corresponds to an observed ratings. Each row has four columns: the ID of the user, the ID of the movie, a rating, and a timestamp. The data consists of in total $100,000$ ratings $(1-5)$ from 943 users on 1682 movies.

Tasks:

1. Define at least three different versions of Bayesian matrix factorization.
2. Implement your models using Stan.
3. Learn the posteriors given MovieLens data.
4. Evaluate and select models. Note that you also need to select k (too large k may lead to overfitting).
5. Quantify uncertainty in your predictions using predictive distributions.

To clarify the above instructions, different model means different types of distributions. In other words, changing values of hyperparameters does not count as a new model.

When you create the train/test split, select observations randomly. (If you pick random rows/columns for your test set, then your test will contain users/movies that were not in the training set.)

6.1 Potential tweaks

Here are some ideas on how to tweak your models. Naturally, implementing your own ideas is highly appreciated.

[6] `https://grouplens.org/datasets/movielens/`

As a general rule, start with a simple model and then try to improve it.

Prior distributions for the factors W and Z, $P(W_{j\cdot})$ and $P(Z_{i\cdot})$ are often Gaussians. If the prior mean and variance are not known, we can place a hyperprior on them (or just one of them). Hamiltonian Monte Carlo does not require conjugate priors so you have lots of freedom in choosing a distribution. If you want to have flexible regularization, you can try sparsity inducing priors like ARD or horseshoe[7].

Of course, $P(W)$ and $P(Z)$ do not have to be Gaussians. We notice that the ratings are positive. Thus, we could try non-negative matrix factorization, that is, constraining $W \geq 0$ and $Z \geq 0$. In this case, you should use some distribution whose support is non-negative values. Examples include exponential and Gamma distributions.

If you want to induce sparsity, you can use Laplace distribution (called `double_exponential` in Stan) for $P(W)$ and $P(Z)$.

The matrix factorization model is unidentifiable which may cause trouble in sampling. You can try to get rid of unidentifiability using "symmetry breaking" priors[8].

6.2 Practical challenges

One practical challenge with sampling is that it takes time and requires computational resources. If your computer is too slow to handle the full data matrix, **it is ok to use only a subset of rows and columns**. To get best results, keep the rows and columns with most observations.

At minimum, you want to have few hundreds of samples. If getting this takes hours, then you should consider a smaller matrix. However, the extensive running time can be caused by some other issues, too[9]. Note that we are working with a fairly difficult model with a reasonably large number of parameters and thus **sampling is going to take time also in a good case so start working early**.

Ideally, one should run several chains. However, this is problematic for matrix factorization because of unidentifiability. (Different chains can sample from different modes and this makes using $\hat{R}$ to assess convergence unusable.)

6.2.1 Hamiltonian Monte Carlo works in mysterious ways

Note that due to the special proposal function, the time that Stan uses to produce a sample can vary drastically even within one chain. That is, it can happen for example that sampling 200 samples takes ten times more time than sampling 100 samples.

[7] See, for example, `https://betanalpha.github.io/assets/case_studies/bayes_sparse_regression.html`

[8] See, for example, `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3419391/`.

[9] The shape of the posterior distribution affects the sampling speed. Some models are significantly faster to sample than others. Also hyperparameter values can have a large effect.

There is also random variation depending on the initial values and random numbers. I have experienced cases where doubling the number of samples actually reduces running time.

6.3 Report

At minimum, the report should contain the following information.

The report should contain specification all models that you have used. You should explain the rationale behind choosing these particular models. What are trying to achieve? Why do you think it will work?

How did you assess convergence and mixing? If there were problems with convergence, how did you try to mitigate them?

What were the results from the inference? How well did the models perform? Which model is best? Remember to visualise your results.

Self-evaluation: What did I learn? What things went well? What could have been done better?