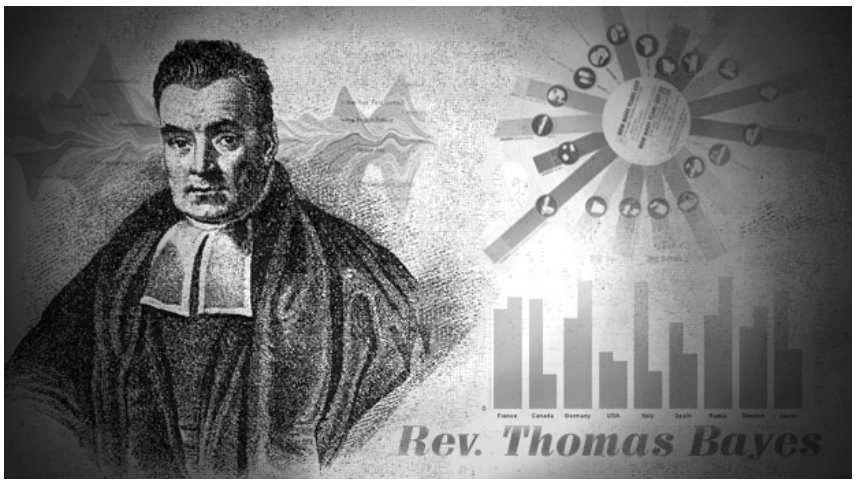
Kungliga Tekniska Högskolan
Valhallavägen 79
100 44 Stockholm

Statistical Methods in Applied Computer Science

« Project 2 - The magic word »

December 15^{th} - January 18^{th}



Author
Rémi DOMINGUES

Teacher
Jens LAGERGREN

Scholar year 2014-2015

1 Intro

The aim of this project is to implement a **generator** which outputs **sequences** containing a **magic word**, and a **Gibbs sampler** which aims at finding the magic words in the previous sequences.

For this purpose, we generate N sequences s_i of length M, each containing a magic word of length w. Our Gibbs sampler should find $R = r_1, ..., r_n$ with r_i the starting position of the magic word in the i^{th} sequence.

The **background** of a sequence is the union of every observation in a sequence which does not belong to the magic word.

A **magic word** is the union in a sequence of every observation having its index i between r_i and $r_i + w - 1$.

The experiment described in this report will focus on finding this motif described by a Dirichlet prior related to an alphabet. We have chosen to apply our Gibbs sampler to DNA sequences. Therefore, the alphabet K is K = {A, T, G, C}.

2 Generation

The background and magic word have the same alphabet

We start by generating for each sequence the starting position r_i of the magic word : $r_i = U(N - w + 1)$

Then, we sample the background of each sequence :

$$s_{i,j} \sim Cat(x|\theta) \tag{1}$$

With with $\theta \sim Dir(\theta|\alpha')$ and $s_{i,j}$ the letter j in the sequence i.

We also do the same for the magic words :

$$s_{i,k} \sim Cat(x|\theta_j) \tag{2}$$

With $\theta \sim Dir(\theta_j|\alpha)$ and $k \in [r_i, r_i + w - 1]$.

α and α' are vectors of continuous reals, and parameters of our Dirichlet distributions which are multivariate probability distributions.

For our generation process, we define $\alpha' = (1, ..., 1)$ as a vector of same size as the alphabet K. The categorical distribution obtained from $Dir(\alpha)$ will here contain a lot of randomness, since we do not specify a higher weight for any letter of the alphabet. However, all sequences will be generated from the same categorical distribution. If using a α different enough from α', we should thus be able to decide whether a sequence of letters has been generated from $Dir(\alpha)$ or $Dir(\alpha')$.

We will use various α, parameter for the magic word generation, during this experiment. The tricky part in the inference process is that each letter in a given magic word is generated using distinct samples from $Dir(\alpha)$, giving us distinct categorical distributions. This should add more randomness to our words, and therefore will require more data to give an accurate r_i. Let us precise however that every word will be generated from the same set of categorical distributions (one distribution for each letter).

The code of the generator is written in *generator.py*

3 Inference

Our goal is now to implement a Gibbs sampler in order to estimate the posterior $P(R|D)$ with $R = r_1, ..., r_N$ and $D = s_1, .., s_N$.

Each state of our Gibbs sampler will be a $\widehat{R_i}$ with $\widehat{R_i} = \widehat{r_{i,1}}, .., \widehat{r_{i,N}}$. Finding those steps by iterating is the aim of the Markov Chain Monte Carlo algorithm described here.

Please note that α and α' are given to our algorithm.

To simplify to computation of the posteriors, we use a collapsed Gibbs sampler, i.e. we collapse the Dirichlet distribution $Dir(\theta_j|\alpha)$ which has been used as prior to generate our categorical distributions $Cat(x|\theta_j)$. Collapsing our Gibbs sampler will introduce a dependences between those categorical distributions.

In order to get a better accuracy for our results, we must also pay attention to the *burn-in period* and the correlation between a state (a R_i in the Markov Chain generated by our Gibbs sampler) and its child. Thus, once this Markov Chain generated, the first S steps (*burn-in period*) must be ignored since the stationary distribution has not been reached yet by our algorithm, giving unstable results until we reach a convergence.

Then, we must build our final results by aggregating every i^{th} state after the state R_S. Keeping a certain number of steps between each considered for the results should lower the impact of the correlation between the states, and improve the final accuracy.

The implementation follows the shema below :

▷ Gibbs sampling
Generate randomly R_0
for i from 0 to NITER **do**
    for j from 0 to NSEQUENCES **do**
        $P(\widehat{r}_{i_j}|\widehat{R}_{i_{-j}}, D) = \cup_{l=0}^{M-w} P(D_{background}|\widehat{R}_{i_{-j}}, \alpha') \prod_{k=l}^{l+w} P(D_{motif_k}|\widehat{R}_{i_{-j}}, \alpha)$
        $\widehat{r}_{i_j} = sample(P(\widehat{r}_{i_j}|\widehat{R}_{i_{-j}}, D))$ ▷ $\widehat{r}_{i_j}$ is generated from the categorical $P(\widehat{r}_{i_j}|\widehat{R}_{i_{-j}}, D)$
    end for
end for
▷ Results sampling from the Markov chain generated by the Gibbs sampler

10: **for** i from 0 to NSEQUENCES **do**
11: seq = sequences[i]
12: $\widehat{r}_i = argmax(count_{t=S:k:NITER}(\widehat{R}_{t_i}))$ ▷ We select the starting index which has been the most elected for a sequence. Only the k^{th} results after the S first states are taken into account (steps from S to N with a step size of k)
13: **end for**

With

$$P(D_{background}|\widehat{R}_{i\_l}, \alpha') = \frac{\Gamma(\sum_{k=1}^{|K|} \alpha'_k)}{\Gamma(|K|(M-w) + \sum_{k=1}^{K} \alpha'_k)} \prod_{k=1}^{|K|} \frac{\Gamma(B_k + \alpha'_k)}{\Gamma(\alpha'_k)} \tag{3}$$

and

$$P(D_{motif_j}|\widehat{R}_{i\_l}, \alpha) = \frac{\Gamma(\sum_{k=1}^{|K|} \alpha_k)}{\Gamma(|K| + \sum_{k=1}^{|K|} \alpha_k)} \prod_{k=1}^{|K|} \frac{\Gamma(N_k + \alpha_k)}{\Gamma(\alpha_k)} \tag{4}$$

Where $|K|$ is the length of the alphabet K, B_k is the number of times the symbol K_k appears in the background, and N_k is the number of times the symbol K_k appears in the given column j of the motif.

Therefore, the equation in line 4 returns a categorical from which we sample a starting position $\widehat{r}_{i_l}$ for the i^{th} state for the sequence l. This is done by computing equations 3 and 4 for every possible value $\widehat{r}_{i_l}$ in a range from 0 to M-w-1, i.e. by using $\widehat{R}_i$ with various $\widehat{r}_{i_l}$.

The implementation of this algorithm can be found in *gibbs_sampler.py*

4 Testing

The performances depend on the whether the differences between α and α' are important or not. Obviously, if one of the α_i is really high, we are more likely to detect the magic word with high accuracy.

With $\alpha' = (1, ..., 1)$ and $\alpha = (1, 7, 10, 2)$, we can observe the following results (5 sequences of length 30 with a magic word of length 10, sample selection with 200 iterations, every 10^{th} iteration from the 100^{th}) :

FIGURE 1 – Starting position (vertical axis) for each sequence (function) - 40% accuracy

We can observe quite stable results. However, the the *burn-in* period seems nonexistent with an immediate convergence. For the previous chart we had an accuracy of 40%. The real positions were [13, 8, 2, 17, 11] and the positions found [13, 4, 0, 17, 8].

We must remind also that an accuracy of 40% means that we have found 40% of the correct starting positions among $N * (M - w)$ positions, so we found for example 2 correct positions among the 5 real positions in a set of 150 positions, which is still really nice, considering that a wrong position can be very close to the real position, and be misleaded because of an unlucky sequence generation.

Those results are highly dependent on the sequence sampling, since it can be hard sometimes to find when a magic word starts. Below is another example for the same parameters :

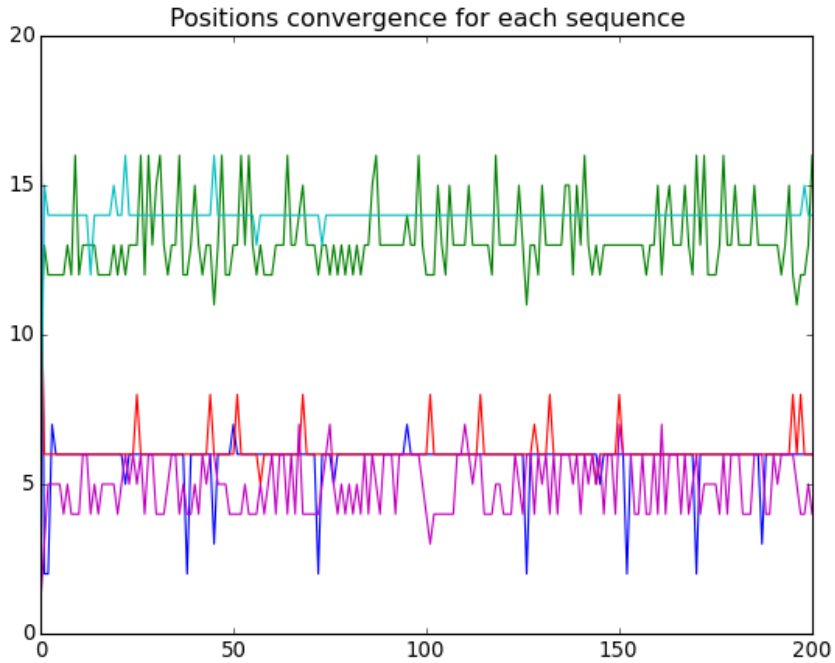


FIGURE 2 – Starting position (vertical axis) for each sequence (function) - 60% accuracy

60% accuracy were found here, with the real positions [6, 12, 6, 14, 5] and the estimated positions [6, 13, 6, 14, 4] (with 13 and 4 really close to the real positions 12 and 5).

Using a **higher number of sequences** or a α **and** α' **more different** could therefore lead to better results.

We now use $\alpha = (1, 7, 20, 2)$, increasing the difference between α and α' to ease the identification :

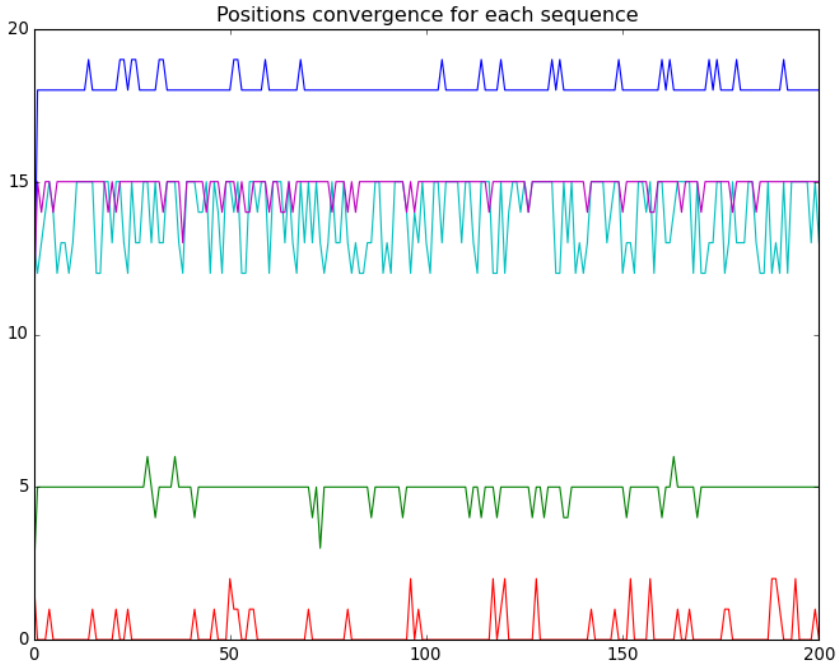


FIGURE 3 – Starting position (vertical axis) for each sequence (function) - 100% accuracy

100% accuracy is now observed for the starting positions [18, 5, 0, 12, 15].

However, our performances also depend on the parameters chosen for our samples selection among the states generated by our Gibbs sampler. The *number of iterations* is the number of states generated, *min_ step* is the length of the *burn-in period* and *step_ size* determines the number of steps between each state taken into account for our results (all of these steps are between *min_ step* and the total *number of iterations*).

They also depend on the parameters selection for N, M and w. Below are various benchmarks for $M = 20$:

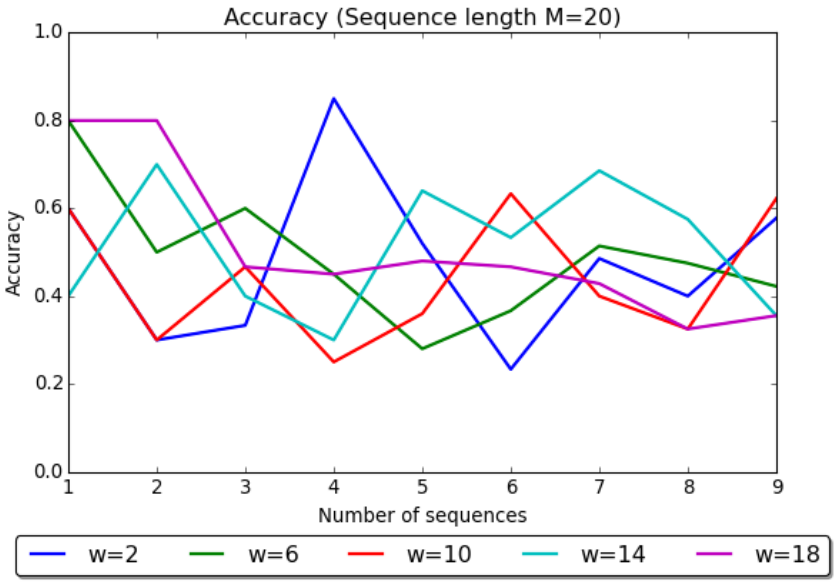


FIGURE 4 – Benchmarks for various N and w for $M = 20$

Nevertheless, we can hardly infer any trend and certainty from those functions. Indeed, despite of running the estimation for 5 different sampling for a given N and w, no trend can be observed here.

However, we can at least see that using more sequences should make our results more stable. Theorically, if we had not run the sampling multiple times for a given configuration, we should also have had better and more stable results since we make our Gibbs sampler more resistant to the randomness induced when sampling our sequences.

For more details, the testing implementation is written in *testing.py*