Markov Chains and Algorithmic Applications Project

Adrian Valente Armand Boschin Quentin Rebjock

December 18, 2017

1 Introduction

This project aims at learning unknown parameters by minimizing a cost function through a specific probabilistic algorithm, the Markov Chain Monte-Carlo (MCMC) method. This process is particularly efficient to minimize functions whose domains' dimensions are large or admitting a lot of local minimums. In these cases, most greedy algorithms often fail to converge.

More formally, let N be a large integer and ω^* be an unknown element of $\{-1, 1\}^N$. We would like to find the best approximation of ω^* by learning it from a set of M given features $x_i \in \mathbb{R}^N \quad \forall i \in \{1, \ldots, M\}$ and labels $y_i = sign\left(\omega^* \cdot x_i\right)$. To measure how well an estimation ω of ω^* is, we define the following energy (or cost) function :

$$E\left(\omega\right) = \frac{1}{2} \sum_{i}^{M} \left(y_i - sign\left(\omega \cdot x_i\right)\right)^2$$

The lower the cost is, the better the predicted ω is.

ω^* is generated randomly, each component having a probability of $\frac{1}{2}$ to be either -1 or 1. For $i \in \{1, \ldots, M\}$, x_i is also generated randomly, its components being i.i.d Gaussian random variables with mean 0 and variance 1. Eventually $(y_i)_{i \in 1..M}$ is computed from ω^* and $(x_i)_{i \in 1..M}$.

Obviously, ω^* minimizes the cost function E and a brute-force search would find its exact value. However, there are 2^N possible values of ω^* and that naive process is not viable for N large.

Knowing the features x_i and the labels y_i, we can hope finding a good approximation of ω^* using some probabilistic algorithms, provided that they are in sufficient numbers, i.e that M is large enough. Let $\alpha = \frac{M}{N}$ be the ratio of the number of examples divided by the number of components to learn.

2 The MCMC method

The Markov Chain Monte-Carlo method consists in starting with an initial random estimation $\omega^{(0)}$ and applying the Metropolis-Hastings algorithm :

1. We fix a constant $\beta > 0$.

2. We pick an index i uniformly at random and let ω' the current vector $\omega^{(t)}$ with flipped value at index i.

3. Let $p\left(\omega^{(t)}, \omega'\right) = min\left(1, \exp\left(-\beta\left(E\left(\omega'\right) - E\left(\omega^{(t)}\right)\right)\right)\right)$. Then,

$$\omega^{(t+1)} = \begin{cases} \omega' \text{ with probability } p\left(\omega^{(t)}, \omega'\right) \\ \omega^{(t)} \text{ with probability } 1 - p\left(\omega^{(t)}, \omega'\right) \end{cases}$$

4. Repeat the process from step 2. iteratively.

2.1 The approach we followed

In theory, N is supposed to be very large (tend to $+\infty$) but the computations become rapidly very long. That's why we chose $N = 1,000$ which is a good compromise and which will remain a fixed value. However, the value of α can easily be modified so M is not a fixed value.

We ran the MCMC algorithm for several values of α and β and observed how these parameters influence the results.

2.2 Results

For all $(\alpha, \beta) \in [1, 5] \times [1, 5]$ we ran the MCMC algorithm 5 times and averaged the energies obtained over $T = 100,000$ iterations. Here are the plots for $\beta = 1$ and two extreme values of α :

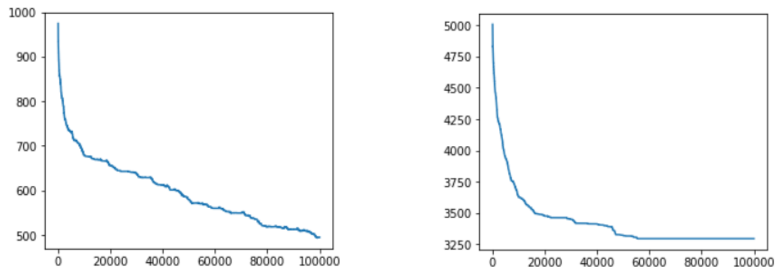

Figure 1: Energies as a function of the number of iterations for $\beta = 1$ and respectively $\alpha = 1$ and $\alpha = 5$

The energy is globally decreasing and when a plateau is reached it has to first increase slightly to escape from it (it is not visible on this figure though because the scale is two small).

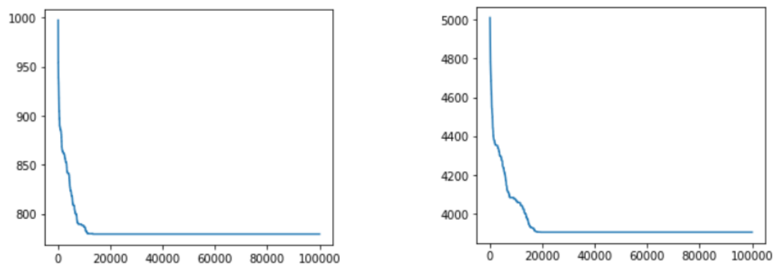

Figure 2: Energies as a function of the number of iterations for $\beta = 5$ and respectively $\alpha = 1$ and $\alpha = 5$

One can observe that the higher β is, the more plateaus of stagnation there are. The algorithm is indeed more prone to get stuck in local minimums.

The case $\alpha = 1$ and $\beta = 1$ seems to be optimal : it is where the energy seems to be decreasing the fastest and attaining the lowest value.

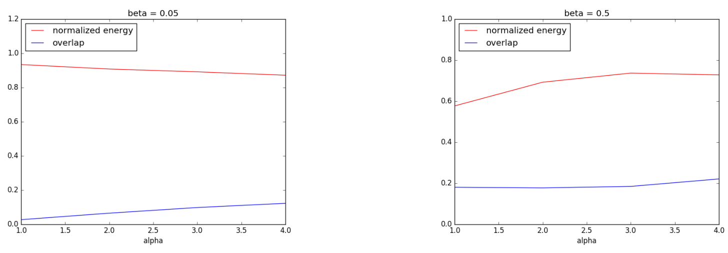

Figure 3: Normalized energies and overlaps

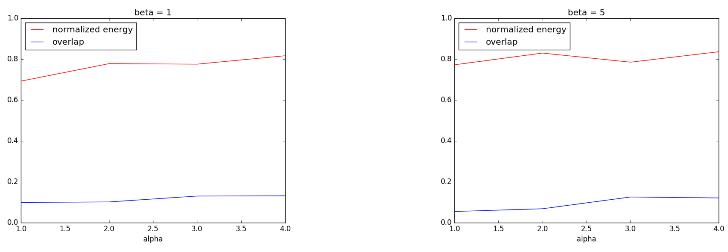

Figure 4: Normalized energies and overlaps

We could expect from a classic machine learning algorithm to get better results when the number of examples (M here) increases. We can see from these plots that it is not the case here. Our interpretation is that for a fixed value of β, the local minimums of the energy are deeper when α increases and it's harder to get out of them.

3 Simulated annealing

The simulated annealing algorithm is based on the MCMC method and is used to improve the estimation of ω^*, essentially by slowly increasing the value of β. When β is low, the algorithm doesn't get stuck in local minimums because the probability $p\left(\omega^{(t)}, \omega'\right)$ is high, but it doesn't converge rapidly because it can easily get out of the minimum that it found. On the contrary, when β is high, it often gets stuck in local minimums but it converges quickly. The idea is to start with a value of β sufficiently low and to increase it periodically to take advantage of both behaviors.

The algorithm we chose was to start with $\beta = \beta_{\texttt{initial}}$ and to multiply it by a constant every $\beta_{pace} = 500$ iterations : $\beta = \beta_{\texttt{increase}} * \beta$. This gives us 2 hyper-parameters to play with to obtain the best result. The energies attained after 10,000 iterations are plotted in figure 5 for different values of the hyper-parameters and $\alpha = 1$, but other factors can influence the choice of hyper-parameters (like the speed of convergence).

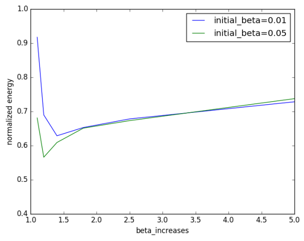

Figure 5: Cross-validation for the choice of $\beta_{increase}$ for different values of $\beta_{initial}$

It is interesting to see that for each value of $\beta_{initial}$ there is a single optimal value for $\beta_{increase}$. In order to understand what was going on, we plotted the evolution of the energy as a function of time for different values of the hyper-parameters, which is displayed in the following figure. In graphic on the left we see that if β increases too slowly, the algorithm tend to easily accept moves that increase the energy, and thus does not converge fast enough. In the figure on the right we can see that if β increases too quickly, the algorithm tends to get stuck in local minima from which it can hardly escape. Finally, the values that we chose give the figure in the middle which is a compromise between those extremes.

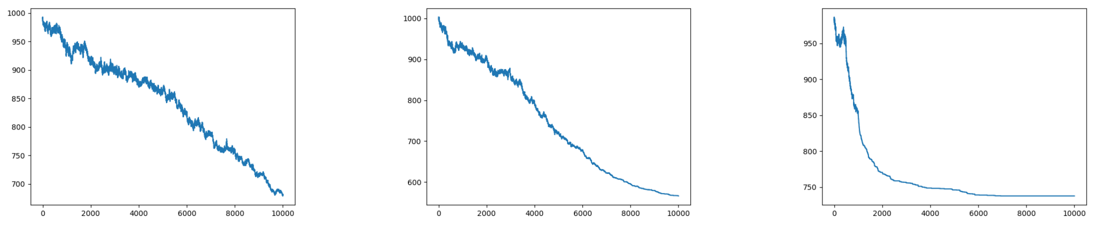

Figure 6: Evolution of the energy for hyper-parameters $\beta_{initial} = 0.05$ and respectively $\beta_{increase} = 1.1$, $\beta_{increase} = 1.2$, and $\beta_{increase} = 5$

Finally, we stood for the values $\beta_{\texttt{initial}} = 0.05$ and $\beta_{increase} = 1.2$ and tried different values of α, as plotted in figure 7.

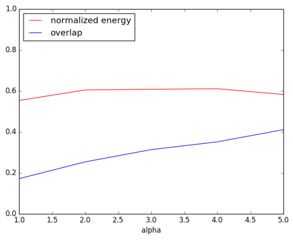

Figure 7: Normalized energies and overlaps as functions of α for $\beta_{\texttt{initial}} = 0.05$ and $\beta_{\texttt{increase}} = 1.2$

4 Conclusion

The choice of the hyper-parameters to get the best results from the MCMC method and the simulated annealing is complicated and it seems like there is no general rule to choose them. We get satisfying approximating values of ω^* in a reasonable computing time.