Projet de MC2A : Équipe one

Redouane ELGHAZI Pierre MAHMOUD–LAMY Enguerrand PREBET

1 Introduction

Le but de ce projet était d'implémenter une méthode de Monte-Carlo par chaînes de Markov appelée algorithme de Metropolis-Hastings. Cet algorithme a pour entrée un ensemble P de points de $\mathbb{R}^N$ et une fonction $label^*$ associant un label à chaque point.

Le but de l'algorithme est de trouver une fonction *label* minimisant le nombre de mauvais labels. Dans le cadre de ce projet, la fonction recherchée est de la forme :

$$label(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x})$$

Où w est un vecteur de $\{-1, 1\}^N$. Cet objectif est modélisé par une énergie proportionnelle au nombre de mauvais labels.

Pour se faire, à chaque étape, un bit de $\mathbf{w}$ est proposé à la modification, et est accepté avec une probabilité dépendant du nombre de points mal classifiés avant et après modification. La distribution de Boltzmann est utilisée faisant intervenir l'énergie citée précédemment.

2 Langage

Nous avons choisi d'utiliser deux langages pour ce projet. L'algorithme est implémenté en `C++`. Nous avons choisi ce langage pour ses performance et le contrôle que l'on peut avoir sur les opérations élémentaires.

Nous avons ensuite testé les performances de notre exécutable à l'aide d'un script `Python`. Nous avons choisi le `Python` car c'est un langage de script haut niveau, qui permet aisément de générer des graphes.

3 Implémentation

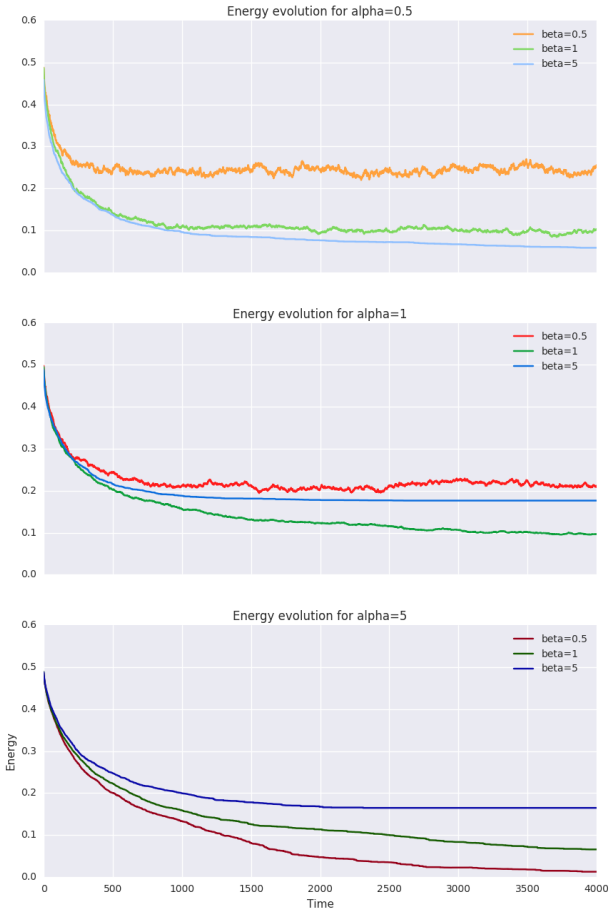
Le programme en C++ génère $\mathbf{w}^*$ et les $(\mathbf{x}_\mu)$ avant d'effectuer l'algorithme selon les paramètres données en entrée. Selon la question, il écrit dans un fichier les énergies normalisées tout au long de l'algorithme, ou bien uniquement l'énergie normalisée ainsi que le chevauchement entre le modèle original et notre réponse finale.

Comme au plus un bit est changé à chaque ité-

ration, il est possible de modifier le label de chaque $\mathbf{x}$ en temps $O(1)$. Cela nous permet d'effectuer une itération en $O(M)$ pour une complexité totale en $O(M(T+N))$ en comptant l'initialisation.

4 Résultats

Nos résultats sont obtenus pour $N = 100$, ce qui dans notre implémentation est instantané pour des paramètres adéquats de α et T.



On peut remarquer que pour ces 3 tests avec des valeurs α différentes, le β optimal diffère, allant du plus grand pour $\alpha = 0.5$, au plus petit pour $\alpha = 5$. Cette valeur nous permet aussi d'obtenir une erreur d'au plus 10% dans les 3 cas.

5 Analyse de la valeur de β

oui des plot c'est cool (q2-3) puis replot de la q1 avec des beta hinted

6 Qu'apporte le recuit simulé ?

La méthode du recuit simulé ()

7 Conclusion

De toutes nos observations, il ressort que l'algorithme de Metropolis-Hastings permet d'atteindre un taux de classification plutôt élevé, et que ce taux est encore plus élevé en utilisant la méthode du recuit simulé.

8 Pistes d'amélioration

Il pourrait être intéressant de changer la manière qu'a la température de diminuer avec le recuit simulé (plutôt que d'augmenter linéairement son inverse).

Par ailleurs, dans la chaîne de Markov étudiée, chaque état a N voisins. Ce nombre de voisins pourrait être augmenté à $2^N - 1$ (i.e. travailler sur un graphe complet). Il serait par exemple possible de proposer la modification de K bits à chaque étape, K étant une variable aléatoire. De plus il serait possible de conserver une complexité moyenne en $O(M*(N+T))$ en choisissant par exemple une loi géométrique pour K. Cela permettrait peut-être d'éviter certains minima locaux.