Elements Logiciels pour le Traitement de données Massives

Homework

Gabriel Kasmi, Hugo Thimonier

3 février 2020

Résumé

Ce projet a pour objectif d'optimiser la vitesse d'execution d'un Dirichlet Process Mixture Model (DPMM). Ce dernier est un modèle bayésien non paramétrique de classification non supervisé qui permet de clusteriser des données sans disposer à priori du nombre de cluster.
Pour optimiser la vitesse d'execution nous avons procédé en deux temps, d'une part nous avons utilisé cython pour coder certaines parties de l'algorithme identifiées comme coûteuses en temps d'execution, puis nous avons parallélisé les étapes de l'algorithme qui pouvaient l'être. Les résultats sont encourageants car semblent montrer une amélioration de la vitesse d'execution de l'ordre d'un acteur 10.

Introduction

L'idée générale est d'implémenter une méthode permettant de clusteriser une base de données à partir de features X dans un sous-espace de $\mathbb{R}^d$. Ce type de problème de classification non supervisée est courant et très utile car elle permet d'identifier plusieurs groupes au sein d'une population. Les applications de ce type d'approche sont nombreuses : la segmentation de clientèle, la détection d'outliers, la détection de fraudes ... etc.
Il existe un certain nombre d'algorithmes qui permettent de réaliser ce genre de clusterisation, les plus connus étant le *K-means* ou encore *la Classification Ascendante Hiérarchique*. En ce qui concerne le K-means, il est nécessaire de connaître à l'avance le nombre de clusters présents dans la base de donnée, or cette hypothèse est très souvent restrictive et donc très rarement vérifiées dans des cas réels. De même, le nombre de cluster dans le cas de la Classification Ascendante hierarchique est choisi de manière discrétionnaire.
Une des méthodes bayésienne non paramétrique les plus efficaces pour contourner ce type de problème est le Dirichlet Process (Gaussian) Mixture Model (DPMM). Ce modèle est une généralisation pour K *a priori* infini des modèles de mélange dont la variante la plus simple est le modèle de mélange binomial. L'attrait principal de ces modèles, également appelés *infinite mixture models*, est qu'il n'est pas nécessaire de connaitre à l'avance le nombre de clusters au sein de la population.
Afin de tester l'efficacité de ce type de modèle nous procèderons comme suit : tout d'abord nous détaillerons le fonctionnement explicite du DPMM, puis nous l'implémenterons sur une base de donnée simulée pour tester sa rapidité et procédons à mettre en place des méthodes d'optimisation de la vitesse d'execution via cython et de la parallésation quand l'hypothèse de Bernstein est satisfaite.

1 Présentation formelle

Considérons tout d'abord le cas d'un modèle de mélange fini. Ceci nous permettra d'expliciter les différences avec le cas infini. Nous notons K le nombre total de clusters. Une fois qu'un cluster a été tiré (avec probabilité π_k), les observations sont générées selon une loi normale de moyenne μ_k et de variance σ_k^2. Notons X le vecteur des observations générées selon ce processus.

Dans ce cas, la vraisemblance du modèle est défini comme une combinaison de distributions gaussiennes :

$$p(X|\mu_1,\dots,\mu_K,\sigma_1^2,\dots,\sigma_K^2,\pi_1,\dots,\pi_K) = \sum_{k=1}^{K} \pi_k \mathcal{N}(X;\mu_K,\sigma_k^2)$$

où π_k est la proportion de chaque cluster de sorte que $\sum_k^K \pi_k = 1$.
Considérons c_i la variable catégorielle indiquant le label du cluster auquel appartient l'observation i. Ainsi, c_i prend ses valeurs dans l'ensemble discret $\{1,2,\dots,K\}$. Lorsque $c_i = k$, cela signifie que l'observation i est assignée au cluster k par le modèle. Si k était connu, alors dans ce cas on aurait :

$$p(X_i|c_i = k) = \mathcal{N}(X_i;\mu_K,\sigma_k^2) \tag{1}$$

On peut alors utiliser le théorème de Bayes pour calculer la probabilité jointe à posteriori des paramètres, en particulier des paramètres des clusters et la distribution des clusters. On a :

$$\begin{aligned} p(c,\sigma^2,\mu|X) &= \frac{p(X|\mu,\sigma^2,c)p(\mu,\sigma^2,c)}{p(X)} \\ &\propto p(X|\mu,\sigma^2,c)p(\mu,\sigma^2,c) \\ &\propto p(X|\mu,\sigma^2,c)p(\mu)p(\sigma^2)p(c) \end{aligned}$$

On aura ici noté μ et σ^2 les vecteurs contenant les valeurs des paramètres des différents clusters. Comme habituellement, la partie droite de l'équation dépend de la vraisemblance et de l'équation (1), mais également des lois de probabilité à priori de c, μ et σ^2.
Les lois à posteriori de chacun de ces trois paramètres doivent donc être spécifiés.
On considèrera ici $\forall k \in \{1,\dots,K\}$, $\mu_k \sim \mathcal{N}(\mu_0,\sigma_0^2)$, de même en ce qui concerne le terme de variance σ^2, on considérera une inverse-Wishart comme loi a priori. En effet, dans le cas d'une loi normale univariée, l'inverse-Wishart se simplifie en une loi inverse-Gamma de paramètres γ,β.
Dans ce cadre, là il advient que :

$$\begin{aligned} p(\mu_k|y,c=k) &\propto p(y,c=k|\mu_k)p(\mu_k) &(2)\\ &\sim \mathcal{N}\left(\frac{\bar{y_k}\mu_k\tau_k + \mu_0\tau_0}{n_k\tau_k+\tau_0}, n_k\tau_k+\tau_0\right) &(3)\end{aligned}$$

Où $\bar{y_k} = \frac{1}{n_k}\sum_{i:c_i=k}^{n_k} y_i$, $\tau_k = 1/\sigma_k^2$ et n_k le nombre d'individus dans le cluster k.
De même

$$p(\tau_k|\mu,y,\gamma,\beta) \propto \Gamma\left(\gamma + n_k/2, \beta + \frac{\sum_{i=1}^{n_k}(y_i-\mu_k)^2}{2}\right) \tag{4}$$

En ce qui concerne les poids des clusters π_k, la loi a priori sera une Dirichlet symétrique de paramètre α/K,

$$\begin{aligned} p(\pi_1, \ldots, \pi_K) &\sim \text{Dirichlet}(\alpha/K, \ldots, \alpha/K) \quad (5) \\ &= \frac{\Gamma(\alpha)}{\Gamma(\alpha/K)^K} \prod_{k=1}^{K} \pi_K^{\alpha/K-1} \quad (6) \end{aligned}$$

Notons également que la distributions a priori de $c_1, \ldots, c_k$ est une multinomiale dont le vecteur de probabilités $(\pi_1, \ldots, \pi_k)$ correspond aux proportions des clusters (ou de manière équivalente à leur probabilité d'appartenance respective). La densité *a priori* de c_k s'écrit donc :

$$p(c_1, \ldots, c_k | \pi_1, \ldots, \pi_k) = \prod_{k=1}^{K} \pi_k^{n_k} \quad (7)$$

Dès lors, dans ce cas où le nombre de cluster est connu à l'avance, peut déduire **l'algorithme de Gibbs** suivant (en notant $\bar{y_k} = \frac{1}{n_k} \sum_{i:c_i=k}^{n_k} y_i$) :

$$\begin{aligned} \pi_k^{(t)} &\sim \text{Dirichlet}(n_1 + \alpha/K - 1, \ldots, n_k + \alpha/K - 1) \\ \mu_k^{(t)} | c^{(t-1)}, y, \tau^{(t-1)} &\sim \mathcal{N}\left(\frac{\bar{y_k} n_k \tau_k^{(t-1)} + \mu_0 \tau_0}{n_k \tau_k^{(t-1)} + \tau_0}, n_k \tau_k^{(t-1)} + \tau_0 \right) \\ \tau_k^{(t-1)} | c^{(t-1)}, y, \mu_k^{(t-1)} &\sim \Gamma\left(\alpha + n_k/2, \beta + \frac{1}{2} \sum_{i:c_i=k}^{n_k} (y_i - \bar{y_k})^2 \right) \\ c_i^{(t)} \mid y, \mu^k, \tau^k &\sim \text{Multinomial}\left(1, \pi_1^{(t)} \mathcal{N}(y_i | \mu_1^{(t)}, \tau_1^{(t)}), \ldots, \pi_k^{(t)} \mathcal{N}(y_i | \mu_k^{(t)}, \tau_k^{(t)}) \right) \end{aligned}$$

Cet algorithme ne fonctionne que dans le cas où K est connu et fixé au préalable. Dans le cas où le nombre de clusters n'est pas borné comme nous allons le mettre en place, nous devons considérer un vecteur π infini. Pour contourner ce problème, Ferguson (1973) propose un *a priori* qui ne dépend pas de π.
Si π le vecteur des proportions des clusters est de taille infinie, alors c en tant que fonction de π (7) et π fonction de α (6), on peut obtenir par Rasmussen (2000) :

$$p(c_1, ..., c_k | \alpha) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + n)} \prod_{k=1}^{K} \frac{\Gamma(n_k + \alpha/K)}{\Gamma(\alpha/K)} \quad (8)$$

De plus, on en notant c_{-i} comme le vecteur des c_j pour tout $j \in \{1, ..., i-1, i+1, ...n\}$, *i.e.* le vecteur comprenant toutes les valeurs de c_j excepté pour l'observation c_i, de même $n_{-i,k} = \sum_{k:k \neq i}^{K} n_k$ l'ensemble des invidivus n'appartenant pas au cluster i, on a

$$p(c_i = k | c_{-i}, \alpha) = \frac{n_{-i,k} + \alpha/K}{n - 1 + \alpha} \quad (9)$$

Une première propriété intéressante de cette équation est que chaque point est interchangeable en ce qu'il peut être considéré comme le dernier point arrivé. Si on considère $K \to \infty$ dans cette dernière équation, il advient alors que pour les clusters $n_{-i,k} > 0$

$$p(c_i = k | c_{-i}, \alpha) = \frac{n_{-i,k}}{n - 1 + \alpha}$$

On remarque ici que si on somme les probabilités pour tous les clusters existants, *i.e.* tels que $n_{-i,k} > 0$, on obtient $\frac{n-1}{n-1+\alpha} \neq 1$, ceci permet alors de déterminer quelle est la probabilité de créer un nouveau cluster à chaque étape :

$$1 - \frac{n-1}{n-1+\alpha} = \frac{\alpha}{n-1+\alpha}$$

Ceci permet donc de comprendre l'importance du choix de α en ce qui concerne les paramètres, puisque l'apparition d'un nouveau cluster à chaque étape est proportionnelle à ce paramètre. Plus ce paramètre, souvent appelé paramètre de concentration, est élevé plus la probabilité d'apparition d'un nouveau cluster est élevée. Tandis que la probabilité que le nouveau point soit assigné à un cluster existant k est proportionnelle à la taille de ce cluster $n_{-i,k}$. Finalement on a donc

$$\begin{aligned} \forall k \text{ tq } n_{-i,k} \neq 0 : p(c_i = k | c_{-i}, \alpha) &= \frac{n_{-i,k}}{n-1+\alpha} \\ p(c_i \neq c_j \forall j \neq i | c_{-i}, \alpha) &= \frac{\alpha}{n-1+\alpha} \end{aligned} \tag{10}$$

Enfin, la distribution *a priori* des proportions suit une distribution Griffiths–Engen–McCloskey. On a $\pi = (\pi_1, \pi_2, ...) \sim GEM(\alpha)$, $\alpha > 0$.

Introduisons ici le concept bayésien de *posterior predictive distribution*, qui est simplement la distribution a posteriori d'une nouvelle observation compte tenu de l'information obtenues des données déjà observée. On note la nouvelle observation $\tilde{y_i}$.
Intuitivement, la probabilité que la nouvelle observation appartienne à un cluster est d'autant plus élevée que cette nouvelle observation est proche de celui-ci. Les distributions décrites en équation (10) sont conditionnelles à tous les clusters sauf précisemment celui de y_i. Il faut donc introduire la *posterior predictive distribution* conditionnelle à tous les clusters.

Supposons que $y \sim \mathcal{N}(\mu, \sigma_y^2)$, $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$, on obtient alors la distribution prédictive a posteriori est la suivante.

$$\tilde{y} \sim \mathcal{N}(\mu_0, \sigma_y^2 + \sigma_0^2)$$

Par conséquent, dans le cas présenté jusqu'ici on obtient en utilisant le résultat en equation (3), en notant $\mu_p = \frac{\bar{y_i} n_k \tau_k + \mu_0 \tau_0}{n_k \tau_k + \tau_0}$ et $\tau_p = n_k \tau_k + \tau_0$

$$\begin{aligned} \tilde{y} &\sim \mathcal{N}(\mu_p, \sigma_p^2 + \sigma_0^2) \\ &\sim \mathcal{N}(\frac{\bar{y_i} n_k \tau_k + \mu_0 \tau_0}{n_k \tau_k + \tau_0}, \frac{1}{n_k \tau_k + \tau_0} + \sigma_y^2) \end{aligned}$$

Ici nous avons fait l'hypothèse que σ_y^2, l'erreur de mesure, est connue et est la même pour chacun des clusters.

Introduisons également la distribution prédictive a priori. Considérons ici une variable y tel que $y \sim \mathcal{N}(\mu, \sigma_y^2)$, de même en considérant toujours $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$ où les paramètres a priori μ_0 et σ_0^2 représentent l'incertitude dans les données. Dans ce cadre, Novick and Jackson (1974) montrent que la distribution prédictive a priori est une loi normale de paramètre μ_0 et $\sigma_y^2 + \sigma_0^2$. Cette notion de distribution predictive a priori est nécessaire, en effet, avant qu'un cluster ne soit réalisé sa moyenne et précision (*i.e.* l'inverse de la variance) ne

sont pas connues. Dès lors la vraisemblance d'une nouvelle observation $\tilde{y}$ doit être estimée compte tenu de la distribution prédictive a priori.

Nous pouvons désormais calculer la distribution conditionnelle de c_i a posteriori. Pour les clusters déjà existant, *i.e.* k tels que $n_{-i,k} > 0$, on a que

$$\begin{aligned} p(c_i|c_{-i}, \mu_k, \tau_k, \alpha) &\propto p(c_i|c_{-i}, \alpha) p(\tilde{y}_i|\mu_k, \tau_{k,-i}) \\ &\propto \frac{n_{-i,k}}{n-1+\alpha} \mathcal{N}\left(\tilde{y}_i, \frac{\bar{y}_i n_k \tau_k + \mu_0 \tau_0}{n_k \tau_k + \tau_0}, \frac{1}{n_k \tau_k + \tau_0} + \sigma_y^2\right) \end{aligned}$$

et pour tous les autres clusters

$$\begin{aligned} p(c_i \neq c_j \forall j \neq i|c_{-i}, \mu_0, \tau_0, \alpha) &\propto p(c_i \neq c_j \forall j \neq i|c_{-i}, \alpha) \\ &\quad \times \int p(\tilde{y}_i|\mu_k, \tau_k) p(\mu_k, \tau_k|\mu_0, \tau_0) \mathrm{d}\mu_k \mathrm{d}\tau_k \\ &\propto \frac{\alpha}{\alpha + n - 1} \mathcal{N}\left(\tilde{y}_i; \mu_0, \sigma_0^2 + \sigma_y^2\right) \end{aligned}$$

Ceci nous permet alors de mettre en place un algorithme itératif présenté dans la section suivante.

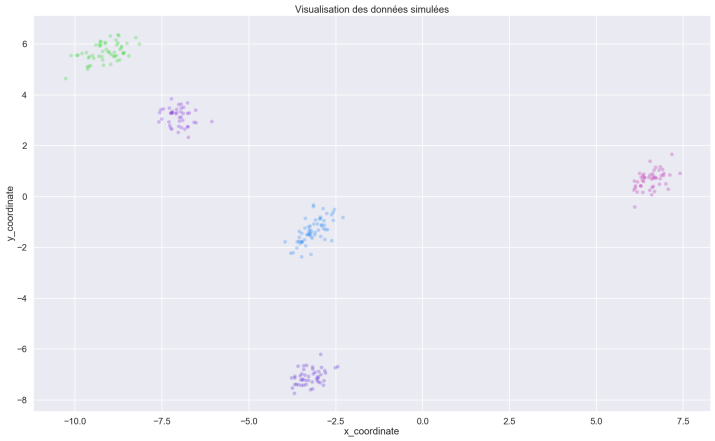
2 Algorithme et optimisation

La méthode de clusterisation décrite ci-dessus a été codée en python. L'algorithme utilisé est le suivant :

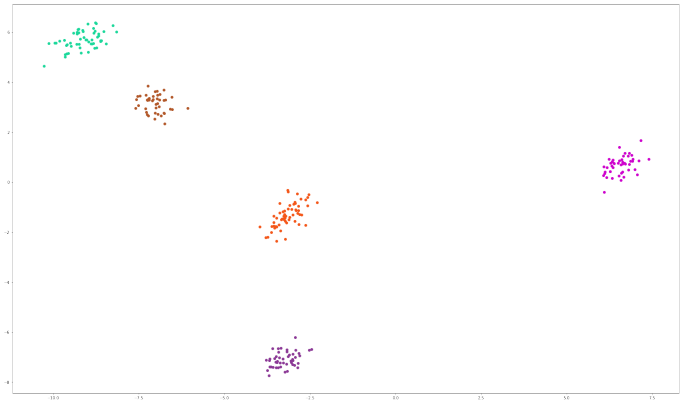
Algorithm 1 Algorithme Dirichlet Process Mixture Model

Input: α, μ_0, σ_0^2, σ_x^2, $\tau_0 = 1/\sigma_0^2$
Output: c = $\mathrm{c}^{(n_{iter})}$
for t from 1 to n_{iter} **do**
  for i from 1 to n **do**
    retirer x_i de son cluster, si le cluster devient vide alors retirer le cluster
    for k from 1 to $N_{cluster}$ **do**
      $p(c_i = k \mid c_{-i}, \mu_p, \tau_p, \alpha) \propto \frac{n_{-i,k}}{n-1+\alpha} \times \mathcal{N}(\tilde{x_i}; \mu_p, \sigma_p^2 + \sigma_x^2)$
      $p(c_j \neq c_k \forall j \neq i \mid c_{-i}, \tau_0, \mu_0, \alpha) \propto \frac{\alpha}{n-1+\alpha} \times \mathcal{N}(\tilde{x_j}; \mu_0, \sigma_0^2 + \sigma_x^2)$
      if $c_i = k$ pour $j \neq i$ **then**
        Update $\bar{x}_k$, n_k, τ_k.
      else
        $c_i = N_{cluster} + 1$ un nouveau cluster a été créé, on ajoute ce cluster au vecteur c des clusters
      end if
    end for
  end for
  $c^{(t)} = c_i$
end for

Ce dernier, par essence itératif, est malheureusement très lent à tourner même s'il donne des résultats plus que satisfaisant en ce qui concerne la prédiction des clusters. En effet, lorsque les données sont séparées, l'algorithme DPMM parvient à prédire avec une accuracy de 100% les clusters d'appartenance de données simulées (voir Figure 1).



(a) Données simulées avec le vrai cluster



(b) Données simulées avec les prédictions de l'algorithme

FIGURE 1 – Données simulées et prédictions

Note : Notre algorithme prédit avec une précision de 100% l'appartenance aux clusters lorsque les données sont séparées

L'objectif de ce projet a donc été d'améliorer la vitesse d'execution de l'algorithme via deux types de solutions : cython et la parallélisation. Pour bien comprendre les enjeux, et structurer l'amélioration des performances nous avons fonctionnalisé notre algorithme : nous avons créé des fonctions externes auquel fait appel notre algorithme. Les Figure 4 et Figure 5 permettent de comprendre les changements opérés. Ces changements ne permettent en rien d'améliorer la performance de l'algorithme mais permettent de comprendre les différents appels aux fonctions réalisés lors des différentes itérations.
Notons ici que le nombre de fonctions appelées décrites dans les différents graphique en Figure 4 à 7 le sont pour un dataset de taille 250 de $X \in \mathbb{R}^2$, et 1000 itérations de l'algorithme de Gibbs sous-jacent au DPMM.

Nous avons donc créé trois fonctions utilisées par l'algorithme décrit ci-avant :

- `log_probability` : fonction qui transforme un vecteur en vecteur de probabilité via une transformation Log-Sum-Exp.
- `compute_not_cluster_log_prob` : qui calcule la probabilité de pas appartenir à un

cluster existant
— `convert_into_probabilities` : qui calcule la probabilité d'appartenir à chacun des clusters

Il vient alors que la fonction `log_probability` est la fonction la plus appelée parmi les 3 fonctions crées et également la plus coûteuse en ce qu'elle fait appel à de nombreuses reprises aux fonctions `numpy.dot()` et `numpy.inv()`. Il est donc important d'accélérer sa vitesse d'execution. Même si moins utilisées les fonctions `compute_not_cluster_log_prob` et `convert_into_probabilities` ont également été cythonisées afin de réduire le temps global d'execution.

Les Figures 6 et 7 décrivent les appels à des fonctions pythons de la part de notre algorithme une fois cythonisé et parallélisé. Il apparait clairement que, du fait de la cythonisation des fonctions précédemment citées, le nombre de fonctions python utilisées à été drastiquement réduit.
Par ailleurs, nous avons également recouru à de la parallélisation quand cela était possible. Notamment, lorsque l'algorithme calcule une à une les probabilités d'appartenance aux clusters, nous avons pu paralléliser cette étape et la scinder en deux calculs parallèles venant remplir un même vecteur de probabilité. En effet, il n'est pas nécessaire de connaître les probabilités d'appartenance aux autres clusters pour calculer la probabilité d'appartenir à un cluster.

La Figure 2 décrit le temps moyen d'execution (sur 10 executions) en fonction du nombre

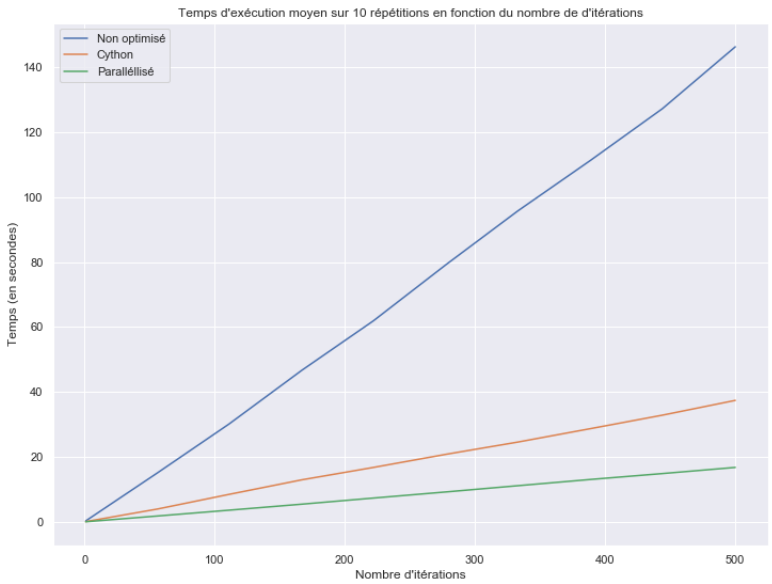


FIGURE 2 – Comparaison des performances en fonction du nombre d'itérations

d'itération de trois algorithmes :
— L'algorithme non optimisé
— L'algorithme avec les fonction cythonisée
— L'algorithme avec les fonctions cythonisées et parallélisé

On observe tout d'abord qu'il y a une nette amélioration grâce à cython puisque qu'on

passe d'un temps d'execution pour 500 itérations de l'ordre de 140 secondes pour le modèle non optimisé à 40 secondes pour le modèle cythonisé. Le gain est donc plus que signigicatif. De même, La parallélisation semble apporter un autre gain significatif puisque le temps d'execution passe à moins de 20 secondes. En ce qui concerne la figure 3 on observe le même

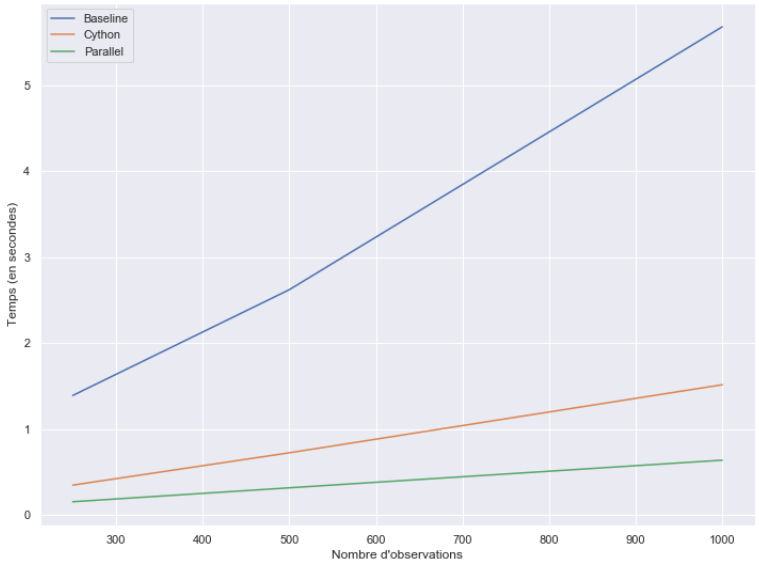


FIGURE 3 – Comparaison des performances en fonction du nombre d'observations

type d'évolution entre les trois algorithmes confirmant bien l'amélioration significative du temps d'execution.

Annexes

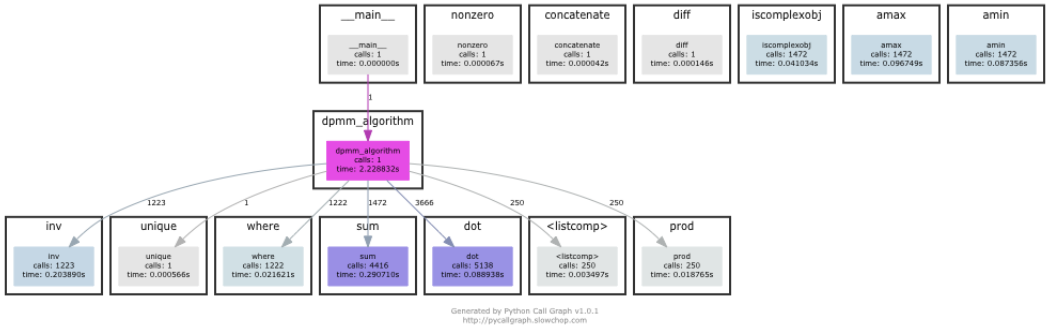


FIGURE 4 – Baseline Algorithme

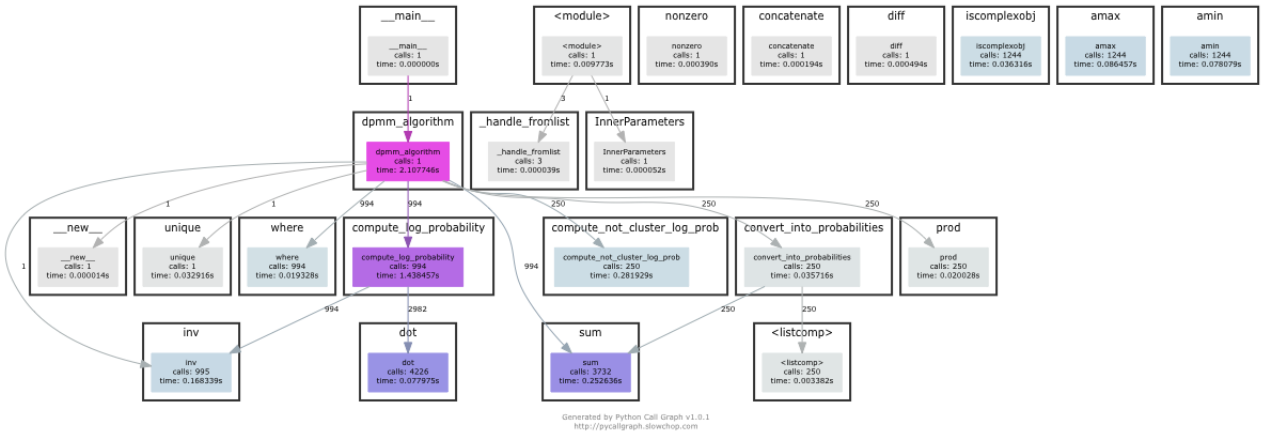


FIGURE 5 – Détails Algorithme Fonctionnalisé

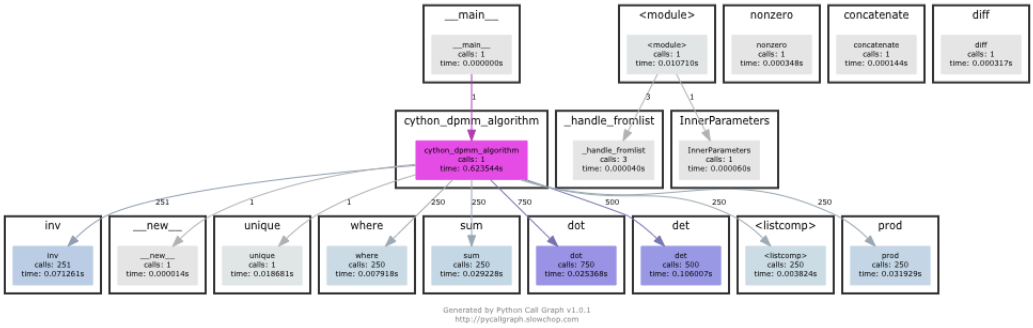


FIGURE 6 – Détails fonction Algorithme Cythonisé

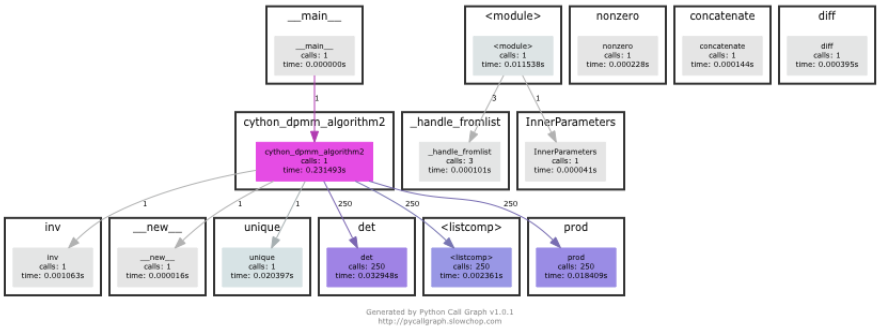


FIGURE 7 – Détails fonction Algorithme Cythonisé et parallélisé