Analyse des solutions proposées pour l'épreuve grandes données

Ce rapport a pour but d'analyser les solutions proposées pour l'épreuve grande données. Il s'agira d'évaluer la performance des solutions afin de choisir celle qui sera la plus adaptée à notre problème.

Hassan BERRADA
Élève-Ingénieur en deuxième année à l'École des Mines de Nancy

Sommaire

Introduction

Les solutions proposées sont sous la forme de fonctions prenant en entrée le dataset *df* et retournant le dataset sous le format indiqué dans le sujet, c'est à dire avec une première colonne A représentant les états et une colonne B comptant le nombre de fois où un même état est resté invariant.

Les fonctions proposées sont sous la forme :

count_state_changes_caractéristique_version(df)

1 Analyse des fonctions de l'épreuve en une heure

1.1 Boucle for

La première solution proposée met en place une boucle for et un compteur. Cette boucle for va itérer sur les éléments du tableau en gardant en mémoire l'élément précédant. A chaque tour de boucle, cet élément est mis à jour s'il est différent de l'élément précédant et le compteur est remis à 0, sinon, le compteur est incrémenté de 1.

En général, il est déconseillé d'appliquer des boucles for sur un jeu de données et il est préférable de vectoriser le calcul. La difficulté ici est que le traitement est difficilement vectorisable, car la valeur du compteur à un instant t donné dépends de celle à l'instant $t-1$.

Nous avons donc décidé dans un premier temps d'implémenter cette solution simple qui nous permettra par la suite de la comparer à d'autres solutions.

1.1.1 Implémentation avec iterrows

Fonction décrite : count_state_changes_for_iterrows(df)

La première version proposée implémente une boucle for à l'aide de la méthode pandas *iterrows*. Celle-ci n'écrit pas évidemment les valeurs de retour directement sur le dataframe car il est déconseillé par pandas de modifier un dataframe ligne par ligne. A la place, nous construisons la liste des compteurs d'état que l'on intégrera dans le dataframe initial. Cette solution se révèle très vite chronophage et peu prendre plus d'une quinzaine de minutes à être exécutée.

1.1.2 Implémentation avec iloc

Fonction décrite : count_state_changes_for_index(df)

La première amélioration sur cette fonction consiste simplement à remplacer la méthode d'itération en utilisant iloc, privilégié pour itérer sur un dataframe, et plus spécifiquement sur une colonne du dataframe

1.1.3 Comparaison des deux méthodes

La première comparaison ci-dessous des deux méthodes, nous permet de sélectionner la deuxième méthode beaucoup plus efficace. Nous l'utiliserons pour comparer les nouvelles solutions que l'on produira par la suite.

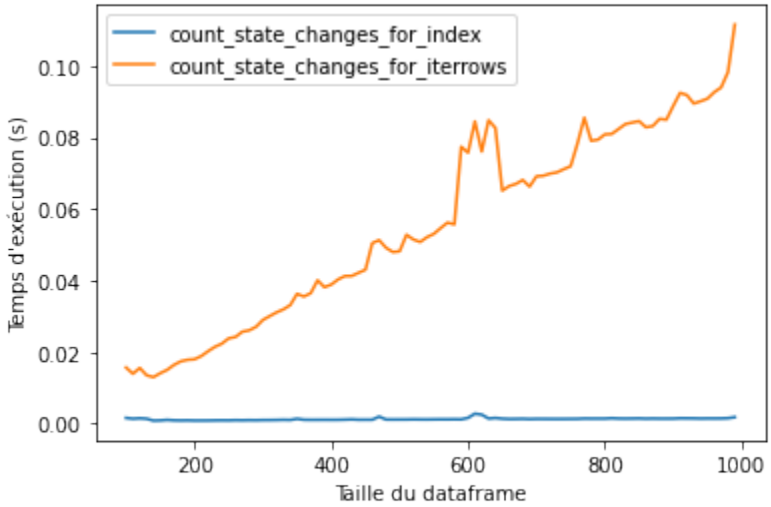

FIGURE 1 – Comparaison des deux fonctions sur un dataframe de taille 100 à 1000

1.2 Group By

Fonction décrite : count_state_changes_groupby(df)

Afin d'utiliser le potentiel de pandas, l'idéal serait d'arriver à vectoriser les opérations qui sont exécutées. Une des premières façon de faire, serait de grouper les états consécutifs, de récupérer la longueur l de chaque groupe et de créer les listes dont les éléments vont de 0 à $l-1$. Enfin, il faudrait les concaténer et les insérer dans le tableau initial.

Toutefois, si l'on applique directement la méthode groupby sur le dataframe, nous obtiendrons deux groupes consitués de 0 et de 1. Il faut donc trouver une manière d'identifier ces groupes de manière unique.

Pour cela, on construit le tableau des différences logiques entre df et le décalage de df d'une ligne. Chaque changement d'état est donc identifié par un 1 dans ce nouveau tableau et lorsque l'on construit les sommes cumulatives de ce tableau, on obtient un tableau indexant de manière unique les groupes d'états.

Voici un exemple de ce type de tableau :

	A	C	D
0	1	True	1
1	1	False	1
2	0	True	2
3	1	True	3
4	0	True	4
5	1	True	5
6	0	True	6
7	1	True	7
8	1	False	7
9	0	True	8

FIGURE 2 – Tableau des différences entre les états successifs et somme cumulatives

1.2.1 Comparaison de l'implémentation avec Group By et l'implémentation avec une boucle For

Une exécution sur des dataframes de taille comprise entre 1000 à 10 000, nous montre que la méthode utilisant une boucle for reste plus efficace. Toutefois, nous n'avons pas pu à l'issu de cette épreuve d'une heure optimiser la fonction utilisant Group By. Les temps d'exécution de ces deux méthodes sur un dataframe de 10 000 000 prends une dizaine de secondes, ce qui était satisfaisant pour le moment. Le temps d'exécution de la fonction utilisant groupby étant de 15s environ et celui de la fonction utilisant la boucle for étant de 10s, nous choisissons donc la fonction *count_state_changes_for_index(df)*.

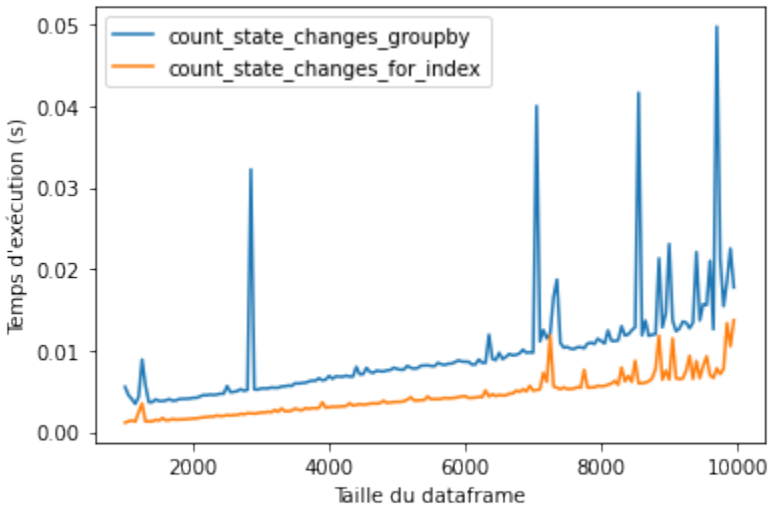

FIGURE 3 – Comparaison des méthodes avec une boucle for et avec groupby

2 Analyse des fonctions de l'épreuve étendue

Dans cette partie, nous verrons des pistes d'amélioration de la fonction utilisant *groupby*.

```
def count_state_changes_groupby_v1(df):

    states_change = df['A'] != df['A'].shift(1)
    group_indexes = states_change.cumsum()
    groups = df['A'].groupby(group_indexes)

    state_changes_count = groups.count().apply(
    lambda x: list(range(x))
    ).explode()

    df["B"] = state_changes_count.reset_index(drop=True)

    return df
```

En mesurant le temps pris par la fonction dans chaque étape du calcul, on remarque que les lignes 1 à 3 et 5 sont exécutées relativement rapidement par rapport à la ligne 4. Sur un dataframe de taille 10 000 000, les lignes 1 à 3 sont exécutés en 0.15s, la ligne 5 en 0.36s alors que la ligne 4 est exécutée en 13s.

Ainsi, la récupération de l'information utile (ligne 1 à 3) prends nettement moins de temps que le traitement de cette information (ligne 4). Cela veut dire que cette étape doit être optimisée car elle est un goulot d'étranglement pour l'exécution de l'algorithme.

2.1 Passage par une liste temporaire

Fonctions décrites :
count_state_changes_groupby_v2(df)
count_state_changes_groupby_v3(df)

Les deux fonctions suivantes construisent une liste temporaire en concaténant les sous-listes suivante : [0, ... , taille(groupe)[.

La première fonction implémente une boucle for et la deuxième importe l'opérateur de concaténation de liste de python.

Ces deux fonctions s'exécutent toutefois en temps équivalent.

Elles s'exécutent un peu plus rapidement que la fonction précédente car le traitement des données prends moins de temps. Toutefois, elles ne s'exécutent pas plus rapidement que la fonction count_state_changes_for_index car la création du dataframe à partir de la liste des résultats prends un temps plus considérable par rapport à la solution précédente car celle-ci ne nécessite pas la création d'une colonne dans le tableau.

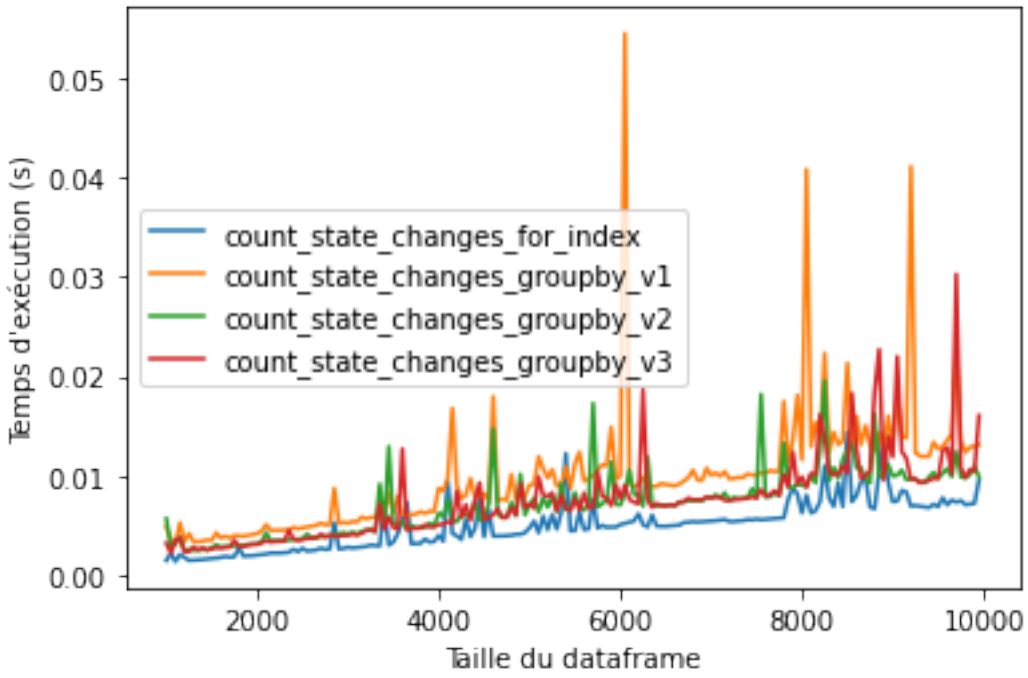

FIGURE 4 – Comparaison de performances de toutes les fonctions

Conclusion

La fonction étant la plus adaptée à l'issu de l'épreuve reste celle implémentant une boucle for. Le principal obstacle à l'utilisation d'une méthode comme celle utilisant groupby provient de l'exécution coûteuse de la méthode apply (ou transform) sur une dataframe en utilisant une fonction lambda qui associe à chaque taille de groupe une liste. De plus, le fait d'utiliser cette fonction sur des groupes de taille 2 en moyenne (un changement d'état s'effectuant une fois sur deux) fait que l'exécution de la fonction devient beaucoup plus lente.

Globalement les résultats s'exécutent en une dizaine de secondes, ce qui peut être optimisé par la suite.