Clustering prototypique de séries temporelles

Baptiste GAUTIER

Ce travail, présente un algorithme de clustering de séries temporelles que j'ai mis au point. Plus précisément, cet algorithme est une modification de l'algorithme du *KMeans* afin d'appliquer ce dernier au cadre temporel. Les données utilisées pour cet exemple sont des données relatives à la demande d'électricité sur une périodicité horaire. Ces données sont disponibles sur Kaggle en cliquant ici. Le github associé au projet, comprenant le notebook de présentation et le script sont disponibles ici.

Table des matières

Présentation du problème et des données

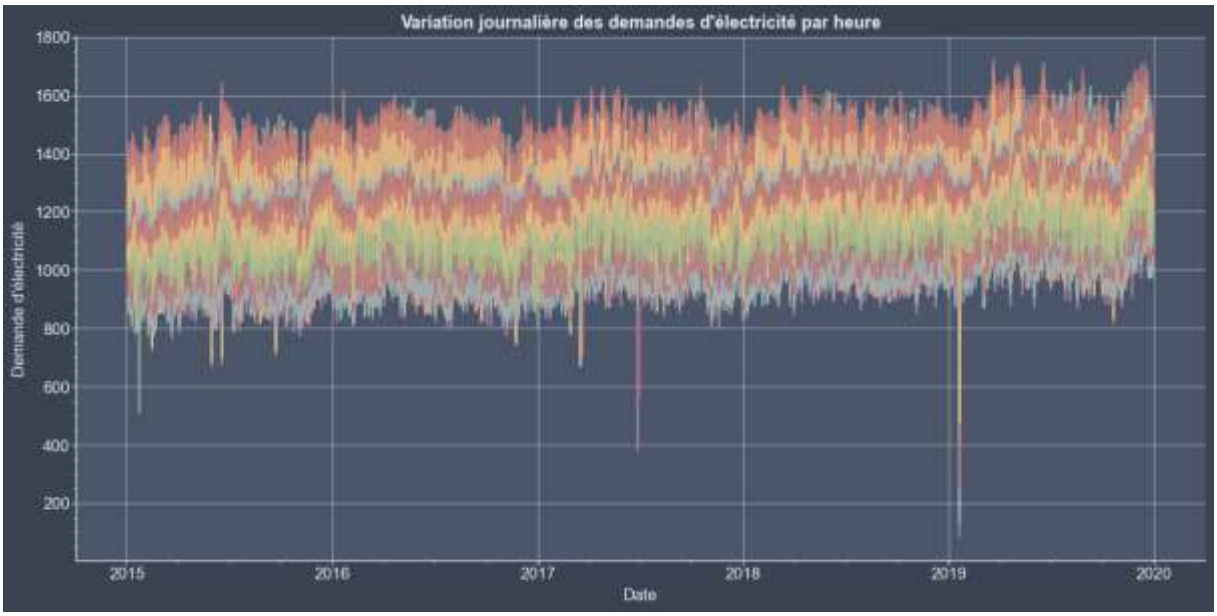
Imaginez que vous disposez de plusieurs séries temporelles définies sur un même domaine. Un exemple typique peut être un ensemble de séries relatives à la productivité journalière de salariés ou de machines. Un autre exemple peut être le montant total acheté par les différents clients d'un magasin sur une périodicité hebdomadaire, mensuelle ou annuelle. Dans notre exemple, nous étudions une demande en électricité journalière. Nos différentes séries correspondent à la demande aux différentes heures de la journée (soit 23 séries de 1h à 23h).

Le but de l'algorithme développé ici consiste à associer chaque série à un prototype, un *comportement moyen*. Dans les exemples cités précédemment, l'usage d'un tel algorithme pourrait être de segmenter automatiquement les salariés, machines ou clients étudiés en différentes populations types. On pourrait avoir les « gros producteurs » face aux « plus petit producteurs » ou bien les clients « très rentables » face aux « moins rentables ». En termes d'analyse de demande, cela permet automatiquement de classer les différentes heures de la journée comme heure plus ou moins creuse.

Un autre aspect pouvant s'avérer intéressant est qu'une fois les prototypes établis, il est possible de réaliser une prévision de ces derniers pour estimer la tendance que chaque groupe prendra à l'avenir.

Enfin, cet algorithme peut également permettre d'identifier les anomalies, si la distance d'une série au prototype associé est trop importante ; et peut donc amener à des actions incitatives ou préventives.

Pour illustrer mon propos, voici l'évolution de la demande pour les différentes heures de la journée :



Le graphique apparait peu lisible. De fait, nous venons d'afficher 23 courbes. La moyenne de nos séries semble relativement constante sur le temps, on pourrait donc classer chaque série en se basant sur cette dernière. Toutefois dans des cas non-stationnaires, cette propriété ne vaudrait plus. La solution proposée ci-après permet une robustesse à cette problématique. De plus, elle évite d'avoir à réaliser l'analyse manuelle des valeurs. Dans notre cas, ce dernier point ne constitue pas une véritable tare, cependant lorsque le nombre de séries temporelles augmente, ce dernier aspect peut s'avérer fastidieux. Sans plus attendre, présentons donc le KMeans – la version originale de l'algorithme utilisé que j'ai adapté aux séries temporelles.

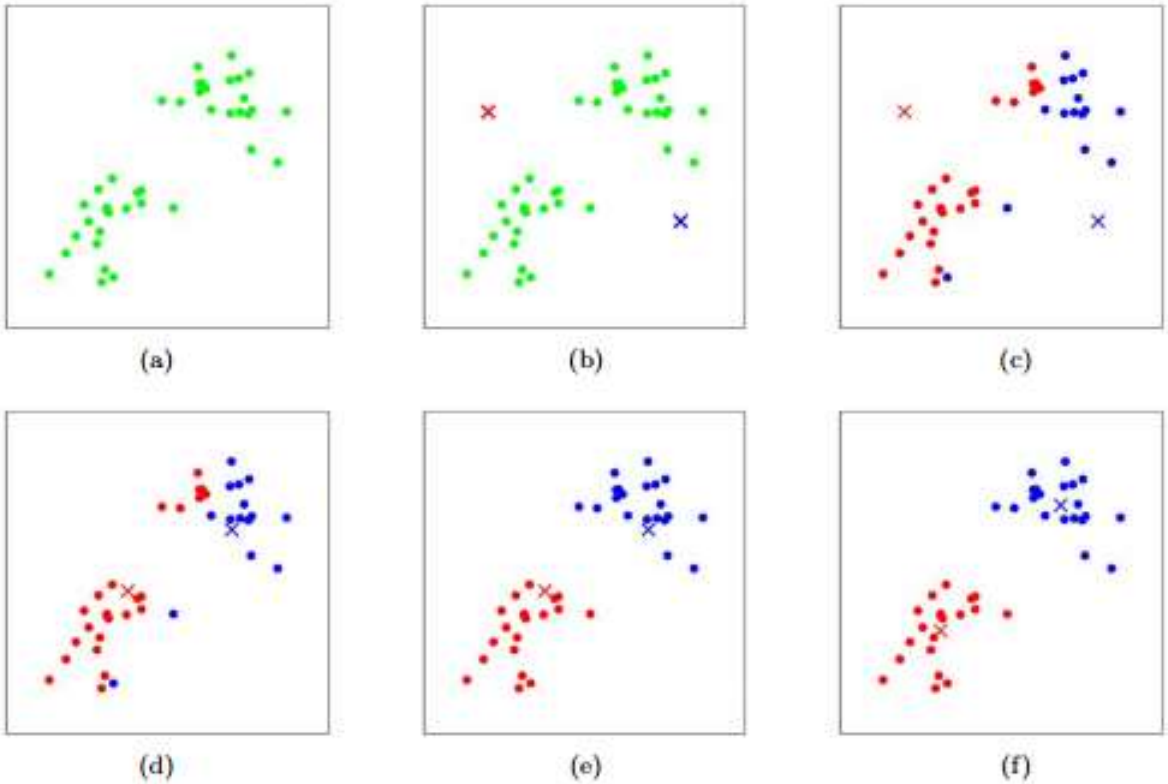
Le clustering par le KMeans

Le KMeans est un algorithme simple de clustering de données tabulaires. Il présente le gros avantage de générer des prototypes représentatifs de chaque cluster.

Son fonctionnement est assez intuitif : On place des points (des prototypes couramment nommés *centroïds*) aléatoirement dans l'espace dans lequel vivent nos points (autant de points que l'on veut de clusters). Ces points seront nos « comportements moyens ». Ensuite, on calcule la distance euclidienne entre les points de notre jeu de données et nos centroïds. Pour rappel, la distance euclidienne entre deux points *x* et *y* de longueur *n* est définie par le théorème de Pythagore comme :

$$D_E = ||\vec{x} - \vec{y}||^2 = \sqrt[2]{\sum_{i=1}^{n}(x_i - y_i)^2}$$

On assigne ensuite à chaque point le centroïd qui en est le plus proche, c'est-à-dire dont la distance euclidienne est la moins grande. Une fois cela fait, on déplace chaque centroïd sur la coordonnée moyenne des points qui lui sont associés. On répète ensuite l'étape d'assignation, puis celle de déplacement du centroïd jusqu'à atteindre la convergence ou un nombre d'itérations maximum. Voici une description graphique de l'algorithme explicitant l'algorithme décrit plus haut :



A noter que les centroïds sont initialement placés aléatoirement. On peut adopter une stratégie pour réduire la part d'aléatoire, cependant entre deux itérations, les résultats du KMeans peuvent varier. Si l'on veut palier à cet effet aléatoire, une stratégie (adoptée par *sklearn*) peut être de relancer plusieurs fois l'algorithme et de prendre comme centroïd final la moyenne des centroïds définis lors des différentes itérations.

Pour une définition mathématique, après avoir initialisé *k* centroïds aléatoirement, on assigne chaque point x_i au centroïd *p* le plus proche selon la règle suivante :

$$r_{i,k} = \begin{cases} 1 \; si \; k = argmin_j \left\| x_i - p_j \right\|^2 \\ 0 \; sinon \end{cases}$$

Où $r_{i,k}$ est une variable binaire indicatrice, indiquant 1 si l'observation *i* est assigné au cluster *k* et 0 sinon.

Une fois cela fait, on déplace chaque prototype à la moyenne des points qui leurs sont assignés :

$$p_k = \frac{\sum_i r_{i,k} x_i}{\sum_i r_{i,k}}$$

Où le dénominateur correspond simplement au nombre de points assignés au cluster *k* et le nominateur à la somme des points assignés au cluster *i* ; le prototype devient donc la moyenne des points qui lui sont assignés.

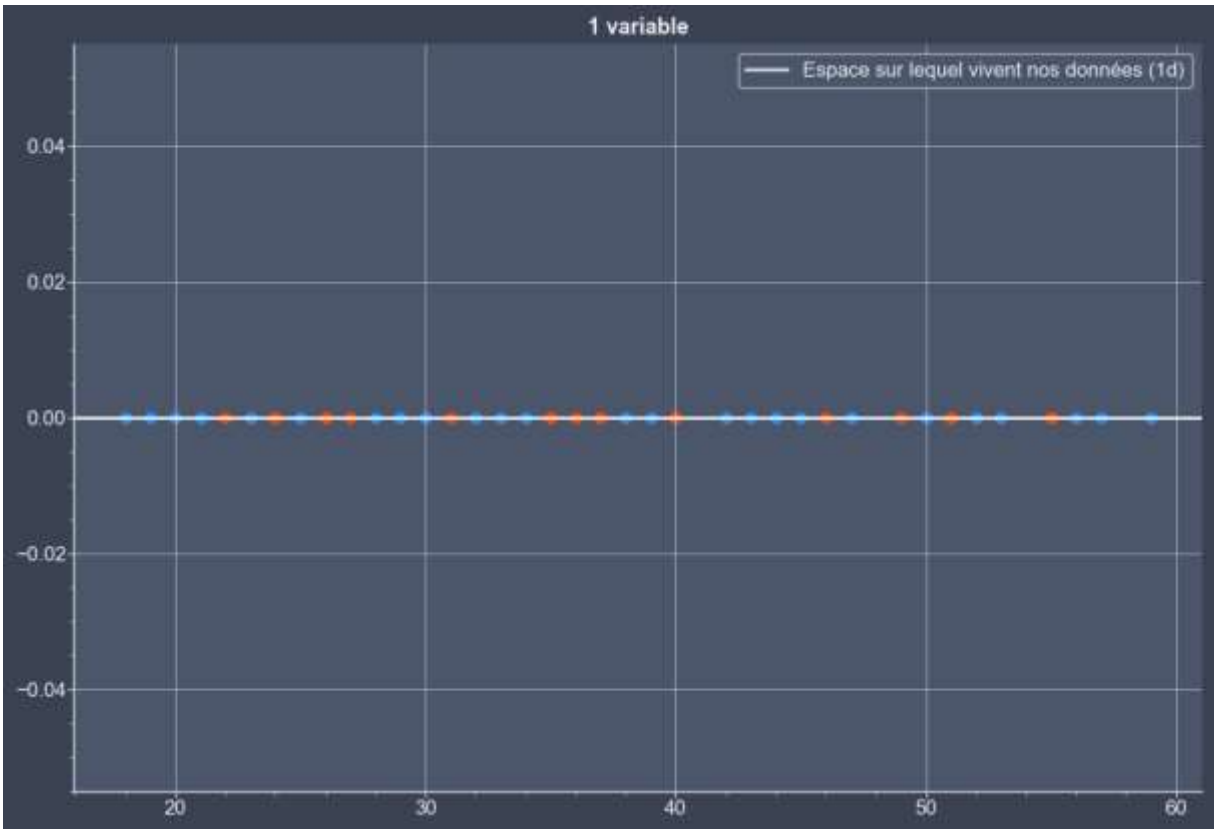
Le problème du KMeans dans le cas temporel

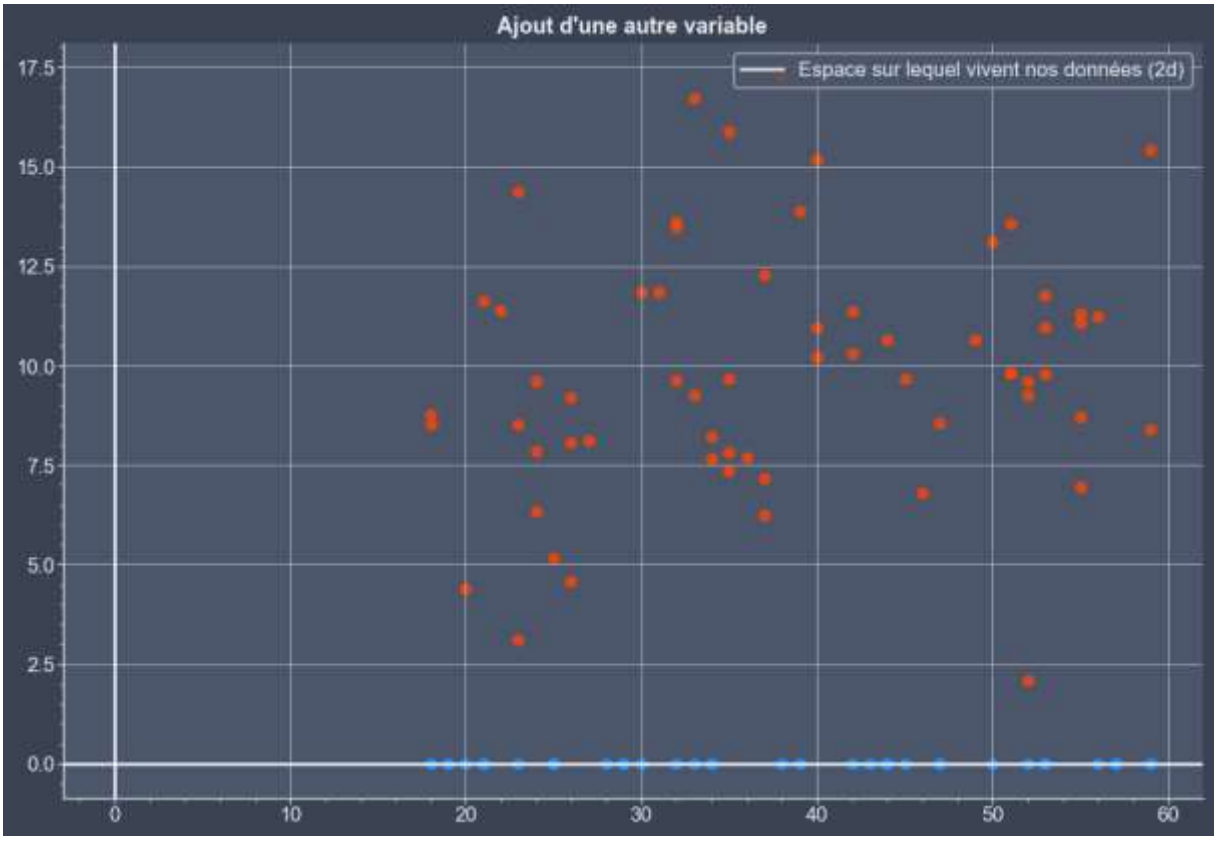
Intuition

Nous l'avons vu dans l'exemple présenté précédemment, le KMeans a été conçu dans un cadre ou les données à disposition étaient des données tabulaires, le cas le plus courant lorsque l'on dispose d'un tableau de données. Voici un exemple fictif de ce type de données :

Age	Années d'expérience	Diplôme	Salaire
25	7	BAC	1800 €
40	15	MASTER	3000 €
50	25	INGENIEUR	4000 €
…	…	…	…

Dans ce cas, les lignes correspondent à une observation, à un cas bien spécifique. Chaque colonne apporte une information sur une observation donnée et permet dans une certaine mesure de distinguer une information des autres. Voici un exemple graphique de l'ajout d'une colonne pour segmenter des données :


1 variable
Espace sur lequel vivent nos données (1d)
0.04
0.02
0.00
−0.02
−0.04
20
30
40
50
60


Ajout d'une autre variable
Espace sur lequel vivent nos données (2d)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
10
20
30
40
50
60

Le KMeans se base sur le même principe, il propose une approche mathématique pour segmenter les observations en fonctions des différentes colonnes.

Nous l'avons dit, le KMeans est un algorithme ingénieux, mais le bât blesse pour les séries chronologiques. En effet une série chronologique est une série représentant les différentes valeurs d'une modalité évoluant sur le temps. Le tableau de données associé ressemble à quelque chose comme ceci :

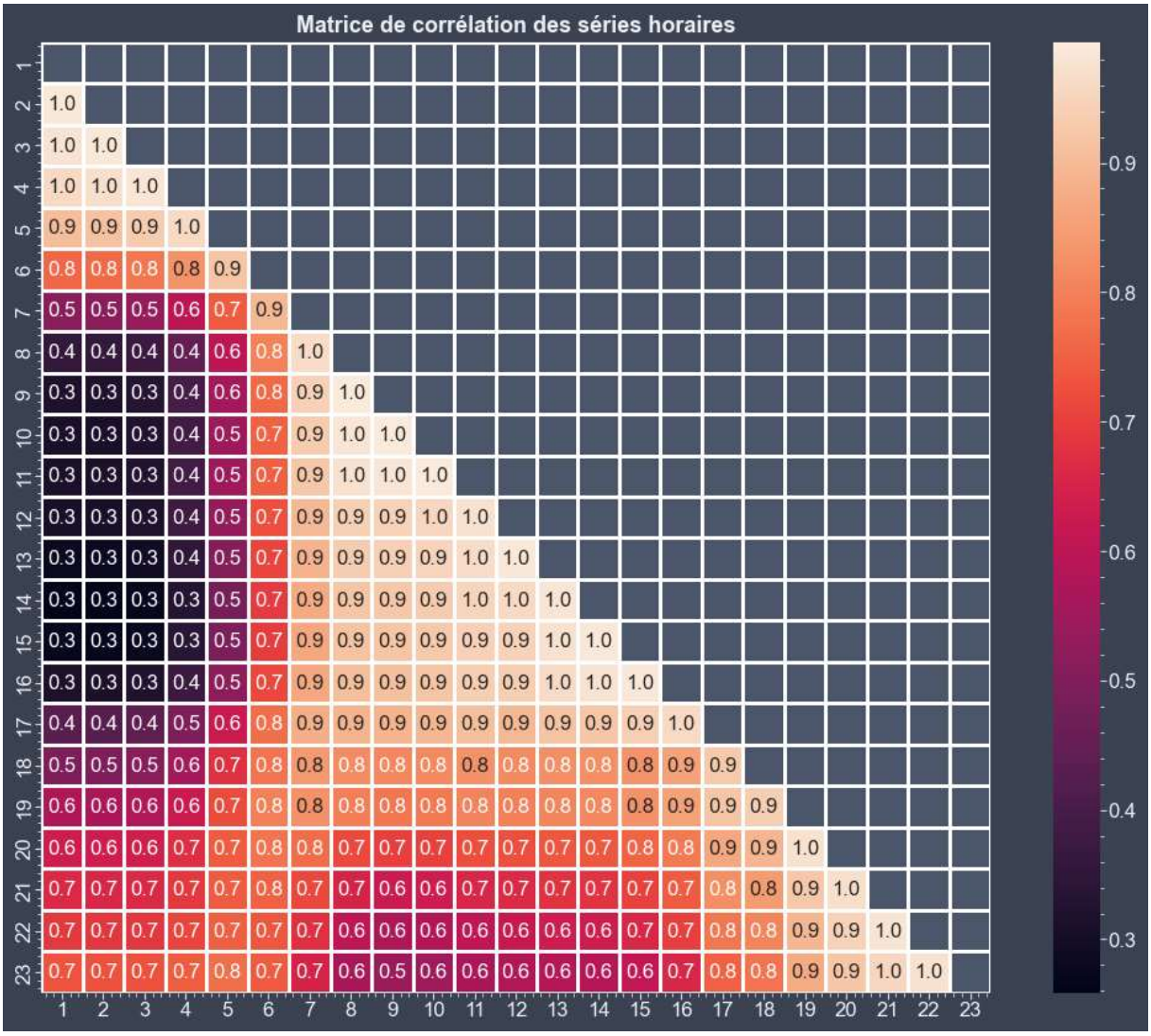
Date	Salaire indiv. 1	Salaire indiv. 2	Salaire indiv. 3
01/01/2021	1200€	3000€	1200€
01/01/2022	1220€	3000€	2200€
01/01/2023	1242€	3000€	3200€
…	…	…	…

Ainsi, utiliser le KMeans dans ce contexte revient en fait à identifier des clusters de dates à partir des différentes séries temporelles. Cela peut être utile. Par exemple si l'on cherche à identifier les jours de la semaine ou la productivité des employés est la plus haute, ou les mois ou l'on retrouve le plus d'arrêt maladie. Cependant, dans nombre de cas, on voudra segmenter les séries temporelles directement. C'est-à-dire identifier des clusters parmi nos différentes séries temporelles. Si l'on reprend le même exemple que précédemment, cela revient à estimer des segments de salaires parmi nos employés. L'objet du KMeans n'est alors plus les observations de notre tableau, mais les variables ! En d'autres termes, la *scène disparaît pour devenir l'un des acteurs*.

La solution simple pouvant être adoptée consiste tout simplement à transposer la matrice initiale et appliquer le KMeans. Ce faisant, on s'expose à deux problématiques. La première étant que dans la majorité des cas, on dispose de plus de pas de temps que de séries. Dans notre cas, nous disposons de 23 séries (pour les 23 heures de la journée) pour 1824 pas de temps. On se place donc dans une situation avec plus de colonnes que de lignes. Cela nous expose au *fléau de la dimension*. Pour faire simple, dans un tel contexte, la distance relative entre les valeurs proche augmente tandis que celle entre les extremums diminue. En d'autres termes, tous les points deviennent équidistants. Difficile donc de classifier selon des distances quand les distances ne signifient plus rien. Le second problème est un problème plus simple, la corrélation.

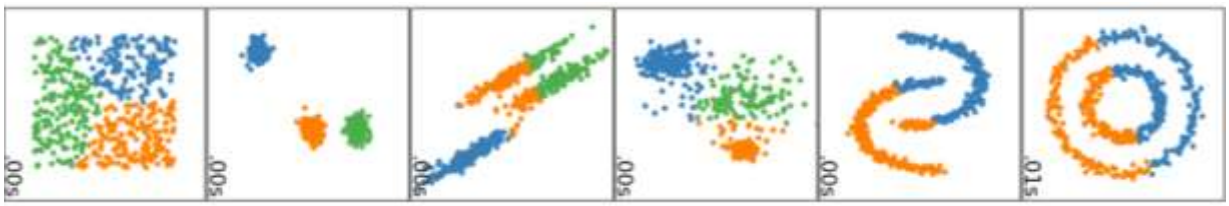
La corrélation entre les séries temporelles

Etant donné que les séries temporelles dont nous disposons décrivent un même phénomène (une demande, une consommation, une productivité, etc), il ne paraît pas aberrant de penser que ces dernières peuvent être corrélées. Affichons la matrice de corrélation pour en avoir le cœur net :



On observe que les heures proches sont très corrélées entre elle.

Le problème majeur du KMeans est que ce dernier considère de par l'utilisation de la distance euclidienne que les variables sont décorrélées les unes aux autres. Plus précisément, il fait l'hypothèse que la matrice de variance-covariance est diagonale. En d'autres termes, le KMeans fait la supposition que les données sont distribuées de façon sphérique. C'est-à-dire qu'il faut pouvoir tracer des cercles autour des nuages de points représentatifs de chaque cluster. Si ce n'est plus le cas, le KMeans donne des résultats pouvant être très mauvais. Voici un exemple issu de *sklean* de la performance du KMeans sur différents nuages de points.



On peut voir que quand les points sont distribués selon des formes d'éclipses (c'est-à-dire que les axes sont corrélés) ou selon une distribution non linéaire, le KMeans perd en efficacité (comme sur la 3e image). On pourrait donc vouloir changer la distance du KMeans pour que cette dernière prenne en compte la covariance, avec la distance de Mahalanobis par exemple :

$$D_M = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})}$$

Nous ne développerons pas ce point ici, mais cela revient en fait à utiliser l'algorithme de *Gaussian Mixture Model*. Les équations de ce dernier changent de par son approche bayésienne, mais le principe revient à celui du KMeans avec la distance de Mahalanobis.

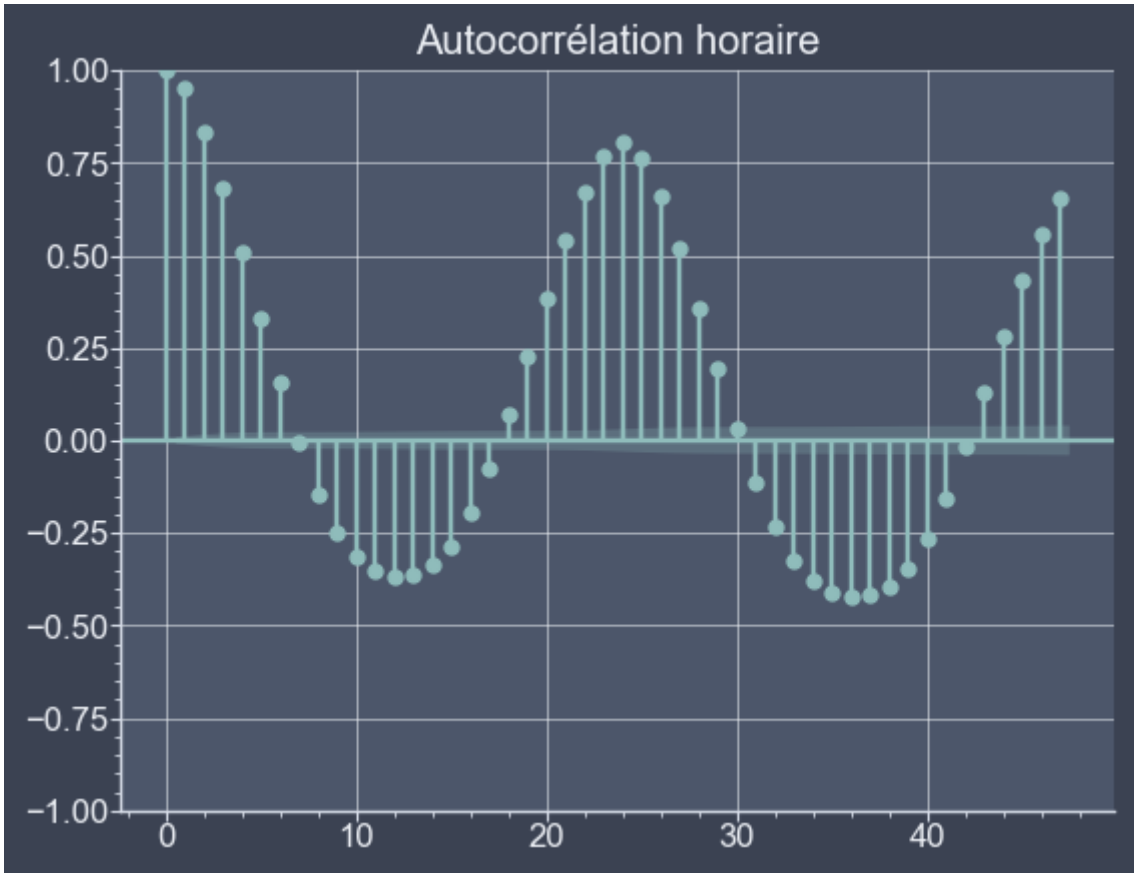
L'autocorrélation

Les séries temporelles présentent un second problème singulier pour celui qui veut utiliser le KMeans sur ces dernières. En effet, en plus d'être corrélées avec les autres, les séries tendent souvent à être également corrélées avec elles-mêmes, c'est le principe de l'autocorrélation. Cela fait sens, la demande d'électricité d'un jour est liée à celle du lendemain. Autrement dit, les séries au moment *t* intègrent une part d'information de leur valeur au moment *t+1.*

Plus formellement, l'autocorrélation mesure la corrélation entre un moment *t* et les moments précédents (*t-1, t-2, … t-k, …*). Mathématiquement, on peut la définir comme :

$$\rho_k = \frac{\sum_{t=k+1}^{n} (r_k - \bar{r})(r_{t-k} - \bar{r})}{\sum_{t=1}^{n} (r_t - \bar{r})}$$

Comme pour la corrélation, cette dernière est comprise entre 1 et -1. Voici un graphique illustrant l'autocorrélation entre la demande sur les différentes heures de la journée :

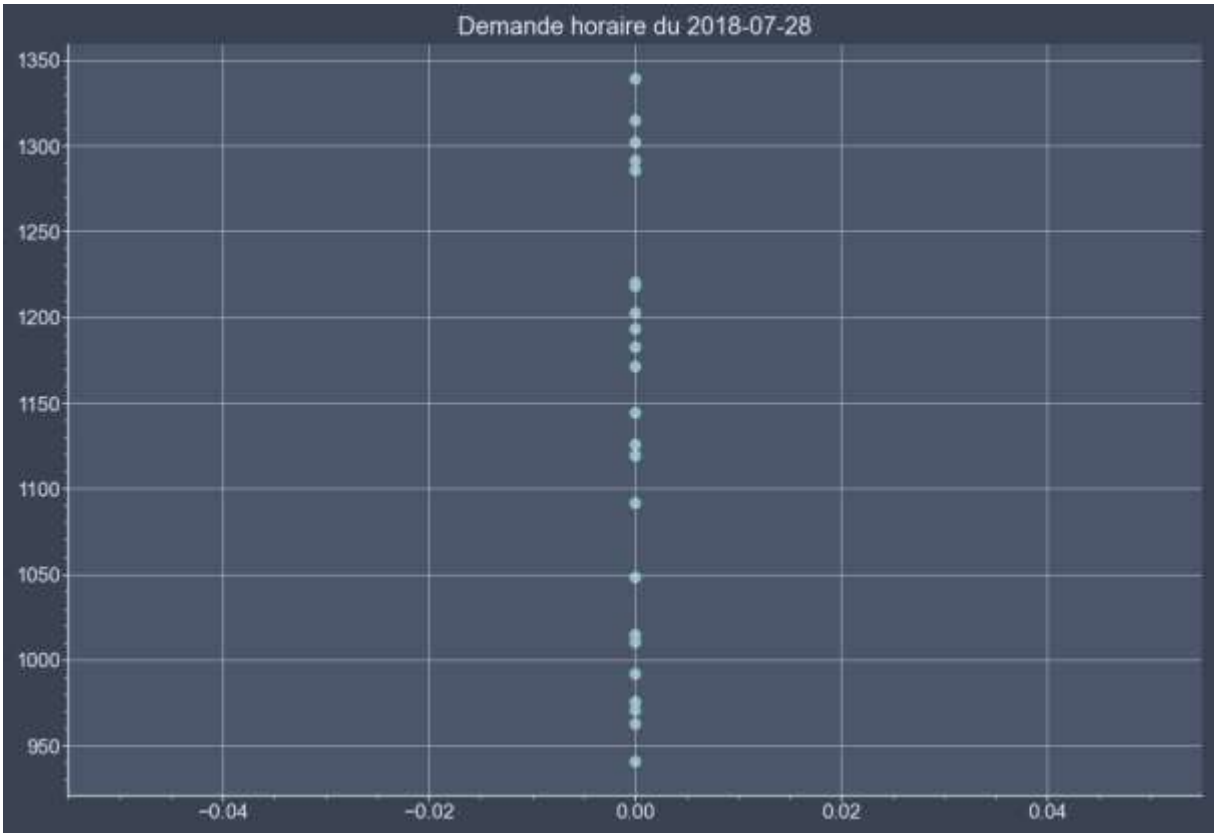


On voit que pour une heure donnée, cette dernière est corrélée positivement aux 6 dernières heures, puis négativement aux 8 à 16 dernières heures etc. De plus, on peut observer un processus saisonnier journalier puisque le même pattern semble se dessiner tous les jours.

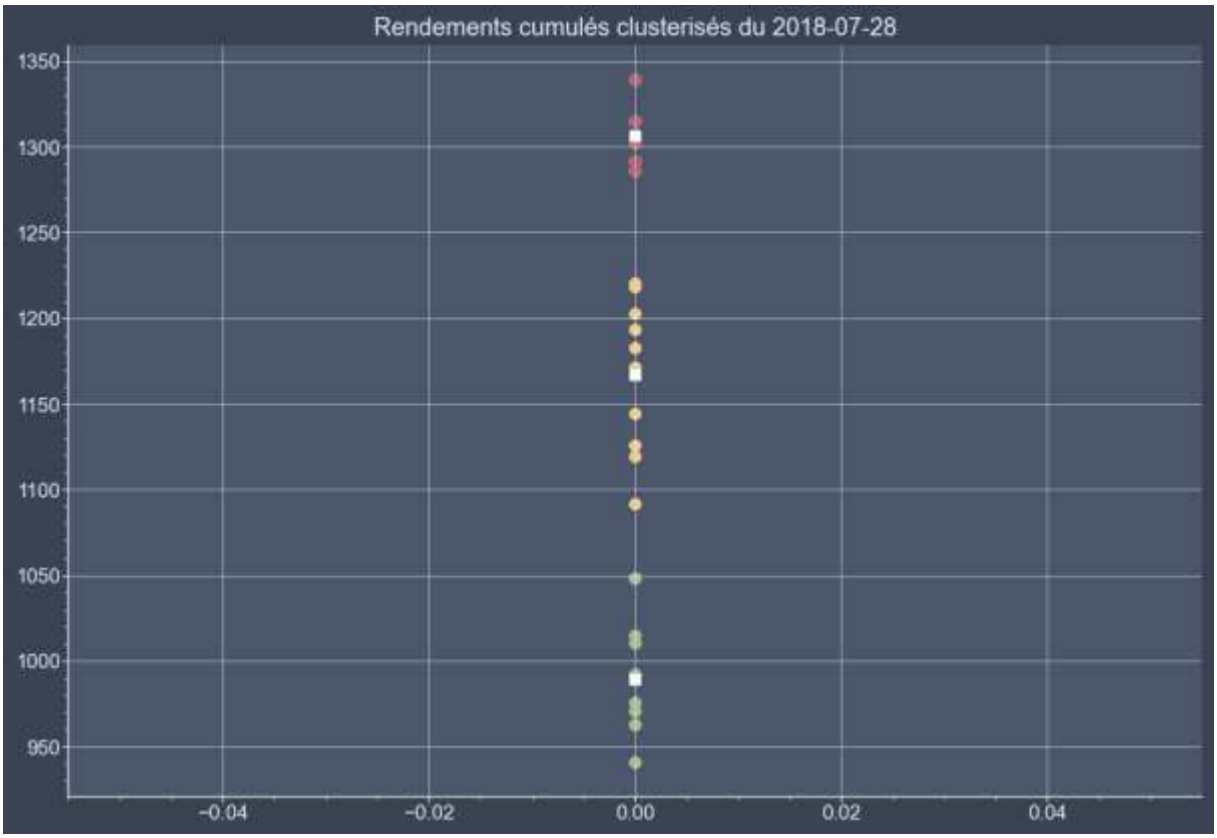
La variation temporelle du KMeans

Ainsi, nous avons vus les 2 grand problèmes du KMeans : La non adaptabilité au contexte des séries temporelles ainsi que le problème des (auto)corrélations. Je propose ici une solution relativement gourmande en temps de calcul, mais qui permette de palier à ces problèmes.

La solution proposée ici repose sur un mécanisme assez simple. Plutôt que d'appliquer l'algorithme directement sur l'ensemble des données, c'est-à-dire l'ensemble des t périodes, on peut appliquer ce dernier sur chaque période indépendamment et agréger ensuite les résultats. Pour illustrer cette idée, prenons l'exemple d'une distribution des demandes horaire pour une date choisie aléatoirement :

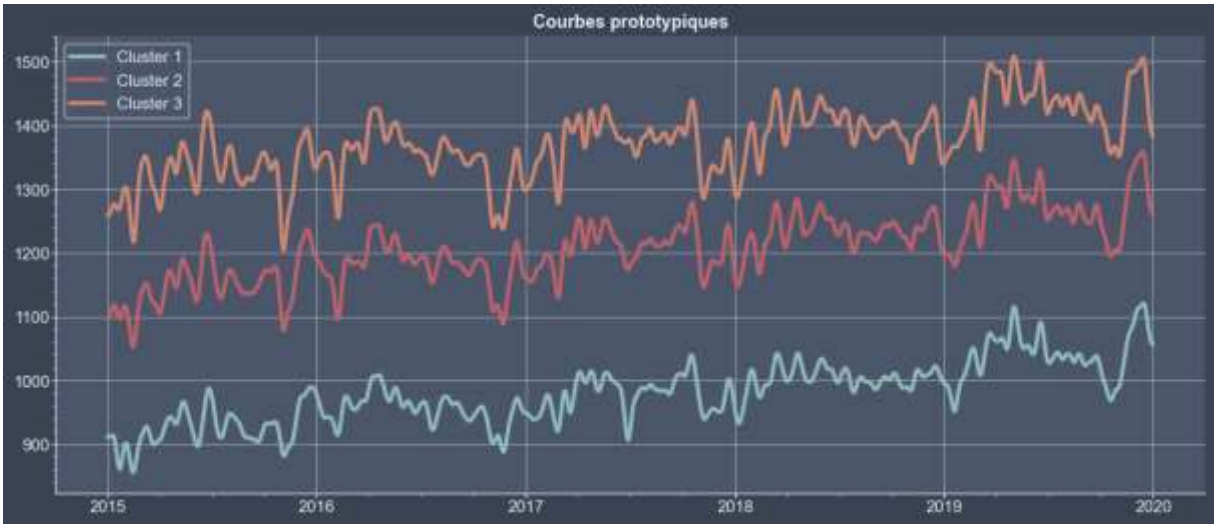


On voit ici que les points vivent dans un espace unidimensionnel, il n'y a donc pas de problématique de corrélations lors de l'application de l'algorithme. Qui plus est, le classement en k clusters paraît trivial sur un espace à une dimension. On s'assure donc une convergence quasi systématique de l'algorithme sur les mêmes centroïds, ce qui évite de relancer plusieurs fois l'algorithme. Voici le résultat pour notre distribution de demande horaire :



On a donc *k* clusters (3 ici) marqués par des carrés blans avec les points qui leurs sont associés marqués par différentes couleurs. L'idée de l'algorithme proposé ici est de reproduire le même processus pour chaque période *t*. En reliant les coordonnées de chacun des centroïds, on peut alors tracer *k* des « courbes prototypiques », ou chaque point constitutif d'une courbe est en réalité la position du cluster associé à cette courbe au moment *t*. Il faut ensuite s'assurer que les clusters soient *cohérents*. C'est-à-dire que si le cluster du haut a été numéroté par le nombre « 1 », il faut que ce soit le même nombre sur tous les pas de temps. Pour ce faire, on renomme chaque cluster en fonction de la position du centroïd (par ordre croissant, de sorte que le cluster du bas soit 0 et le cluster du haut soit *k*).

Si les données dont nous disposons sont bruitées, nous pouvons également avoir recours à une moyenne mobile ou une convolution par un filtre gaussien pour lisser les courbes :



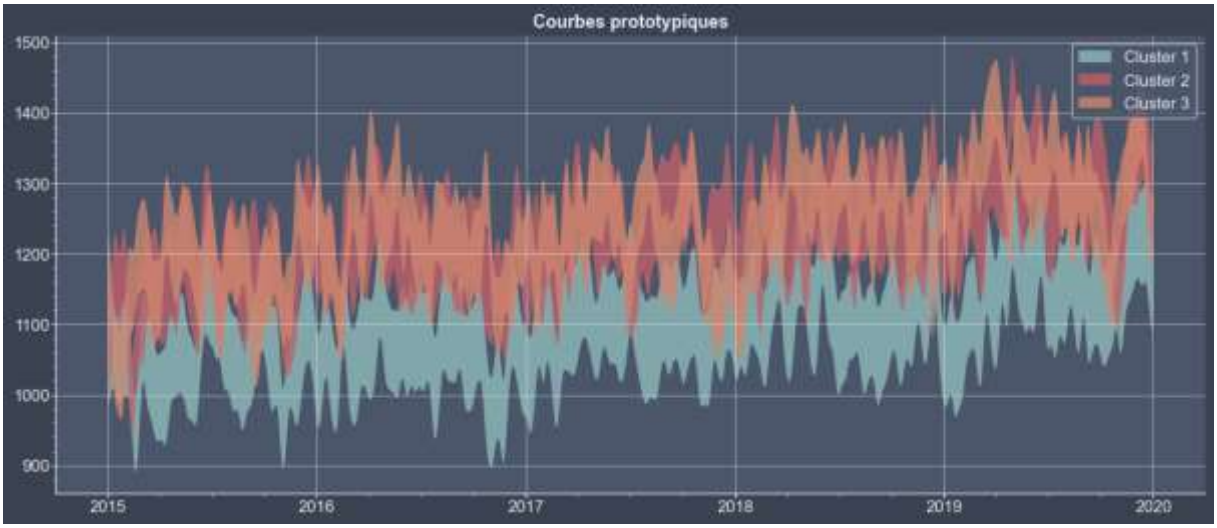
On dispose ici de trois clusters – trois *comportements moyens* (le nombre est définie arbitrairement mais nous reviendrons dessus par la suite). Les clusters obtenus sont les suivants :

- De 23h à 7h : **Le cluster 1**
- De 10h à 15h : **Le cluster 3**
- Le reste : **Le cluster 2**

Cela paraît assez cohérent.

Evaluation des courbes prototypiques

On observe que les 2 clusters hauts obtenus précédemment sont relativement proches, aussi il peut être pertinent de tracer les valeurs minimums et maximums associées à chacun des clusters pour chaque pas de temps afin d'en estimer la pertinence :

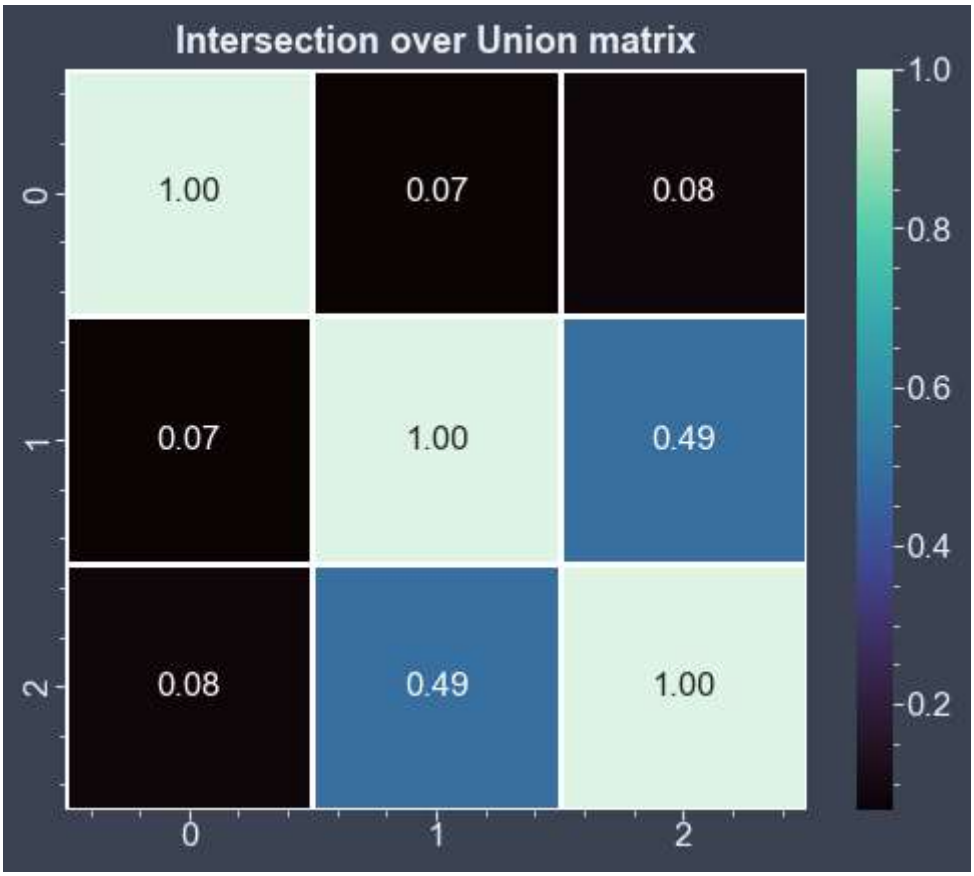


On observe que les courbes supérieures se chevauchent, le recours à 3 clusters ne semble donc pas pertinent. On peut, pour évaluer mathématiquement la qualité des clusters calculer les IoU (*Intersection Over Union*) entre les intervalles relatifs à chaque cluster. Pour rappel, si je dispose de 2 clusters k_1 et k_2, l'IOU se définit comme ceci :

$$IoU = \frac{k_1 \cap k_2}{k_1 \cup k_2}$$

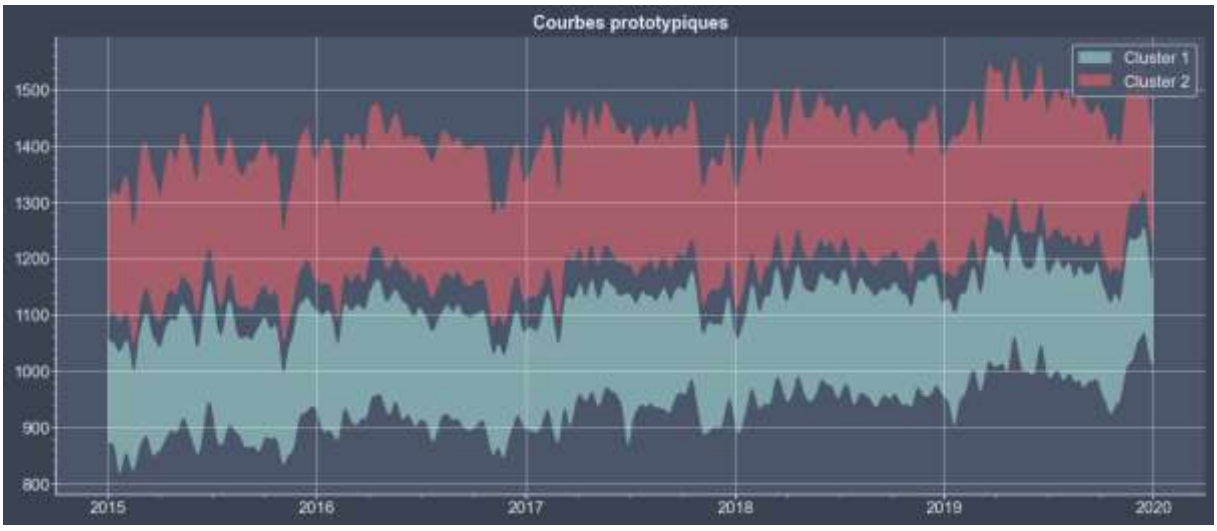
Ou de façon plus visuelle :

Plus cette dernière est proche de 1, plus l'intersection est proche de l'union, autrement dit, plus nos clusters sont mauvais. On peut ainsi calculer une matrice calculant les *IoU* moyens entre chacun de nos clusters :

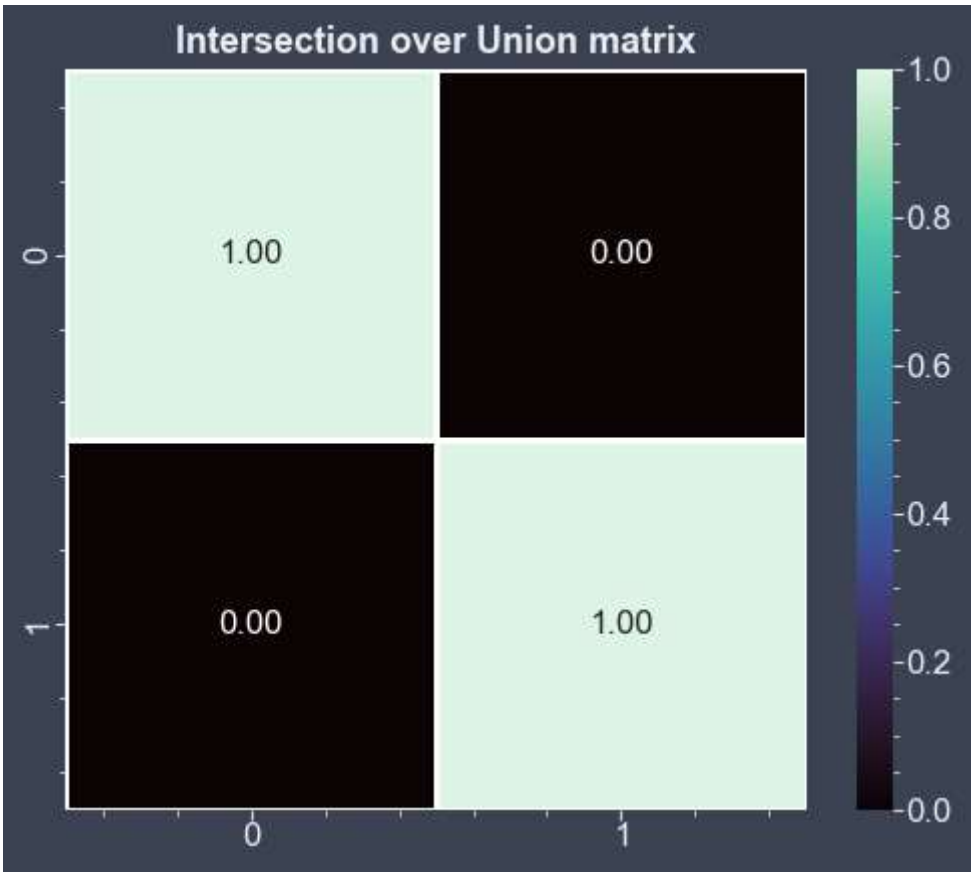


On observe un résultat proche de 0.5 entre les clusters 2 et 3. Ces 2 clusters ne sont donc pas pertinents. A titre indicatif, cette mesure est utilisée en reconnaissance d'image pour évaluer si la position prédite d'un objet est bonne. On considère à partir de 0.5 que le résultat est bon. Dans notre cas, il faut donc interpréter les choses dans le sens inverse.

En réduisant à 2 clusters, le résultat obtenu semble plus pertinent :



Les clusters obtenus semblent bien meilleurs, qu'en est-il de l'IoU :



L'IoU obtenu confirme bien cette hausse de performance.

L'ajustement du nombre de cluster peut donc être réalisé à l'œil ou par le biais de l'IoU - en fixant un seuil de tolérance minimale et itérant jusqu'à l'atteindre.

Prédiction d'une nouvelle observation

Si l'on veut prédire une nouvelle série temporelle, c'est-à-dire l'associer à l'un de nos clusters, il suffit de calculer la norme 1 (qui équivaut à la distance de Manhattan) ou la norme 2 (qui équivaut à la distance euclidienne) entre l'ensemble des points de la série et l'ensemble des différents centroïds, puis de sommer le tout. Pour chaque cluster *k* ayant comme centroïd *c*, on a dans le cas de la norme 1 :

$$d_k = \left|\left|s - c\right|\right|_1 = \sum_{t=0}^{n} |s_t - c_t|$$

Ou dans le cas de la norme 2 :

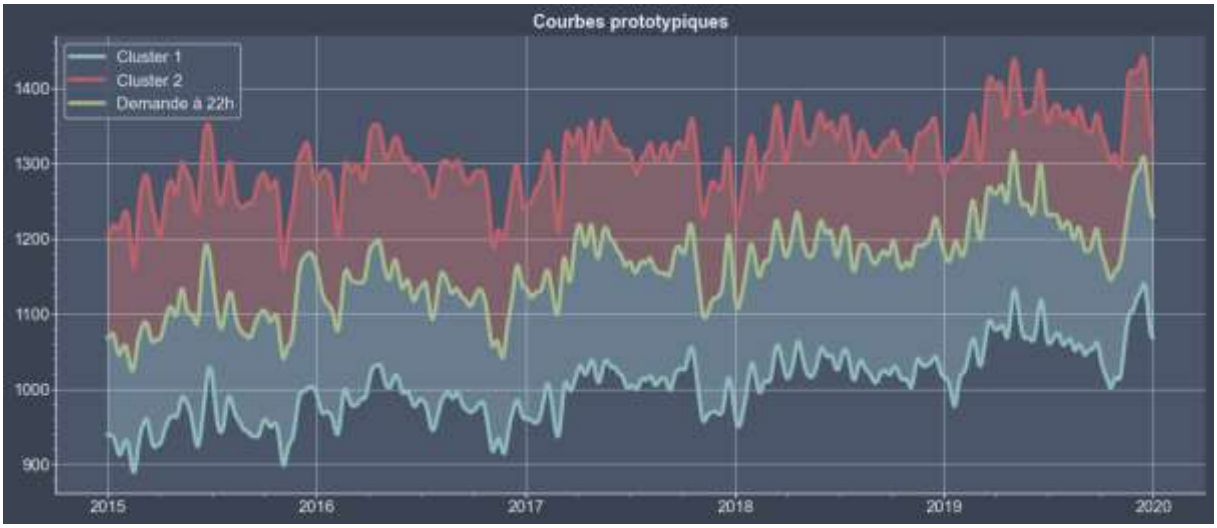
$$d_k = \left|\left|s - c\right|\right|_2 = \sum_{t=0}^{n} (s_t - c_t)^2$$

On obtient donc un vecteur avec la somme des distances correspondant à chacun des clusters. Pour *K*, clusters, on aura :

$$\vec{d} = (d_1, d_2, \ldots, d_K)$$

On peut ensuite adopter deux stratégies à partir de ce vecteur de distance.

Pour donner un exemple concret, si l'on essaye de clustériser la série correspondant à la demande en électricité à 22h, on aurait :



Avec en jaune la demande à 22h et les aires en rouge et bleu correspondant respectivement aux distances aux cluster 1 et 2 (Au lieu d'une aire, nous devrions en réalité avoir une série de segment reliant la courbe jaune aux courbes prototypiques en chaque pas de temps, cependant, ceux-ci étant très nombreux compte tenu de l'échelle qu'il n'est pas aberrant de les considérer comme une aire).

Le hardmin

La stratégie de *hardmin* est simple, elle consiste à choisir le cluster pour lequel la distance à la série est la plus faible. Autrement dit, on assignera à la série le cluster *k* selon la règle suivante :

$$cluster\ assigné = \underset{k}{\operatorname{argmin}}\, \vec{d}$$

En utilisant la stratégie de hardmin sur la courbe précédente, on associe la demande d'électricité à 22h au cluster 1. Cependant, les deux courbes prototypiques paraissent relativement équidistantes, s'arrêter au hardmin peut nous faire perdre cette information vis-à-vis de l'incertitude du modèle.

Le softmax

Une autre approche plus « soft » consiste à estimer pour chaque cluster une probabilité d'appartenance. L'approche se fait en deux temps.

On normalise toutes les distances comme ceci (pour éviter les problèmes de stabilité lors du calcul de la fonction *softmax*) :

$$nd_k = \frac{d_k}{\left|\left|\vec{d}\right|\right|_1}$$

On passe alors par la fonction *softmax*, définie par la formule ci-après :

$$\sigma(d_k) = \frac{e^{nd_k}}{\sum_{j=1}^{K} e^{nd_j}}$$

$\sigma(d_k)$ étant la probabilité que la série étudiée appartienne au cluster k. On a donc une probabilité pour que notre série appartienne à chacun des clusters. L'assignation par la sélection de la probabilité la plus haute (de manière similaire au *hardmin*). L'intérêt de cette méthode, c'est de pouvoir avoir des probabilités d'appartenance à chaque cluster, et donc d'évaluer la « certitude » du modèle concernant le classement d'une série temporelle. Cela peut être un bon point pour évaluer la qualité d'une prévision. Dans notre cas par exemple, pour la demande d'électricité à 22h, il nous ait dit :

```
La probabilité pour que la demande soit associée au cluster 1 est de 0.49
La probabilité pour que la demande soit associée au cluster 2 est de 0.51
```

Dès lors on voit que le modèle est très incertain sur le classement de cette dernière.

Evaluation statistique d'une prévision

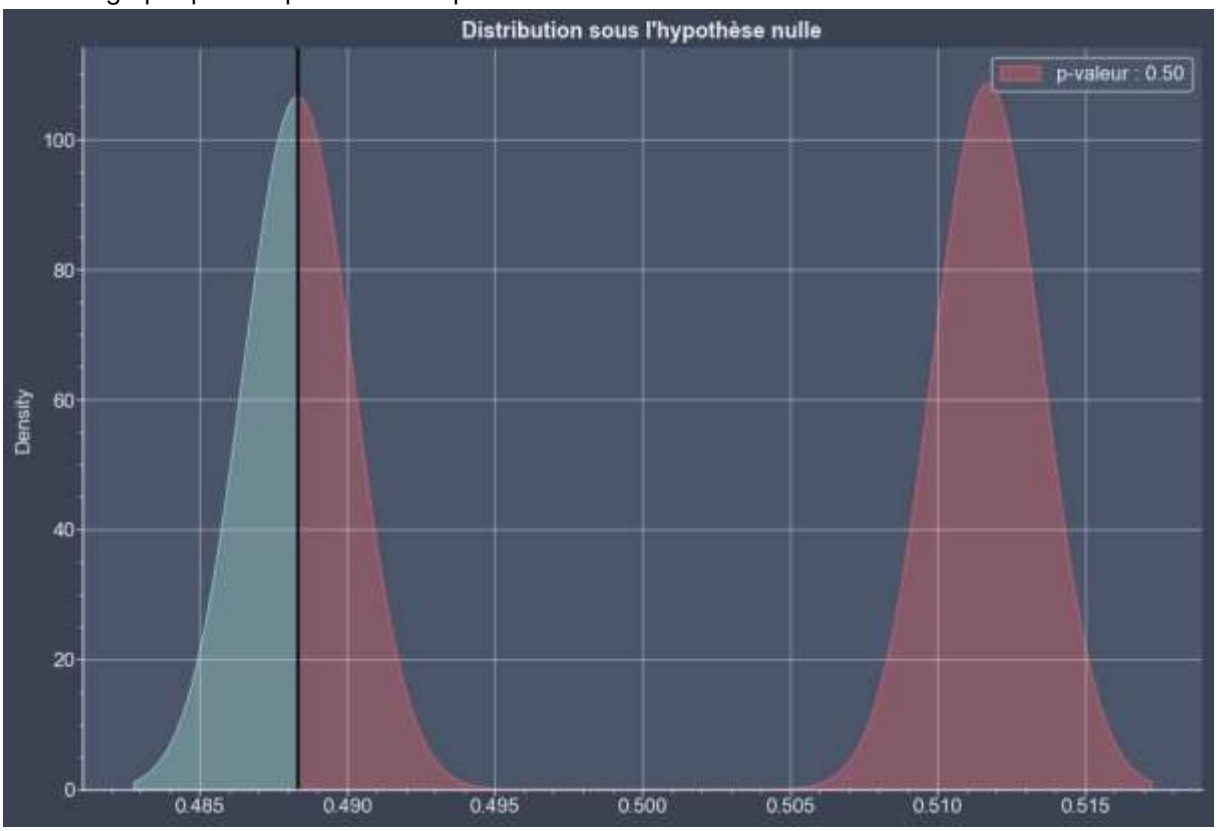
La performance d'un modèle de machine learning peut être décrite sous un œil statistique avec la question suivante : Quelle est la probabilité d'obtenir le résultat que j'observe dans un contexte de hasard ? Plus l'on minimise cette dernière, plus le résultat devient bon : La probabilité que le résultat que j'observe soit dû au hasard devient très faible.

Pour évaluer statistiquement notre prévision, on peut adopter la méthodologie suivante. On permute aléatoirement un grand nombre de fois l'ensemble de nos courbes prototypique et l'on ré effectue la prédiction. En d'autres termes, on se place dans un contexte de hasard (on parle statistiquement de H_0 ou de l'hyptohèse nulle). On enregistre l'ensemble des probabilités obtenues et l'on estime à la fin la part des probabilités supérieures à la nôtre. Cette part est la probabilité que le résultat obtenu soit dû au hasard. Soit plus formellement, si l'on considère la probabilité gagnante p_{obs} :

$$p_{valeur} = P(p > p_{obs}|H_0)$$

p étant la probabilité calculée sous H_0.

Voici un graphique récapitulatif de ce procédé :



Si l'on fait le même exercice mais avec un autre horaire (comme 10h), on obtient un résultat beaucoup plus faible avec une p-valeur de ~0. Cette méthodologie peut donc être utilisée afin d'évaluer la qualité d'une prévision.

Remarques

Pour aller plus loin, on pourrait vouloir généraliser à plusieurs variables par pas de temps, en appliquant le même procédé mais avec un *Gaussian Mixture Model* (si les données sont linéairement séparables) à chaque étape ou un algorithme non linéaire le cas échéant. A noter que dans cette optique, la structure des données doit rester fondamentalement la même au fil du temps. On pourrait également s'atteler à essayer de prédire l'évolution des comportements moyens pour anticiper un comportement futur. Techniquement, il s'agit la de réaliser une prévision de série temporelle directement sur les courbes prototypiques obtenues. Ce seront peut-être des sujets traités prochainement.