École Nationale de la Statistique et de l'Administration Economique

Algorithme de Chow-liu

Liens entre les statistiques d'un joueur de basket

Léo René, Arman Akgönül

Encadrant : Xavier Dupré

24 octobre 2022

Table des matières

Introduction

L'objectif de notre étude est de créer un graphe qui saura représenter la dépendance de certaines variables entre elles. Cette ambition est d'autant plus intéressante que les données sont nombreuses. En effet, ce graphe répond en partie à un problème de stockage, car il va stocker une approximation probabiliste plutôt que l'ensemble de la base. Les problèmes de stockages sont récurrents et anciens [1]. Ce problème de stockage fait partie du "curse of dimensionnality" c'est à dire tous les obstacles qui surviennent lors de l'analyse de données en grande dimension. Dès lors, nous cherchons un moyen de répondre a cette problématique, une des méthode possible est l'approximation produit, c'est-à-dire une approximation d'un ordre élevé par le produit de plusieurs de ses distributions aux ordres plus bas. Pour obtenir cette approximation produit, notre choix se porte sur l'algorithme de Chow-liu. Partant de variables aléatoires à support fini, et en se restreignant à un produit de distributions marginales, il va créer des liens entre ces variables. A partir de ces liens, on peut tracer un arbre au sens de la théorie des graphes. Dans cet arbre, les sommets sont les variables aléatoires, et les arêtes représentent "l'information mutuelle", qui est une mesure de la dépendance de deux variables entre elles. L'algorithme a besoin d'une large base de données discrètes de variables aléatoires, l'algorithme de chow-liu va déterminer un lien entre chacune d'elles avant de classer par ordre croissant ces liens puis d'en faire un graphe. Nous appliquerons cet algorithme à une base de données regroupant de larges informations sur des joueurs de Basket afin d'exhiber les liens entre les différentes statistiques sportives. Nous tâcherons dans un premier temps d'expliciter la théorie sur laquelle se repose notre algorithme I. Ensuite, nous passerons à la partie pratique en implémentant notre algorithme II. Enfin, nous appliquerons ce dernier à notre base de données III.

1. Lewis II : Approximating Probability Distributions to Reduce Storage 1959

I-Comprendre le problème

I.1 - Cadre

Nous ferons l'hypothèse que toutes les variables aléatoires étudiées seront discrètes pour faciliter notre étude.

Par ailleurs, pour pouvoir confronter puis comparer les différentes variables, nous avons besoin d'une base de données pleine, c'est-à-dire dont aucun élément n'est manquant. Nous ferons également cette hypothèse dans la suite du travail

I.2 - Théorie mathématique

I.2.1 - Notations

Soient n variables aléatoires discrètes $X_1, ..., X_n$ dans un espace probabilisé. Avec $\forall i \in \{1, ..., n\}, |\chi_i| < +\infty$ où χ_i est le support de X_i. soit $X = (X_i)_{i=1}^n$ prenant des valeurs dans $\chi = \times_{i=1}^n \chi_i$ et $x = (x_i)_{i=1}^n \in \chi$ une issue possible. La probabilité que $X = x$ est alors :

$$p(x) = \mathbb{P}(X = x)$$

On note p $= \{p(x)\}_{x \in \chi}$ la distribution de probabilité sur χ donné par la probabilité $p(x)$. Pour calculer les distances on s'intérèsse maintenant aux probabilités conditionnelles. On pose $I = \{1, ..., n\}$. Soient $A \subseteq I$ et $B \subseteq I$. On définit $x_{A \cup B}$ comme ceci : Soit $\chi_A = \times_{i \in A} \in \chi_A$. On a alors

$$x_{A \cup B} = (x_i)_{i \in A \cup B}$$

. Si $A \cap B = \varnothing$, par définition :

$$p_{A \cup B}(x_A, x_B) = p_{A \cup B}(x_{A \cup B})$$

Or,

$$p_K(x_K) = \mathbb{P}(X_K = x_K) = \sum_{x_{K^c} \in \chi_{K^c}} p(x_K \cdot x_{K^c}) \; avec \; K \subseteq I$$

Donc

$$p_{A \cup B}(x_A, x_B) = \sum_{x_{(A \cup B)^c} \in \chi_{(A \cup B)^c}} p(x_{A \cup B} \cdot x_{(A \cup B)^c})$$

On s'intéresse qu'au cas $p_B(x_B) > 0$ (car $p_B(x_B) = 0 \Rightarrow p_{A \cup B}(x_{A \cup B}) = 0$) :

$$p(x_A | x_B) = \frac{p_{A \cup B}(x_A, x_B)}{p_B(x_B)}$$

On peut interpréter cela comme $\mathbb{P}(X_A = x_A | X_B = x_B)$.

On introduit maintenant les structures de dépendance. Soit $(A_i)_{i=1}^k$ une partition de I et soit S une séquence de paires telles que :

$$S = (A_i, B_i)_{i=1}^k$$

telle que

$$B_1 = \varnothing, B_r \subset \cup_{i=1}^{r-1} A_i \subseteq I, \; r = 2, ..., k$$

Alors S est une *structure de dépendance*

S nous permet de définir l'approximation produit de la distribution p :

$$p(x|S) = p_{A_1}(x_{A_1}) \prod_{i=2}^{k} p(x_{A_i}|x_{B_i}),\ x \in \chi$$

On introduit aini :

$$\mathrm{p}_S = \{p(x|S)\}_{x\in\chi}$$

Par exemple en prenant k=n, $A_i = \{i\}$ pour $i = 1, 2, ..., n$ et pour $i = 1, 2, ..., n, B_i = \varnothing$ B est une structure de dépendance. On peut alors écrire :

$$p(x|S) = \prod_{i=1}^{n} p_i(x_i), \qquad x = (x_i)_{i=1}^{n} \in \chi$$

On approche une probabilité par un produit de ses distributions marginales de premier ordre.

I.2.2 - Divergence de Kullback-Leibler et entropie de Shannon

Par définition la divergence de Kullback-Leibler est :

$$D(\mathrm{p}||\mathrm{p}_s) = \sum_{x\in\chi} p(x) \ln \frac{p(x)}{p(x|S)}$$

D est une distance. Elle mesure la dépendance des variables aléatoires entre elles, on va ainsi pouvoir les classer. On l'applique ici car l'un des buts de l'algorithme de Chow-Liu est la compression de données.

Le but va être de trouver le minimum de cette distance entre la loi étudiée et un certain type de loi explicité après.

On introduit également l'entropie de Shannon ici. On la note $H(A)$ et elle se définit comme suit :

$$H(A) = -\sum_{x_A\in\chi_A} p_A(x_A) \ln(p_A(x_A))$$

et si $A = \varnothing$ alors $H(A) = 0$ A partir de cette expression on en déduit :

$$D(\mathrm{p}||\mathrm{p}_s) = \sum_{x\in\chi} p(x) \ln p(x) - \sum_{x\in\chi} p(x) \ln p(x|S)$$

On admet que $H(I) = -\sum_{x\in\chi} p(x) \ln p(x)$ L'équation précédente devient alors :

$$D(\mathrm{p}||\mathrm{p}_s) = -H(I) - \sum_{x\in\chi} p(x) \ln p(x|S)$$

Après calcul on obtient une nouvelle expression de la distance en fonction de l'entropie de Shannon.

$$D(\mathrm{p}||\mathrm{p}_s) = -H(I) - \sum_{i=1}^{k} [H(A_i \cup B_i) - H(B_i)]$$

I.2.3 - Information mutuelle (MI)

L'information mutuelle sera une mesure du degré de dépendance entre deux variables aléatoires X_A et X_B avec $A \cap B = \varnothing$. Elle est définit par :

$$MI(A,B) = H(A) + H(B) - H(A \cup B)$$

$$MI(A,B) = \sum_{x_{A\cup B} \in \chi_{A\cup B}} p_{A\cup B}(x_{A\cup B}) \ln \frac{p_{A\cup B}(x_{A\cup B})}{p_A(x_A) p_B(x_B)}$$

Cela peut se réécrire avec la distance vue précédemment :

$$MI(A,B) = D(\mathrm{p}_{A\cup B} || \mathrm{p}_A \times \mathrm{p}_B)$$

Cette notion nous sera utile pour l'application à l'algorithme de Chow-liu

I.3 - Application à l'algorithme de Chow-Liu

I.3.1 - Théorie

L'algorithme de Chow-Liu va construire l'arbre de dépendance au premier ordre grâce a la mutual information précédemment définie. On va donc obtenir la loi $P'(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | X_{\sigma(i)}), \sigma : \mathbb{N} \to \mathbb{N}$ tq $\sigma(i) < i$ $\forall i \in \{1, ..., n\}$ qui minimise la divergence de Kullback-Leibler avec la loi jointe des $(X_1, ..., X_n)$.

Voici par exemple, l'arbre de dépendance correspondant à la loi P' tq :

$P'(X_1, X_2, X_3, X_4, X_5, X_6) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_2)P(X_5|X_2)P(X_6|X_5)$

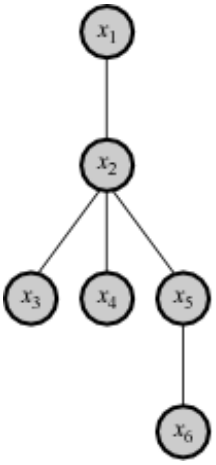

FIGURE 1 – Exemple d'arbre sorti par l'algorithme de Chow-Liu

Pour cela, on calcule $\forall i \in \{1, ..., n\}$ les $MI(X_i, X_j)$ qu'on notera $L_{i,j}$. Les $L_{i,j}$ sont symétriques.

On détermine tout d'abord $\forall i \in \{1, ..., n\}$ les $MI(X_i, X_j)$ notés $L_{i,j}$, on remarquera qu'ils sont symétriques. Ils formeront le poids des arêtes entre les variables aléatoires Xi qui seront les sommets du graphes. On applique ensuite à ce graphe un algorithme d'ACPM (arbre couvrant de poids maximal) qui va mettre en lien les variables les plus dépendantes.

Voici un exemple de cet algorithme :

I.3.2 - Illustration

Soit $U = \{X_1, ..., X_n\}$ avec $\forall i \in \{1, ..., n\} X_i$ une variable aléatoire discrète. On prend n=5 ici. On calcule les différents MI pour les variables aléatoires qu'on regroupe ci-dessous :

- $L_{1,2} = 0.55$
- $L_{1,3} = 0.34$
- $L_{1,4} = 0.11$
- $L_{1,5} = 0.59$
- $L_{2,3} = 0.68$
- $L_{2,4} = 0.03$
- $L_{2,5} = 0.25$
- $L_{3,4} = 0.01$
- $L_{3,5} = 0.22$
- $L_{4,5} = 0.10$

Donc d'après le principe vu précédemment, $L = [L_{2,3}, L_{1,5}, L_{1,2}, L_{1,3}, L_{2,5}, L_{3,5}, L_{1,4}, L_{4,5}, L_{2,4}, L_{3,4}]$, on relie tout d'abord les variables dont le$L_{i,j}$ est le plus élévé (Kruskal). On relie alors X_2 et X_3. Ensuite on relie les sommets X_1 et X_5 sur le même principe. On continue avec X_1 et X_2. Ensuite on devrait relier X_1 et X_3. Cependant ces variables font déjà partie d'un même arbre. On passe alors à la suivante. X_2 et X_5 sont également dans le même arbre, tout comme X_3 et X_5. On passe alors à X_1 et X_4. Une fois ces deux sommets reliés, toutes les variables sont reliées entres elles, le graphe est alors terminé.

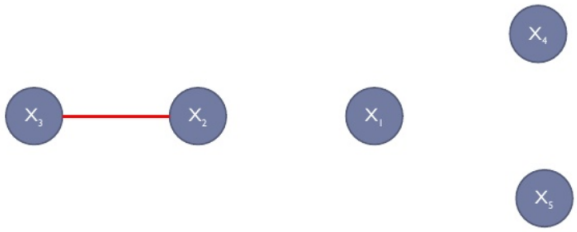

FIGURE 2 – Première étape

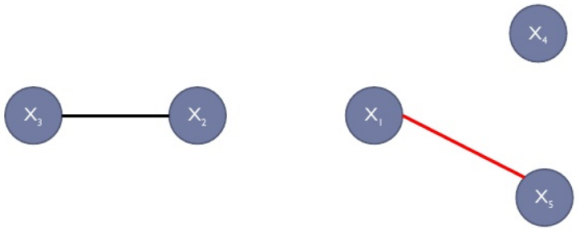

FIGURE 3 – Deuxième étape

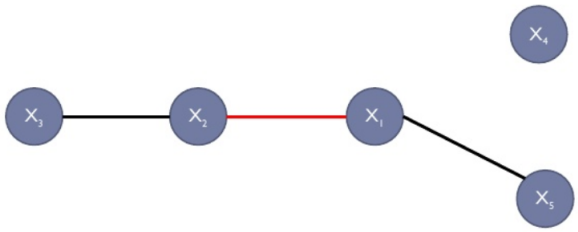

FIGURE 4 – Troisième étape

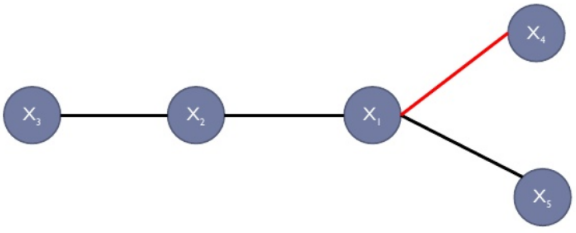

FIGURE 5 – Quatrième étape

II - Mise en pratique de l'algorithme

II.1 - Implémentation de l'algorithme

Notre but étant d'implémenter un algorithme d'ACPM, nous avons besoin d'arbres. Nous représentons les notres par un doublet (liste des sommets, liste des arcs) où un arc se décompose (poids,sommet,sommet). Nous avons ensuite utilisé des dictionnaires pour coder l'algorithme de Kruskal (voir code)

Habituellement, l'algorithme d'ACPM minimise l'arbre couvrant, nous voulons nous le maximiser. Donc pour l'algorithme de Chow-Liu, nous avons cette fois trié notre liste d'arcs de manière décroissante du poids. Ensuite, nous avons crée un graphe pondéré non orienté qui a comme sommets nos variables aléatoires. Elles sont toutes reliées et le poid de la liaison est déterminé en fonction de l'information mutuelle des deux variables. Pour calculer ces informations mutuelles nous avons également utilisé un dictionnaire (détailler ?)

Enfin, pour représenter cet arbre, nous utilisons le module networkx

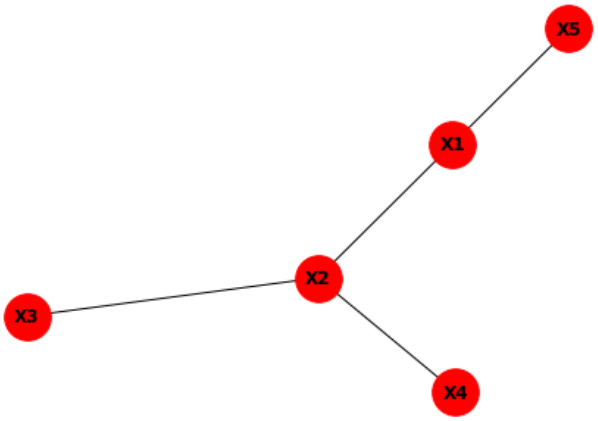

FIGURE 6 – Exemple d'arbre en sortie de notre algorithme avec le module networkx

II.2 - Analyse de notre algorithme

II.2.1 - Calcul de complexité

Soit n le nombre de sommets, on suppose que $n = O(m)$ avec m, le nombre d'arcs. On a donc que ACPM s'exécute en $O(mlog(m))$ car c'est là, la complexité de notre algorithme de tri. On a considéré que c'était le cas de tri le plus rapide comme le tri fusion.

Les variables étant discrètes, supposons que le maximum au carré du nombre de valeurs prises par une variable donnée est un $O(d)$ avec d, la longueur de notre base de données. L'algorithme de Chow-Liu va donc s'exécuter en $O(n^2d)$.

Cette courbe semble valider notre hypothèse car elle est plus ou moins linéraire :

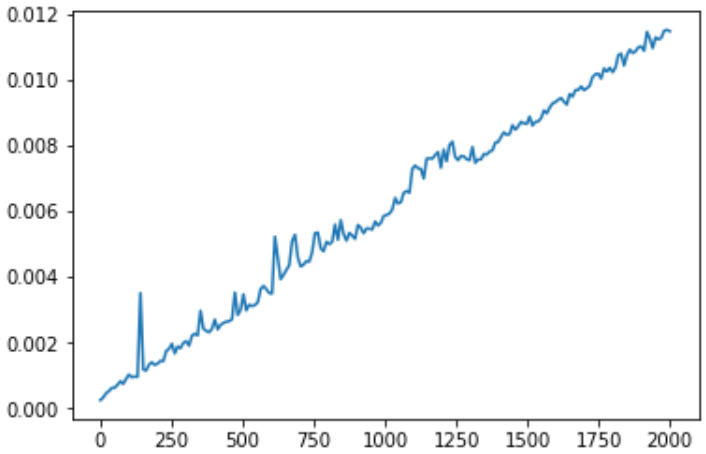

FIGURE 7 – Temps d'exécution de l'algorithme Chowliu en fonction de la taille de la base de donnée

Les aspérités dans la linéarité s'expliquent probablement par un cas défavorable de tri de notre liste. Cependant, on remarque que ces pics s'atténuent d'autant plus que d est grand.

II.2.2 - Test sur un jeu de données synthétiques

Pour tester notre algorithme, nous l'essayons d'abord sur une base de donnée de notre création. Posons n=5 avec des données de longueur 6000. Nous générons des variables aléatoires discrètes avec le module randint.

Création d'un jeu de données synthétique

Ici, P(X1,X2,X3,X4,X5)=P(X1)P(X2|X1)P(X3|X2)P(X4|X2)P(X5|X1)

```
import unittest
from random import randint

D=[]
M=[]
for i in range (6000):
    x1=randint(1,2)
    x2=x1+randint(1,2)
    x3=x2+randint(1,2)
    x4=randint(1,4)+x2
    x5=x1+randint(1,7)
    D.append([x1,x2,x3,x4,x5])
    M.append([x2,x1,x3,x4,x5])


class TestChowLiu(unittest.TestCase):
    def test_classique(self):
        V=Chowliu(['X1','X2','X3','X4','X5'],D)
        self.assertEqual([('X2', 'X1'), ('X3', 'X2'), ('X4', 'X2'), ('X5', 'X1')],V)


if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.038s

OK
```

FIGURE 8 – Test sur une base de donnée synthétique simple

Pour un dataset de longueur 6000, notre algorithme trouve le bon arbre. Ce test unitaire a été réalisé plusieurs fois avec des variables gaussiennes :

```
from random import gauss
D=[]
for i in range (6000):
    x1=gauss(10,15)
    x2=x1+gauss(10,20)
    x3=x2+gauss(100,15)
    x4=x2+gauss(0,4)
    x5=x1+gauss(0,2)
    D.append([x1,x2,x3,x4,x5])

class TestChowLiu(unittest.TestCase):
    def test_classique(self):
        V=Chowliu(['X1','X2','X3','X4','X5'],D)
        self.assertEqual([('X2', 'X1'), ('X3', 'X2'), ('X4', 'X2'), ('X5', 'X1')],V)


if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
F
======================================================================
FAIL: test_classique (__main__.TestChowLiu)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-70-7a621e52bafd>", line 18, in test_classique
    self.assertEqual([('X2', 'X1'), ('X3', 'X2'), ('X4', 'X2'), ('X5', 'X1')],V)
AssertionError: Lists differ: [('X2', 'X1'), ('X3', 'X2'), ('X4', 'X2'), ('X5', 'X1')] != [('X5', 'X1'), ('X5', 'X2')
, ('X5', 'X3'), ('X5', 'X4')]

First differing element 0:
('X2', 'X1')
('X5', 'X1')
```

FIGURE 9 – Test sur une base de donnée synthétique simple

Nous avons toujours obtenu des résultats négatifs peu importe la longueur de notre base de données. Ceci confirme bien notre hypothèse théorique selon laquelle les variables aléatoires doivent être discrètes.

Nous avons ensuite voulu étudier l'influence de la longueur de notre dataset sur la pertinence de nos résultats en reprenant le premier test. Nous avons obtenu la courbe suivante :

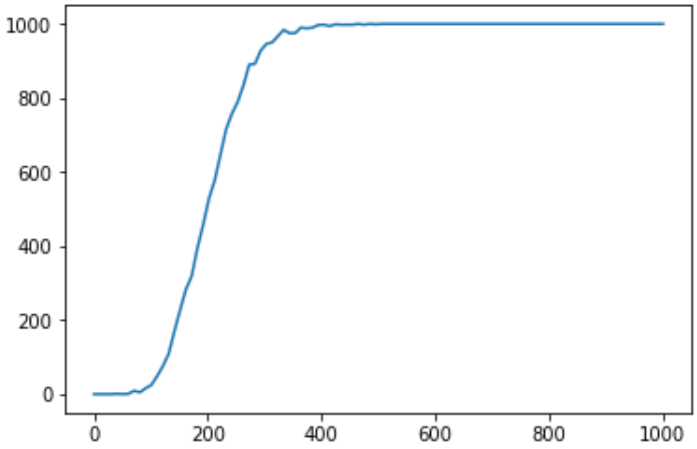

FIGURE 10 – Nombre de résultats corrects sur 1000 tentatives en fonction de la taille de notre base de donnée

Ainsi, on remarque que la précision de notre algorithme croît avec la longueur de notre base

de données ce qui semble logique puisqu'elle permet un meilleure approximations des lois des variables aléatoires.

II.3 - Limites et amélioration

II.3.1 - Problèmes rencontrés

Rappelons que nous avons dans les hypothèse de notre théorie que toutes les bases de données étaient pleines. Or dans l'application à de réelles bases de données, il est très rare que cette hypothèse soit vérifiée.

Par ailleurs un autre de nos souci a été la corrélation inexplicable de toutes les variables avec la variable "nombre de minutes jouées". Nous n'avons pas d'explication à ce résultat, nous pensions que cela était due à l'ordre de grandeur très différent, mais même après avoir normalisée la colonne, le résultat est le même.

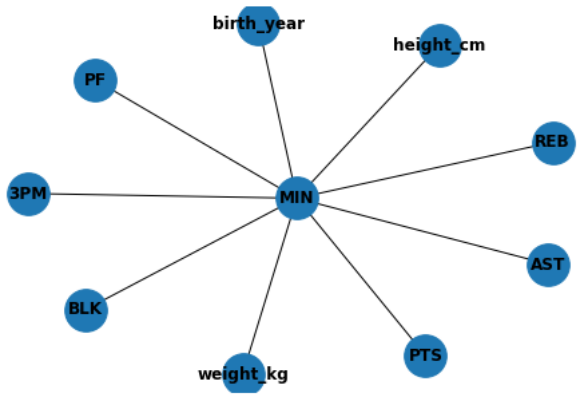

FIGURE 11 – Corrélation étrange

II.3.2 - Solution

Dès lors notre objectif est de combler les valeurs manquantes avec des valeurs potentiellement proches de la réalité. Pour cela nous devons coder un algorithme d'imputation, plusieurs méthodes existent. Nous avons choisit d'utiliser la méthode des "K plus proches voisins". Nous avons cette fois-ci utilisé un module déjà implémenté par sklearn.

Principe : Cette stratégie consiste à remplacer chaque série de valeurs manquantes par la moyenne des valeurs de cette variable pour plusieurs individus dont les caractéristiques sont les plus semblables possibles.

III - Application à notre problématique

Nous avons une base de donnée qui liste toutes les informations sportives sur des joueurs de basket.

III.1 - Analyse de la base

Nous avons trouvé une base assez large de taille 53949*34 pour que l'usage de notre algorithme de data mining soit justifié. Il y a donc 34 variables aléatoires 53949 observations par variable. On peut considérer que la base est assez grande pour utiliser l'algorithme. Cependant certaines variables aléatoires ne sont pas intéressantes comme l'équipe dans laquelle le joueur évolue ou la nationalité . On choisit donc de se concentrer sur 9 variables qui semblent intuitivement être dépendantes. Par exemple, il semble logique que le nombre de points par match soit corrélé au nombre de tirs à trois points réussis de manière plus directe que son poids et son nombre de passes décisives.

	3PM	weight_kg	PF	REB	AST	BLK	PTS	birth_year	height_cm
0	0	147.0	255	1078	299	239	2344	1972.0	216.0
1	95	100.0	263	476	322	92	2107	1977.0	198.0
2	2	120.0	229	779	304	71	2095	1963.0	206.0
3	89	75.0	162	267	328	5	1989	1975.0	183.0
4	177	82.0	178	529	732	18	1982	1968.0	193.0
5	83	99.0	188	315	365	36	1939	1974.0	198.0
6	34	102.0	190	490	385	43	1906	1972.0	203.0
7	30	109.0	205	956	401	126	1857	1976.0	211.0
8	99	102.0	171	518	438	32	1855	1973.0	201.0
9	27	111.0	264	787	345	128	1834	1973.0	206.0

FIGURE 12 – 10 premières observations des 9 variables aléatoires sélectionnées

De plus, la base n'est pas pleine comme le requiert l'hypothèse de l'algorithme nous avons donc comblé les trous grâce à l'algorithme d'imputation par les k plus proches voisins décrit précédemment III.3.2. Cependant on remarque que les résultats ne diffèrent que peu avec ou sans les valeurs manquantes.

III.2 - Résultats de l'algorithme et comparaison aux attentes

III.2.1 - Information mutuelle

Après avoir implémenté l'algorithme nous avons affiché la liste triée comportant les informations mutuelles : $L = [('BLK',' REB'), ('PTS',' 3PM'), ('PTS',' AST'), ('PTS',' PF'), ('PTS',' REB'), ('PTS',' weight'), ('birthyear',' PTS'), ('height',' weight')]$.

La liste semble cohérente mais traçons le graphe associé pour avoir une plus grande lisibilité

III.2.2 - Graphe

On trace enfin le graphe lié aux résultats donnés par Chow-Liu. Les résultats obtenus ne sont pas très étonnants, mais ils ont le mérite d'être clairs. On voit bien que la taille est liée au poids. Par ailleurs, le nombre de points est lui dépendant, logiquement, du nombre de trois points, et de rebonds, mais aussi plus étonnement de nombre de fautes et de passes décisives. On peut supposer un profil de joueur "agressif" ou téméraire engendre fautes et points. L'âge est lui aussi lié aux performances sans grandes surprises.

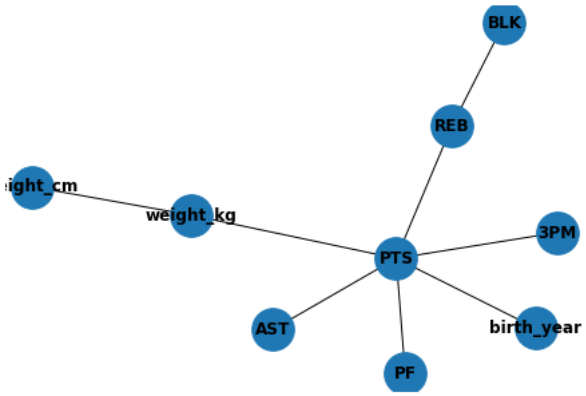

FIGURE 13 – Graphe final issu de notre algorithme

Conclusion

En somme, nous avons trouvé un arbre qui maximise les probabilités (Maximum spanning tree) pour notre dataset. De plus, on a approché la probabilité des variables en n'usant que peu de stockage grâce à l'algorithme. Nous sommes parvenus à tracer le graphe initialement cherché et notre algorithme semble performant. Reste cependant la présence de résultats inexpliqués comme celui exhibé dans cette section.

Concluons ce travail en expliquant ce qu'il nous a apporté. Tout d'abord, il nous faut dire que ce projet n'était pas du tout notre projet initial. Nous avons passé énormément de temps sur un sujet autour du paradoxe de Braess. Notre erreur a été de nous obstiner sur des notions qui, visiblement, dépassaient notre niveau. Nous avons donc perdu beaucoup de temps sans avancer significativement. Nous avons donc du changer de sujet et nous avons donc été pressé par le temps. Une des difficultés a été de choisir un sujet, nous avions beaucoup d'idées et de choix, mais il est parfois compliqué de ce rendre compte de la faisabilité et du niveau nécessaire à la réalisation de ceux-ci. Nous nous sommes rendu compte de nos impasses seulement après avoir commencé à travailler et à coder.

Bibliographie

[1] : Li Guoliang, Jun Yang, Joao Gama, *Databased systems for advanced applications*, Springer, 2019.

[2] : Marina Meila, Michael I. Jordan, *Learning with Mixtures of Trees*

[3] : Jee Vang, *A quick introdution to Chow-Liiu Algorithm.*

[4] : Timo Koski, *Lectures on Statistical Learning Theory for Chow-Liu Trees*, The 32nd Finnish Summer School on Probability Theory, 2010.

[5] : S. Panga, *Chapter 4 : Maximum Spanning Tree Modeling*, 2015.

[6] : Jason Eisner, *State-of-the-Art Algorithms for Minimum Spanning Trees : A Tutorial Discussion*, sur Université de Pennsylvanie, 1997

[7] : Article Wikipédia :*Chow Liu Tree.* https://en.wikipedia.org/wiki/Chow%E2%80%93Liu_tree