

Enseignement de l'informatique

POLYCOPIE

Fouille de Données Orientée Motifs

LEBBAH Yahia

Chargé de cours, Université d'Oran 1

Département Informatique, Faculté FSEA, Université d'Oran 1
B.P. 1524, El-M'Naouar Oran, Algérie

Version du 20 novembre 2023

Table des matières

1	Introduction à la fouille de données	5
1.1	Fouille de données : la vision commerciale	5
1.2	Fouille de données : la vision scientifique	6
1.3	Fouille de données : motivations	7
1.4	Définition de la fouille de données	8
1.5	Origines de la fouille de données	9
1.6	Tâches de la fouille de données	9
1.6.1	Classification	10
1.6.2	Partitionnement (clustering)	11
1.6.3	Découverte des motifs	12
1.6.4	Régression	13
1.6.5	Détection des anomalies	14
1.6.6	Apprentissage par renforcement	15
2	Fouille de motifs structurés	17
2.1	Fouille des motifs ensemblistes (itemset)	17
2.1.1	Contexte et définitions - Motifs fréquents	17
2.1.2	Préliminaires mathématiques	20
2.1.3	Algorithmes de recherche des motifs ensemblistes fréquents	23
2.1.3.1	Algorithme Apriori	25
2.1.3.2	L'approche Patern-Growth pour la fouille ensembliste	27
2.1.3.3	L'approche LCM pour la fouille ensembliste	36
2.1.4	Extraction des règles d'association	38
2.2	Fouille des motifs séquentiels	40
2.2.1	Extraction de motifs séquentiels	40
2.2.2	Projection préfixée et bases projetées	42
2.2.2.1	Illustrations de l'algorithme PrefixSpan sur des séquences d'items	47
2.2.2.2	Illustrations de l'algorithme PrefixSpan sur des séquences d'itemsets	48
2.3	Analyse formel de concepts et treillis de Galois	50
2.3.1	Contexte formel	51
2.3.2	Connexion de Galois	51
2.3.3	Fermeture	52
2.3.4	Concept formel	52
2.3.5	Treillis de Galois	53

2.3.6	Algorithme de construction des treillis de Galois	53
2.4	Fouille de données déclarative	54
2.4.1	Extraction de motifs ensemblistes sous-contraintes	54
2.4.2	Extraction de motifs séquentiels sous-contraintes	55
2.5	Fouille de graphes	58
2.5.1	Basic definitions and concepts	58
2.5.2	Frequent subgraph mining	63
2.6	Partitionnement avec contraintes (constrained clustering)	64
2.6.1	Clustering conceptuel	64
2.6.2	Co-clustering	64
2.6.3	Soft clustering	64
2.6.4	64
3	Fouille de stratégies et apprentissage par renforcement	65
4	Apprentissage par renforcement	67
4.1	Introduction	67
4.2	Processus de décision Markovien	67
4.2.1	Stratégies et motifs stratégiques	69
4.2.2	Critère d'optimalité	70
4.2.3	Fonction d'évaluation et équations de Bellman	70
4.3	Résoudre un MDP	71
4.3.1	Solutions basées sur un modèle prédéfini : programmation dynamique (DP)	71
4.3.2	Solutions basées sur un modèle d'environnement : apprentissage par renforcement (RL)	72
4.3.2.1	Apprentissage par différence temporelle, TD (Temporal Difference learning)	73
4.3.2.2	Méthode Monte-Carlo	74
4.3.3	Algorithmes de bandit	74
4.3.4	Le dilemme exploration/exploitation	74
4.4	Illustration sur un jeu	74
5	Fouille d'images	75
5.1	Rappel : réseaux convolutifs CNN	75
5.2	Détection d'objets	75
5.3	Tracking d'objets	75
5.4	75
6	Fouille de données spatiotemporelles	77
6.1	Modélisation des séries temporelles	77
6.1.1	Comprenez les variations saisonnières	81
6.1.2	Deux moyennes mobiles classiques	85
6.1.3	Utiliser des moyennes mobiles pour corriger des variations saisonnières	86
6.1.4	Evaluer les propriétés d'une moyenne mobile	86
6.1.5	Algorithmes de traitement des moyennes mobiles	87

6.1.6	La notion de bruit blanc	91
6.1.7	Une notion fondamentale : la stationnarité	91
6.1.8	Les processus AR, MA et ARMA	92
6.1.8.1	Les processus AR	92
6.1.8.2	Les processus MA	92
6.1.8.3	Les processus ARMA	93
6.1.8.4	Les processus ARIMA, SARIMA,	93
6.2	Fouille des séries temporelles : algorithme SAX	93
6.3	Fouille de trajectoires	94
6.3.1	Similarité entre deux trajectoires : distance de Fréchet	97
6.3.1.1	Decision problem	97
6.3.1.2	Frechet distance	100
6.3.1.3	Discrete Frechet	101
6.3.2	Mapping d'une trajectoire sur une carte	102
7	Compléments et applications	103
7.1	Détection d'anomalies (outlier detection)	103
7.1.1	Valeurs manquantes et valeurs aberrantes ?	104
7.1.2	Analyse des valeurs extrêmes	104
7.1.3	Méthodes probabilistes	104
7.1.4	Méthodes à base de clustering	104
7.1.5	Méthodes pour des données catégorielles	104
7.1.6	104
7.2	Apprentissage profond (deep learning) et découverte de motifs	104
7.3	Clustering : Manifold learning	104
7.4	Modèles de mélange et algorithme EM (Expectation Maximization)	104
7.4.1	Modèle de mélange	104
7.4.2	Modèle de mélange gaussien	104
7.4.3	Algorithme EM (Expectation Maximization)	104
7.5	Kernel density estimation (Estimation par noyau)	104
7.6	104
8	Rappels : classification et clustering	105
8.1	Régression linéaire [Biernat and Lutz, 2015]	105
8.2	Régression polynomiale [Biernat and Lutz, 2015]	107
8.3	Arbres de décision	107
8.3.1	Comment construire l'arbre de décision ? [Ph Preux 2011]	110
8.3.2	Stratégies de partitionnement de l'attribut	113
8.3.3	Problème du choix de l'attribut [Preux, 2006, Maimon and Rokach, 2005]	114
8.3.4	Exemple de déroulement	117
8.3.5	Elagage [Preux, 2006]	119
8.4	Forêts d'arbres décisionnels (Random forest)	120
8.5	Régression logistique	121
8.6	Validation des méthodes supervisées : illustration sur les arbres de décision [Preux, 2006]	122
8.6.1	Surapprentissage	125

8.7	Classeur bayésien	126
8.8	Classification par réseaux de neurones (Perceptron)	127
8.9	Classification par SVM	128
8.10	k -means	128
9	Travaux pratiques	129
9.1	Environnement SPMF	129
9.2	Fouille itemset	129
9.3	Fouille de séquences	130
9.4	Familiarisation Python	131
9.5	Prétraitement des données	139
9.5.1	Visualisation	139
9.5.2	Importation des données	141
9.5.3	Exploration des données	146
9.6	Prise en main sur les méthodes de fouille de données	152
9.6.1	Introduction à SciKit-Learn	153
9.6.2	Application à des chiffres écrits à la main	155
9.7	Fouille sur les réseaux	157
9.8	Systèmes de recommandations	160
9.8.1	Recommandations à base de similarité entre les utilisateurs . .	161
9.8.2	Recommandations à base de similarité entre items	163

Chapitre 1

Introduction à la fouille de données

Note : Le texte de ce document rassemble des extraits de [?].

1.1 Fouille de données : la vision commerciale

- Beaucoup de données sont collectées : données web, **commerce** électronique, données rotation stock, données des transactions des cartes bancaires, ...
- Les ordinateurs sont accessibles et plus **puissants**
- Exploiter les données pour **améliorer l'existant de l'économie** au sein de l'entreprise, d'un pays, du monde, ...



FIGURE 1.1 – Abondance des données économiques (Image extraite de [Slide "Introduction to data mining"])

1.2 Fouille de données : la vision scientifique

- Les données sont collectées et stockées avec une vitesse vertigineuse : données **génétiques**, données de simulation des systèmes **physiques**, données des capteurs des **satellites** et des télescopes balayant le **ciel**, ...
- Ces données sont de l'ordre des **Terabytes** !
- Les techniques traditionnelles issues des **BDs** sont impraticables sur ces données brutes
- La fouille de données peut **aider les scientifiques** en : classifiant et segmentant les données, proposition d'hypothèses sur la structure des données, ...

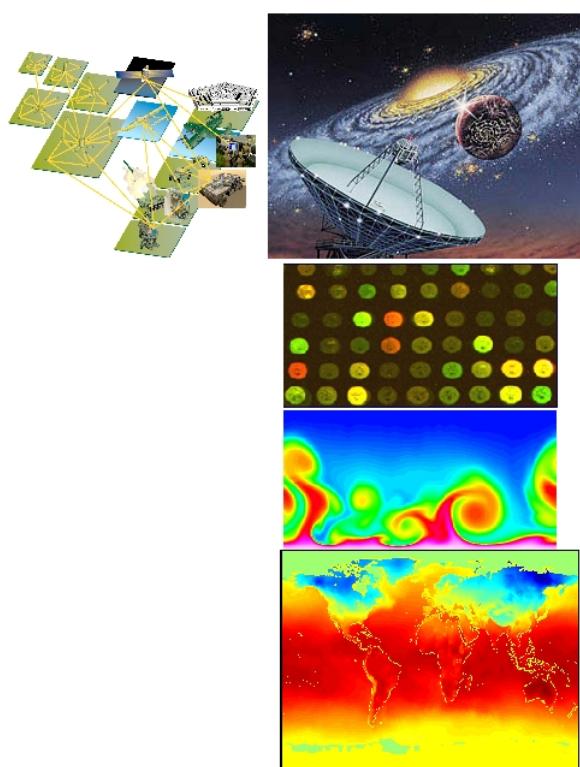


FIGURE 1.2 – Abondance des données scientifiques (Image extraite de [Slide "Introduction to data mining"]))

1.3 Fouille de données : motivations

- Il existe des **informations cachées** qui ne sont pas évidentes à repérer
- Le raisonnement humain a besoin de **plusieurs semaines** pour repérer des informations utiles
- Une quantité importante de ces données **n'est pas exploitée** ...¹

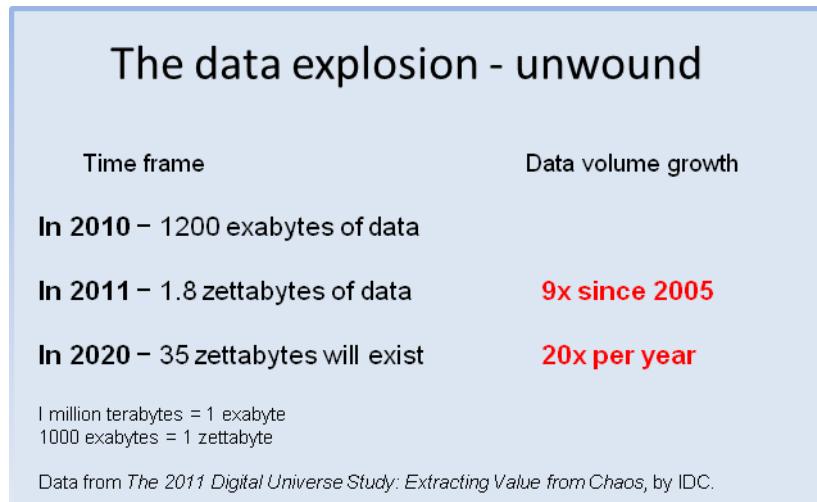


FIGURE 1.3 – Croissance des données

1. kilo/ 10^3 , méga/ 10^6 , giga/ 10^9 , téra/ 10^{12} , péta/ 10^{15} , exa/ 10^{18} , zetta/ 10^{21} , yotta/ 10^{24} .

1.4 Définition de la fouille de données

- Extraction d'information utiles à partir des données brutes
- Exploration et analyse avec des moyens automatiques ou semi-automatique de grandes masses de données pour extraire des motifs significatifs

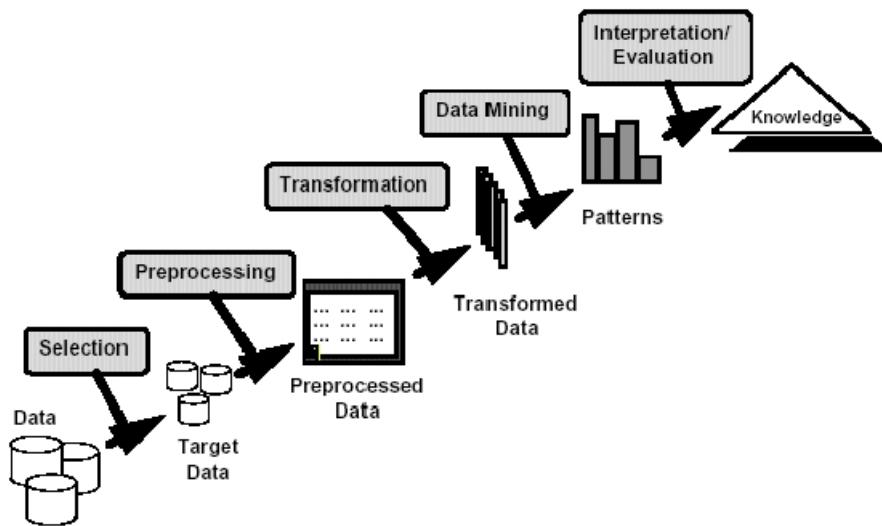


FIGURE 1.4 – Etapes de la fouille de données (Image extraite de [Slide "Introduction to data mining"]))

1.5 Origines de la fouille de données

- Les idées principales de la FDD proviennent de l'apprentissage automatique/IA, reconnaissance des formes, statistiques, SGBD
- Techniques traditionnelles insuffisantes pour traiter de grandes quantités de données ...

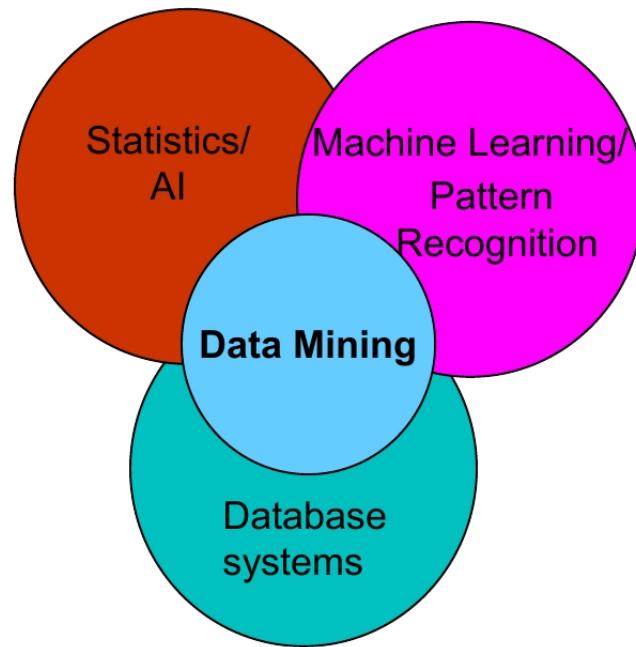


FIGURE 1.5 – Origines de la fouille de données (Image extraite de [Slide "Introduction to data mining"]))

1.6 Tâches de la fouille de données

- Classification et régression (apprentissage supervisé) : arbres de décision, réseaux de neurones, modèles de régression, modèles bayésiens...
- Partitionnement - Clustering (apprentissage supervisé) : k-means, SOM, manifold learning, ...
- Extraction de motifs : structurés, stratégies, ...
- *Applications* : Détection des anomalies ...

1.6.1 Classification

- Soit un ensemble X de N données où chaque donnée contient A attributs. Un attribut particulier A_c , dit classe, a un statut particulier de classification de la donnée.
- La classification vise à trouver un modèle d'une fonction de calcul de la classe A_c d'une donnée à partir des autres attributs.
- Objectif : pouvoir affecter correctement la classe d'une données non traitée : une base d'apprentissage est utilisée pour construire le modèle, et une base de test est utilisée pour évaluer la précision du modèle.

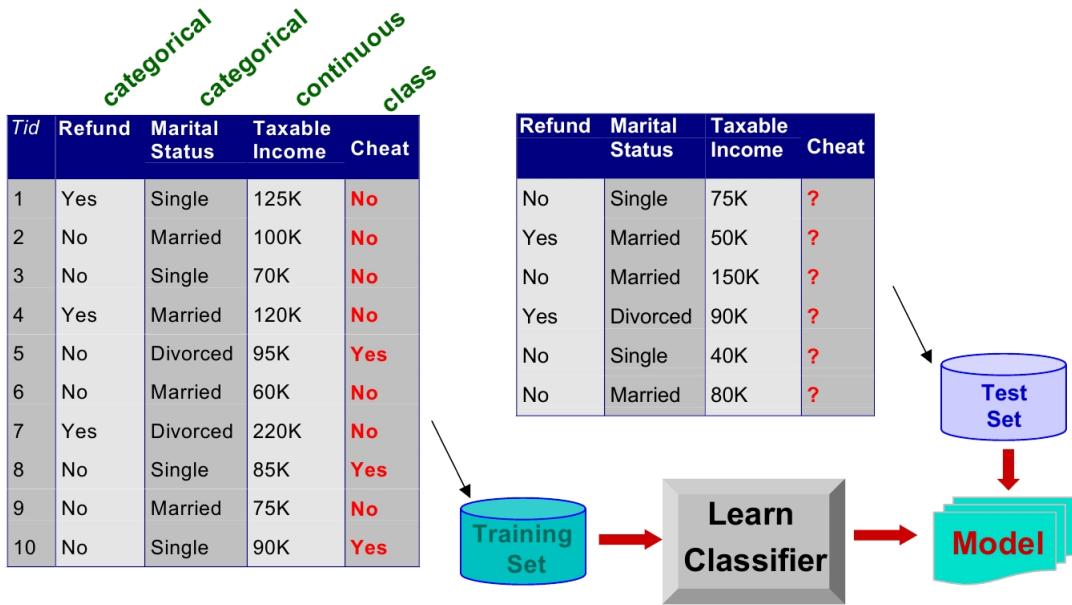


FIGURE 1.6 – Classification (Image extraite de [Slide "Introduction to data mining"]))) - (refund/rembourser, cheat/malveillant)

Applications :

- Données commerciales : cibler les clients, ...
- Détection des fraudes : prévoir les comportements frauduleux, ...
- classification des données scientifiques : astronomie, ...

1.6.2 Partitionnement (clustering)

- Soit un ensemble de données (points) caractérisés par un ensemble d'attributs et une mesure de similarité entre ces données : trouver un partitionnement des données, où chaque partition regroupe les points les plus similaires.
- Mesures de similarité : distance euclidiennes, autres mesures spécifiques.

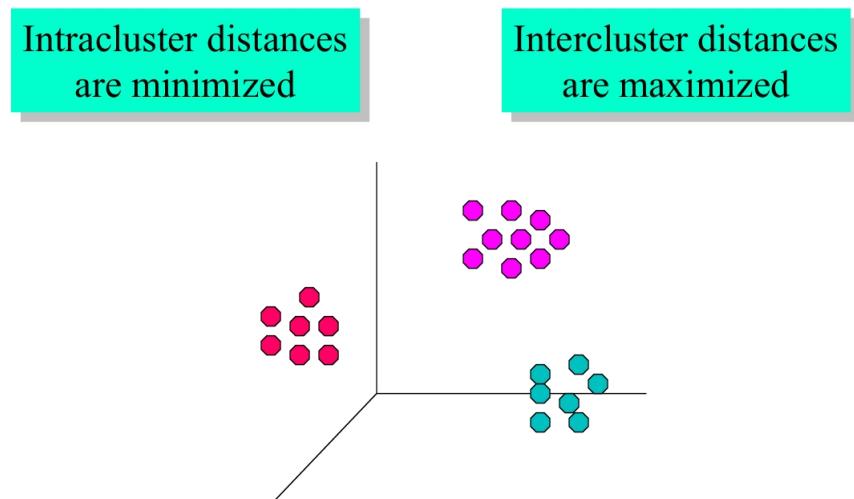


FIGURE 1.7 – Illustration du clustering (Image extraite de [Slide "Introduction to data mining"]))

Applications :

- Segmentation du commerce
- Partitionnement des documents
- Partitionnement du stock

1.6.3 Découverte des motifs

- Etant donné un ensemble de données contenant un nombre d'items d'une collection donnée : détecter des règles de dépendance entre les items.

TID	Items
1	Bread, Coke, Milk
2	Bread
3	Coke, Diaper, Milk
4	Bread, Diaper, Milk
5	Coke, Diaper, Milk

Rules Discovered:

{Milk} --> {Coke}

FIGURE 1.8 – Exemple d'extraction d'une règle d'association (Image extraite de [Slide "Introduction to data mining"])

Applications :

- Recommandations et incitations à l'achat
- Recommandations d'approvisionnement

1.6.4 Régression

- Prédire une valeur continue (réelle) d'une variable à partir des valeurs des autres variables, en supposant un modèle de dépendance linéaire ou non-linéaire.

— Pleinement étudiée en statistique et les réseaux de neurones.

Exemple :

- Prédire le volume des ventes en fonction des nouveaux produits
- Prédire la vitesse du vent en fonction de la température, de l'humidité, de la pression atmosphérique, ...
- ...

1.6.5 Détection des anomalies

- Détection des déviations à partir de comportements normaux.



FIGURE 1.9 – Illustration détection des anomalies (Image extraite de [Slide "Introduction to data mining"]))

Applications :

- Fraudes sur les cartes bancaires
- Détection d'intrusion dans les réseaux

1.6.6 Apprentissage par renforcement

- Apprendre les actions à prendre dans un environnement afin d'atteindre un objectif.

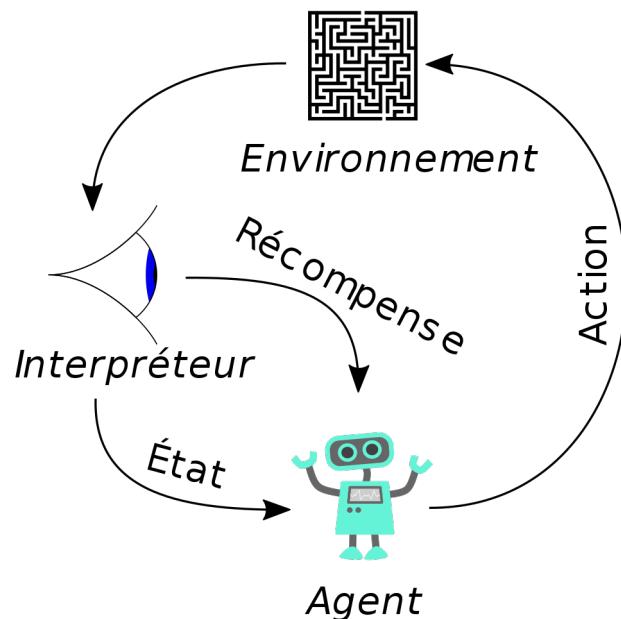


FIGURE 1.10 – Processus d'apprentissage par renforcement (Image extraite de [Wikipedia "Apprentissage par renforcement"]))

- D'une façon abstraite :
 - Données : environnement **Etats/Transitions**, processus Markovien, fonction de récompense sur les actions
 - On cherche un **Motif/Stratégie** pour atteindre un objectif.
- Comme au jeu :
 - Imaginons que vous jouez un nouveau jeu dans lequel vous passez par une étape d'apprentissage en vous disant :
 - Si chaque coup est bon ou mauvais
 - Si vous avez perdu ou gagné à la fin de chaque jeu
 - Le défi : en répétant le jeu, seriez-vous capable de gagner sans qu'on vous aide ?
- Applications :
 - Jeux ! (DeepMind / AlphaStar <https://deepmind.com>)
 - Technique la plus prometteuse pour faire les tâches intelligentes ...

Chapitre 2

Fouille de motifs structurés

2.1 Fouille des motifs ensemblistes (itemset)

¹ Cette section présente les principales notions liées à l'extraction de motifs fréquents.

2.1.1 Contexte et définitions - Motifs fréquents

- Soit $\mathcal{I} = \{I_1, \dots, I_n\}$ un ensemble de littéraux distincts appelés **items** et $\mathcal{T} = \{1, \dots, m\}$ un ensemble d'identifiants de **transactions**.
- Un motif ensembliste d'items est un sous-ensemble non vide de \mathcal{I} . Ces motifs sont regroupés dans le langage $\mathcal{L}_{\mathcal{I}} = 2^{\mathcal{I}} \setminus \emptyset$.
- Un jeu de données transactionnel est l'ensemble $\mathbf{r} \subseteq \mathcal{I} \times \mathcal{T}$.

Définition 1 (Couverture et fréquence) La couverture d'un motif x est l'ensemble de tous les identifiants de transactions qui le supportent : $\text{cov}_{\mathcal{T}}(x) = \{t \in \mathcal{T} | \forall i \in x, (i, t) \in r\}$. La fréquence d'un motif x représente le cardinal de sa couverture $\text{freq}_{\mathcal{T}}(x) = |\text{cov}_{\mathcal{T}}(x)|$.

L'extraction de motifs sous contraintes consiste à extraire les motifs satisfaisant une contrainte C à partir d'un jeu de données \mathbf{r} . La contrainte de **motifs fréquents** est définie en sélectionnant les motifs dont la fréquence est supérieure à un seuil donné min_{fr} . La contrainte de *taille* constraint le nombre d'items d'un motif x .

1. Des extraits sont pris de [Maamar et al., 2015]

transaction database

- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{b, c, e\}$
- 10: $\{a, d, e\}$

frequent item sets

	0 items	1 item	2 items	3 items
\emptyset : 10	$\{a\}$: 7 $\{b\}$: 3 $\{c\}$: 7 $\{d\}$: 6 $\{e\}$: 7	$\{a, c\}$: 4 $\{a, d\}$: 5 $\{a, e\}$: 6 $\{b, c\}$: 3 $\{c, d\}$: 4	$\{a, c, d\}$: 3 $\{a, c, e\}$: 3 $\{a, d, e\}$: 4	
			$\{c, e\}$: 4	
			$\{d, e\}$: 4	

- In this example, the minimum support is $s_{\min} = 3$ or $\sigma_{\min} = 0.3 = 30\%$.
- There are $2^5 = 32$ possible item sets over $B = \{a, b, c, d, e\}$.
- There are 16 frequent item sets (but only 10 transactions).

FIGURE 2.1 – Exemple d'une fouille dans une table transactionnelle (Image extraite de [Bourgelt, 2015]))

Une limite bien connue de l'extraction de motifs est le nombre de motifs produits qui peut être très grand. Les *représentations condensées* de motifs satisfaisant une contrainte C ont été introduites pour augmenter la rapidité d'exécution des algorithmes d'extraction de motifs et réduire le nombre de motifs obtenus. Pour les motifs ensemblistes et la contrainte de fréquence minimale, la représentation la plus usuelle est celle fondée sur les *motifs fermés*.

trans	items							
	A	B	C	D	E	F	G	H
t1	0	I	I	0	0	0	I	I
t2	I	0	0	I	0	0	0	0
t3	I	0	I	0	0	0	0	I
t4	I	0	0	0	I	I	0	0
t5	0	I	0	0	I	I	I	0

FIGURE 2.2 – Exemple d'une table transactionnelle

Définition 2 (Motif fermé) Un motif $x \in \mathcal{L}_{\mathcal{I}}$ est **fermé** par rapport à la fréquence ssi $\forall y \supsetneq x, freq(y) < freq(x)$.

Les motifs fermés structurent le treillis des motifs en *classes d'équivalence* (pour plus de détail voir le chapitre 2.3). Tous les motifs d'une même classe d'équivalence ont la même couverture. Les motifs fermés correspondent aux éléments maximaux (au sens de la taille des motifs) des classes d'équivalence.

Par ailleurs, l'utilisateur est souvent intéressé par la découverte de motifs plus riches satisfaisant des contraintes portant sur un ensemble de motifs et non plus un seul motif. Ces contraintes sont définies comme étant des contraintes sur des *ensembles de motifs* [De Raedt and Zimmermann, 2007] ou sur des motifs *n-aires* [Khiari et al., 2010] (voir le chapitre 2.4). Une approche intéressante consiste à traiter ce type de motifs via les top-*k*motifs.

Définition 3 (top-*k*motifs) Soit m une mesure d'intérêt et k un entier. Les top-*k*motifs est l'ensemble des k meilleurs motifs par rapport à la mesure m :

$$\{x \in \mathcal{L}_{\mathcal{I}} \mid freq_{\mathcal{T}}(x) \geq 1 \wedge \exists y_1, \dots, y_k \in \mathcal{L}_{\mathcal{I}} : \forall 1 \leq j \leq k, m(y_j) > m(x)\}$$

Exemple : interpréter les top-*k*motifs dans la table transactionnelle de la Figure 2.2.

2.1.2 Préliminaires mathématiques

- A **partial order** is a binary relation \leq over a set S which satisfies $\forall a, b, c \in S$:
 - $a \leq a$ (reflexivity)
 - $a \leq b \wedge b \leq a \Rightarrow a = b$ (anti-symmetry)
 - $a \leq b \wedge b \leq c \Rightarrow a \leq c$ (transitivity)
- A set with a partial order is called a **partially ordered set** (or **poset** for short).
- Let a and b be two distinct elements of a partially ordered set (S, \leq) .
 - if $a \leq b$ or $b \leq a$, then a and b are called **comparable**.
 - if neither $a \leq b$ nor $b \leq a$, then a and b are called **incomparable**.
- If all pairs of elements of the underlying set S are comparable, the order \leq is called a **total order** or a **linear order**.
- In a total order the reflexivity axiom is replaced by the stronger axiom:
 - $a \leq b \vee b \leq a$ (totality)

FIGURE 2.3 – Rappels sur la relation d'ordre (image de [Bourgelt, 2015])

Monotonicity in Calculus and Mathematical Analysis

- A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called **monotonically non-decreasing** if $\forall x, y : x \leq y \Rightarrow f(x) \leq f(y)$.
- A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called **monotonically non-increasing** if $\forall x, y : x \leq y \Rightarrow f(x) \geq f(y)$.

Monotonicity in Order Theory

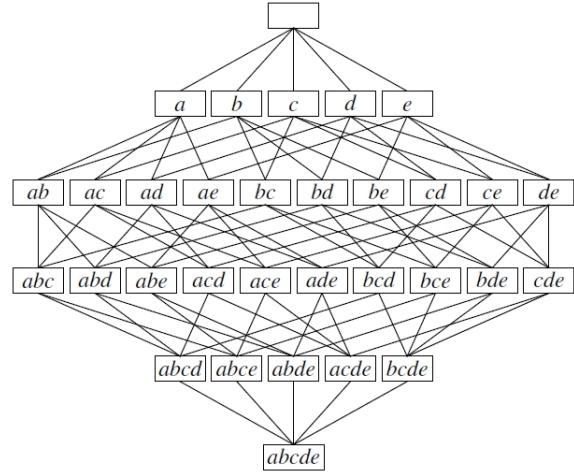
- Order theory is concerned with arbitrary partially ordered sets. The terms *increasing* and *decreasing* are avoided, because they lose their pictorial motivation as soon as sets are considered that are not totally ordered.
- A function $f : S \rightarrow R$, where S and R are two partially ordered sets, is called **monotone** or **order-preserving** if $\forall x, y \in S : x \leq_S y \Rightarrow f(x) \leq_R f(y)$.
- A function $f : S \rightarrow R$ is called **anti-monotone** or **order-reversing** if $\forall x, y \in S : x \leq_S y \Rightarrow f(x) \geq_R f(y)$.
- In this sense the **support** of item sets is **anti-monotone**.

FIGURE 2.4 – Rappels sur les propriétés de monotonie (image de [Bourgelt, 2015])

En conséquence des propriétés de monotonie (voir la Figure 2.3), la fonction support d'un itemset est anti-monotone.

- A finite partially ordered set (S, \leq) can be depicted as a (directed) acyclic graph G , which is called **Hasse diagram**.
- G has the elements of S as vertices.
The edges are selected according to:

If x and y are elements of S with $x < y$ (that is, $x \leq y$ and not $x = y$) and there is no element between x and y (that is, no $z \in S$ with $x < z < y$), then there is an edge from x to y .
- Since the graph is acyclic (there is no directed cycle), the graph can always be depicted such that all edges lead downward.
- The Hasse diagram of a total order (or linear order) is a chain.



Hasse diagram of $(2^{\{a,b,c,d,e\}}, \subseteq)$.

(Edge directions are omitted;
all edges lead downward.)

FIGURE 2.5 – Diagramme de Hasse sur les ensembles partiellement ordonnés (image de [Bourgelt, 2015])

2.1.3 Algorithmes de recherche des motifs ensemblistes fréquents

On peut naïvement penser à un algorithme exhaustif qui parcourt tous les sous-ensembles d'items et teste s'ils respectent la contrainte de fréquence (et autres contraintes). Bien évidemment, un tel algorithme serait très couteux et exponentiel. Pour éviter un tel parcours, nous ferons appel à la propriété suivante :

$$\forall I \subseteq \mathcal{I}, \forall J \supseteq I, freq_{\mathcal{T}}(J) \leq freq_{\mathcal{T}}(I).$$

Le principe : si un itemset est étendu, sa fréquence décroît.

Cette propriété est dite qualifiée de "propriété Apriori", en référence à l'algorithme Apriori [Agrawal and Srikant, 1994]. Elle est aussi qualifiée de propriété d'anti-monotonie (voir figures ci-dessous).

La contraposé est aussi vraie : tous les sous-ensemble d'un itemset fréquent sont fréquents.

Principes de recherche des itemsets fréquents :

- La procédure de recherche standard consiste en une approche d'énumération qui énumère les candidats et vérifie la satisfaction de la contrainte de fréquence.
- La procédure améliore l'approche naïve en exploitant la propriété d'anti-monotonie qui évite d'explorer les itemsets dont un de ses sous-ensembles n'est pas fréquent.
- L'espace de recherche est l'ensemble muni d'un ordre partiel $(2^{|\mathcal{I}|}, \subseteq)$.
- La structure d'ordre partiel permet d'éviter des candidats en exploitant la propriété d'anti-monotonie. L'approche est donc une approche descendante qui démarre du candidat vide, l'étend au fur et à mesure. On se prive de continuer toute branche descendante d'un candidat non-fréquent.

Hasse diagram with frequent item sets ($s_{\min} = 3$):
transaction database

- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{b, c, e\}$
- 10: $\{a, d, e\}$

Blue boxes are frequent item sets, white boxes infrequent item sets.

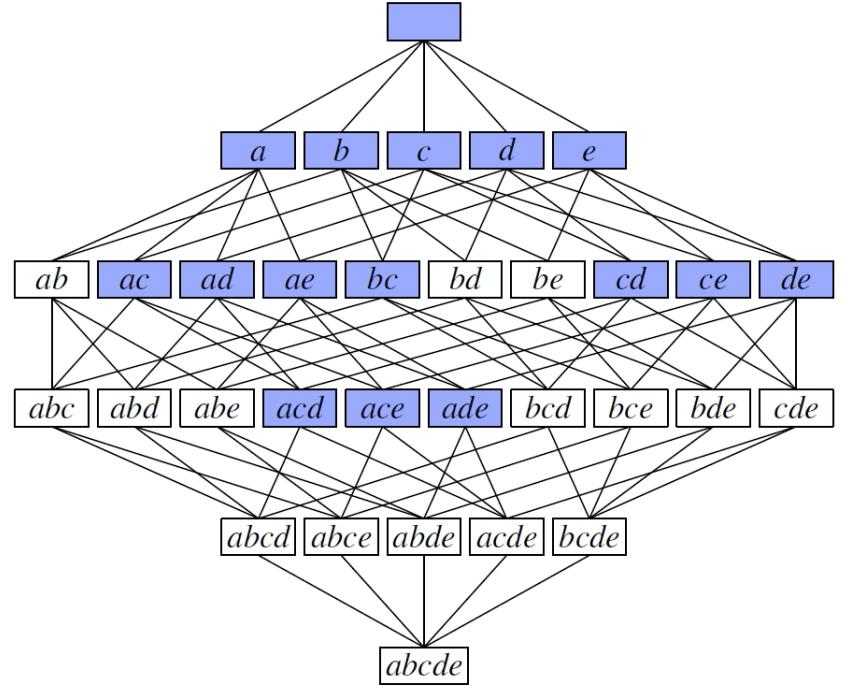


FIGURE 2.6 – Recherche de motifs ensemblistes en exploitant l'anti-monotonie (image de [Bourgelt, 2015])

2.1.3.1 Algorithme Apriori

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D , a database of transactions;
- min_sup , the minimum support count threshold.

Output: L , frequent itemsets in D .

Method:

```

(1)    $L_1 = \text{find\_frequent\_1-itemsets}(D);$ 
(2)   for ( $k = 2; L_{k-1} \neq \emptyset; k++$ ) {
(3)      $C_k = \text{apriori\_gen}(L_{k-1});$ 
(4)     for each transaction  $t \in D$  { // scan  $D$  for counts
(5)        $C_t = \text{subset}(C_k, t);$  // get the subsets of  $t$  that are candidates
(6)       for each candidate  $c \in C_t$ 
(7)          $c.\text{count}++;$ 
(8)     }
(9)      $L_k = \{c \in C_k | c.\text{count} \geq min\_sup\}$ 
(10)   }
(11)  return  $L = \cup_k L_k;$ 

procedure apriori_gen( $L_{k-1}$ :frequent  $(k - 1)$ -itemsets)
(1)  for each itemset  $l_1 \in L_{k-1}$ 
(2)  for each itemset  $l_2 \in L_{k-1}$ 
(3)  if ( $l_1[1] = l_2[1]$ )  $\wedge (l_1[2] = l_2[2])$ 
      $\wedge \dots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$  then {
(4)     $c = l_1 \bowtie l_2;$  // join step: generate candidates
(5)    if has_infrequent_subset( $c, L_{k-1}$ ) then
(6)      delete  $c;$  // prune step: remove unfruitful candidate
(7)      else add  $c$  to  $C_k;$ 
(8)  }
(9)  return  $C_k;$ 

procedure has_infrequent_subset( $c$ : candidate  $k$ -itemset;
                                $L_{k-1}$ : frequent  $(k - 1)$ -itemsets); // use prior knowledge
(1)  for each  $(k - 1)$ -subset  $s$  of  $c$ 
(2)  if  $s \notin L_{k-1}$  then
(3)    return TRUE;
(4)  return FALSE;

```

FIGURE 2.7 – Algorithme Apriori (image de [Han, 2005])

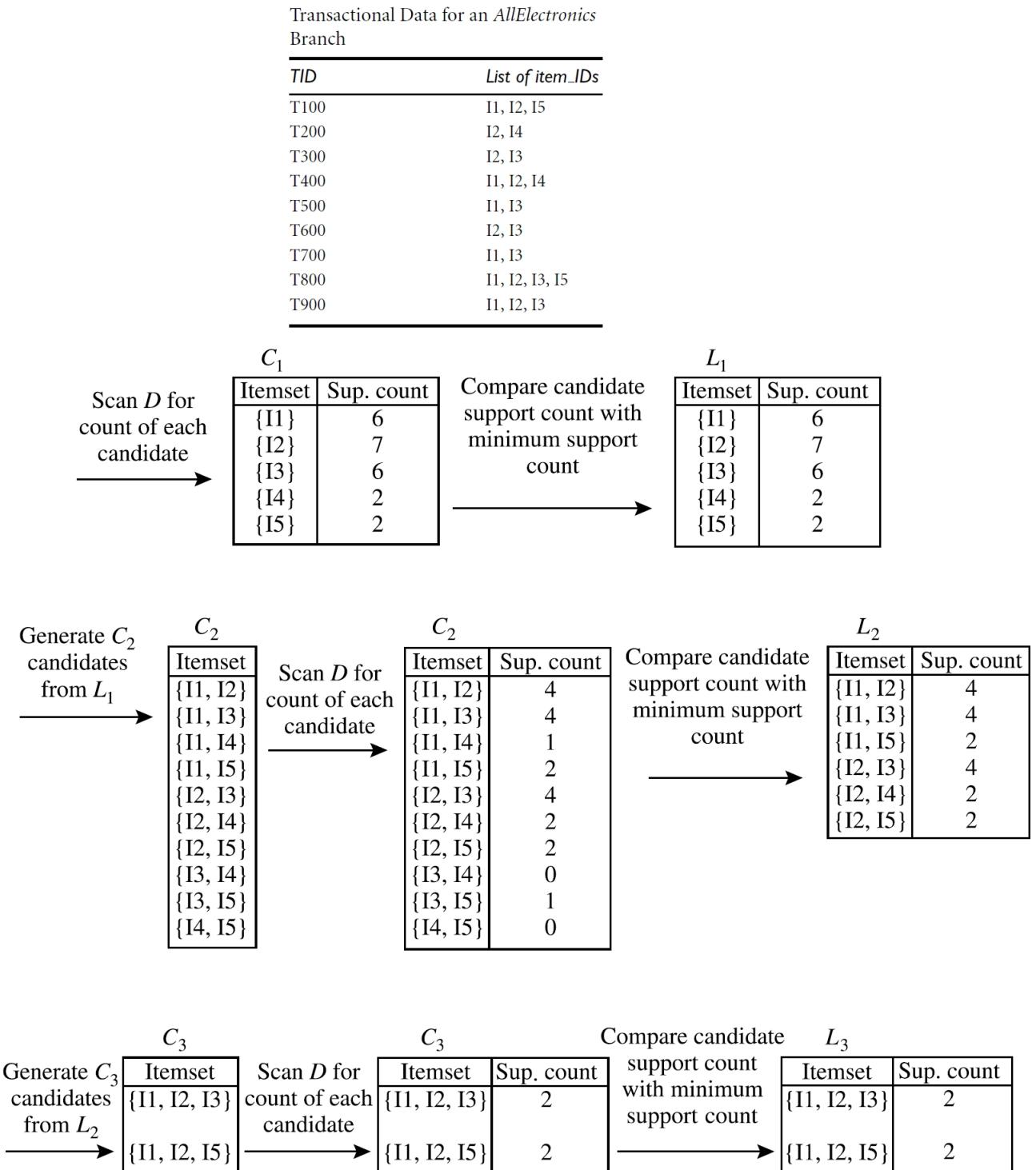


FIGURE 2.8 – Déroulement de l'algorithme Apriori (image de [Han, 2005])

2.1.3.2 L'approche Patern-Growth pour la fouille ensembliste

L'approche Apriori évite d'explorer plusieurs motifs non-fréquent grâce au principe d'anti-monotonie. Cependant, en pratique, il est très couteux pour deux raisons :

- Il génère un nombre énorme de candidats. Par exemple, s'il existe 10^4 1-itemsets fréquents, l'algorithme Apriori générera plus de 10^7 candidats 2-itemsets.
- Il a besoin de scanner à chaque fois toute la base de transactions et vérifier l'égalité entre un grand nombre de candidats. Il devient très cher de prendre à chaque fois un candidat et calculer sa fréquence en explorant toutes les transactions de la base.

La question : peut-on concevoir l'ensemble complet des itemsets fréquents en évitant le processus couteux de génération des candidats ? La méthode **frequent pattern growth**, dite **FP-growth**, qui adopte une approche divisor pour régner.

Finding Association Rules

Two-step approach:

1. Frequent Itemset Generation

- Generate all itemsets whose support $\geq \text{minsup}$

2. Rule Generation

- Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

FIGURE 2.9 – Règles d'association

Algorithme FP-growth

1. Balayer la base de transactions T une première fois
 - Créer L , la liste des items fréquents avec leur support
 - Trier L en ordre décroissant du support
2. Créer l'arbre N contenant une racine étiquetée « Null »
3. Procédure FP-growth(Fp-tree, $null$)
 - A. Si FP-tree contient un seul chemin P alors
 - Pour chaque combinaison β de P faire
 - Générer l'itemset $\beta = \cup \alpha$ de support = minimum des supports des nœuds de β
 - B. Sinon pour chaque a_i dans l'indexe de FP-tree faire
 - Générer l'itemset $\beta = a_i \cup \alpha$ de support = $a_i.support$
 - Construire l'itemset condition de base de β
 - Construire FP-tree $_{\beta}$
 - Si FP-tree $_{\beta} \neq 0$
 - FP-growth(FP-tree $_{\beta}$, β)

FIGURE 2.10 – Algorithme FP-Growth (image de <http://blog.khaledtannir.net/2012/07/lalgorithme-fp-growth-les-bases-13>)

<i>TID</i>	<i>Items bought</i>	<i>(ordered) frequent items</i>
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Steps:

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree

<i>Item</i>	<i>frequency head</i>	
f	e	4
c	b	4
a	d	3
b	a	3
m	f	3
p	c	3

FIGURE 2.11 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

Steps Contd. (Example)

- Scan of the first transaction leads to the construction of the first branch of the tree listing

(ordered) frequent items

$\{f, c, a, m, p\}$
 $\{f, c, a, b, m\}$
 $\{f, b\}$
 $\{c, b, p\}$
 $\{f, c, a, m, p\}$

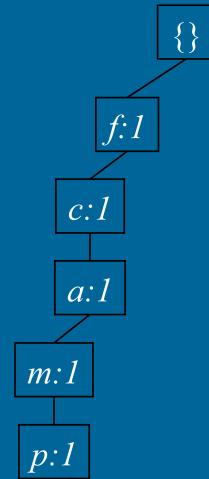


FIGURE 2.12 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

Steps Contd. (Example)

- Scan of the first transaction leads to the construction of the first branch of the tree listing
- Second transaction shares a common prefix with the existing path the count of each node along the prefix is incremented by 1
- Two new nodes are created and linked as children of $(a:2)$ and $(b:1)$ respec.

(ordered) frequent items

$\{f, c, a, m, p\}$

$\{f, c, a, b, m\}$

$\{f, b\}$

$\{c, b, p\}$

$\{f, c, a, m, p\}$

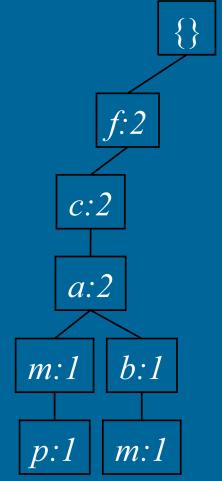


FIGURE 2.13 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

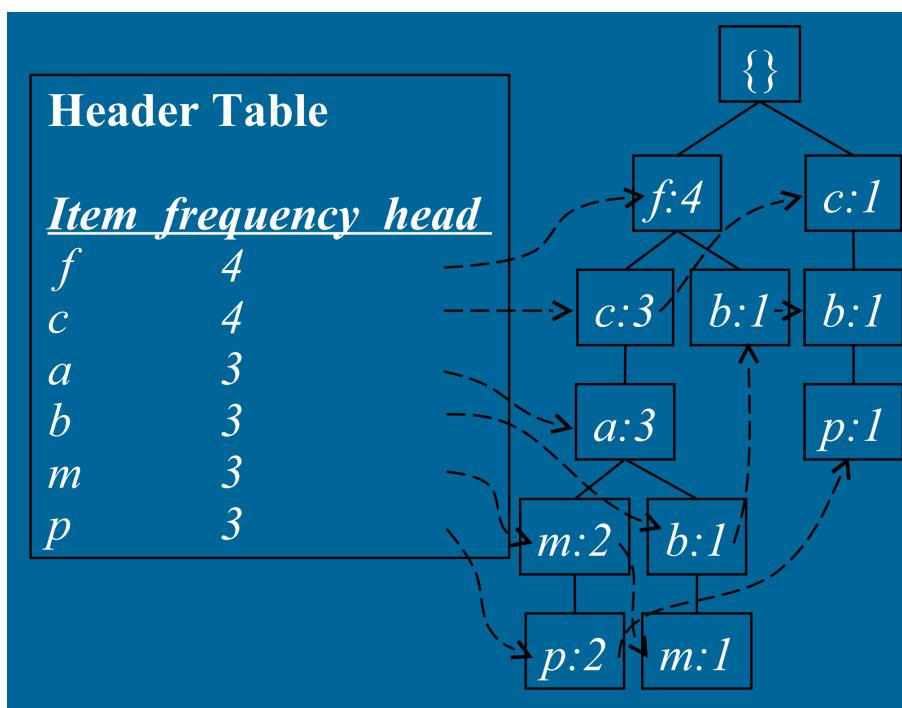


FIGURE 2.14 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

Mining frequent patterns of p

For node p , its immediate frequent pattern is $(p:3)$, and it has two paths in the FP-tree: $(f:4, c:3, a:3, m:2, p:2)$ and $(c:1, b:1, p:1)$

These two prefix paths of p , “ $\{(fcam:2), (cb:1)\}$ ”, form p 's conditional pattern base

Now, we build an FP-tree on P's conditional pattern base.

Leads to an FP tree with one branch only i.e. C:3 hence the frequent pattern associated with P is just CP

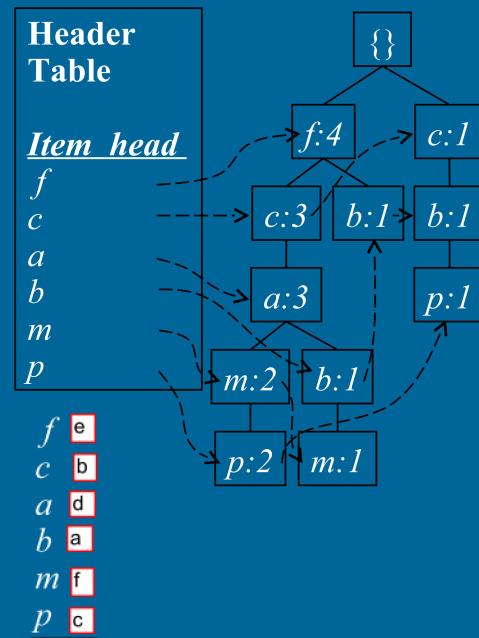


FIGURE 2.15 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

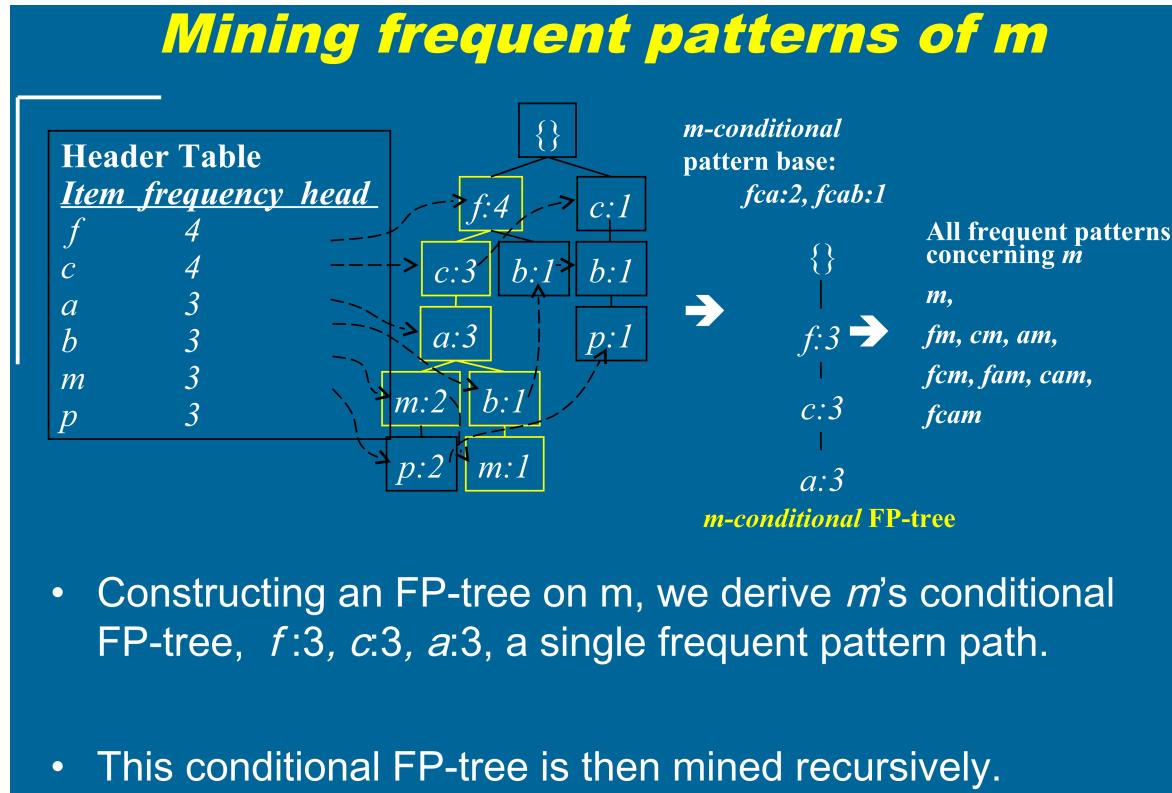


FIGURE 2.16 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

Mining Frequent Patterns by Creating Conditional Pattern-Bases

Item	Conditional pattern-base	Conditional FP-tree
p	$\{(fcam:2), (cb:1)\}$	$\{(c:3)\} p$
m	$\{(fca:2), (fcab:1)\}$	$\{(f:3, c:3, a:3)\} m$
b	$\{(fca:1), (f:1), (c:1)\}$	Empty
a	$\{(fc:3)\}$	$\{(f:3, c:3)\} a$
c	$\{(f:3)\}$	$\{(f:3)\} c$
f	Empty	Empty

FIGURE 2.17 – Illustration de l'algorithme FP-Growth (min-sup=3, image de https://fr.slideshare.net/Shani729/frequent-itemset-mining-using-pattern-growth-method?from_action=save)

2.1.3.3 L'approche LCM pour la fouille ensembliste

Enfin, Uno et al. [Uno et al., 2004] ont proposé LCM (Linear time Closed pattern Miner), un des algorithmes les plus performants en fouille de données ensemblistes.

Pour générer tous les motifs clos, on peut faire appel à une procédure naïve donnée dans la figure 2.18.

Algorithm Closure_version

1. $\mathcal{D} := \{\perp\}$
2. $\mathcal{D}' := \{ \text{Clo}(P \cup \{i\}) \mid P \in \mathcal{D}, i \in \mathcal{I} \setminus P \}$
3. **if** $\mathcal{D}' = \emptyset$ **then output** \mathcal{D} ; **halt**
4. $\mathcal{D} := \mathcal{D} \cup \mathcal{D}'$; **go to** 2

FIGURE 2.18 – Description d'un algorithme naïf de recherche de tous les clos [Uno et al., 2004]

Cette procédure naïve est extrêmement couteuse. LCM propose une stratégie dédiée qui tire profit de la structure particulière des clos pour les générer en temps linéaire du nombre de clos. Il utilise les deux représentations horizontales et verticales. Le noyau de LCM est une technique dite *prefix preserving closure extension* (PPCE), qui permet de générer un nouveau motif clos fréquent à partir d'un clos précédemment généré. Nous expliquons ce principe PPCE dans la suite. Soit le motif clos P .

Définition 4 (Fermeture) La fermeture d'un pattern P dans SDB, dénotée par $\text{Clos}(P)$, est l'ensemble des items communs de sa couverture $\mathcal{T}_{SDB}(P)$, c'est-à-dire, $\text{Clos}(P) = \mathcal{I}_{SDB}(\mathcal{T}_{SDB}(P))$. Un motif est clos si et seulement si $\text{Clos}(P) = P$.

Définition 5 Soit $P(i) = P \cap \{1, \dots, i\}$ le sous-ensemble de P des items qui ne sont pas plus grand que i .

Définition 6 (core index) L'indice core (core index) de P , dénoté $\text{core}(P)$, est l'indice minimum i tel que $\mathcal{T}_{SDB}(P(i)) = \mathcal{T}_{SDB}(P)$.

Définition 7 (PPCE) Un motif Q est PPCE de P si :

- $Q = \text{Clos}(P \cup \{i\})$ pour un certain $i \notin P$ (Q est obtenu en ajoutant i à P puis prendre sa fermeture)
- $P(i - 1) = Q(i - 1)$,
- $i > \text{core}(P)$.

transaction database

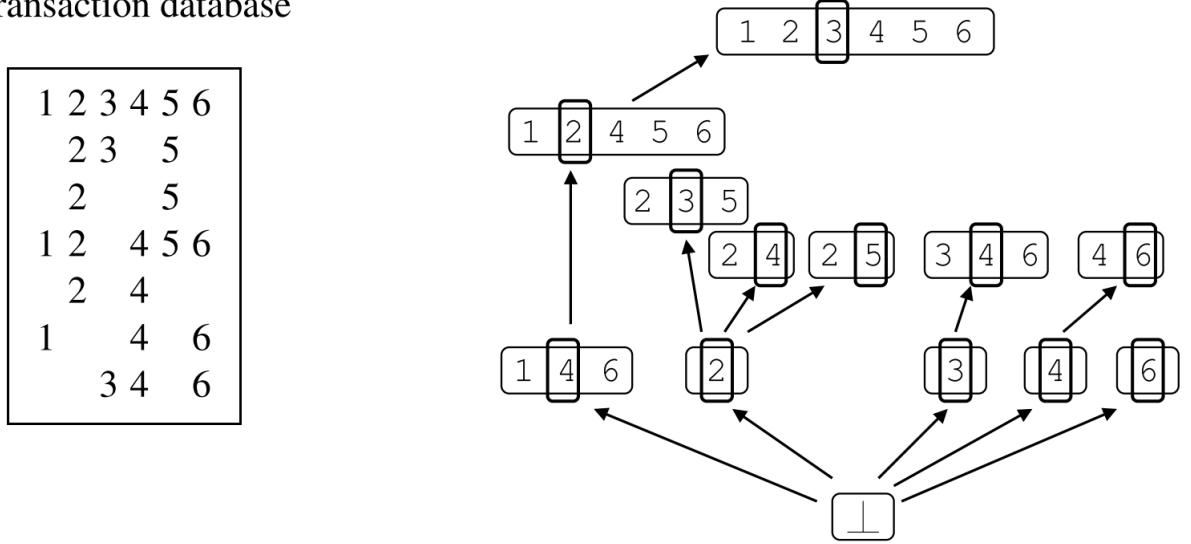


FIGURE 2.19 – Exemple de motifs clos et leurs extensions PPC [Uno et al., 2004]

La complétude de PPCE est assurée par la propriété suivante :

Proposition 1 ([Uno et al., 2004]) *Si Q est un motif clos non-vide, alors il existe uniquement un itemset clos P tel que Q est PPCE de P .*

Le processus de fouille de LCM est une recherche en profondeur où à chaque noeud, nous avons un motif clos P . LCM utilise la technique PPCE comme une stratégie de branchement pour passer d'un lotif clos à un autre clos en ajoutant des items. Avec cette stratégie particulière de recherche en profondeur, LCM réussit à énumérer rapidement l'ensemble des clos. Dans la Figure 2.19, on illustre le comportement de LCM construisant l'arbre de recherche où chaque noeud est clos par PPCE d'un autre unique clos.

Algorithm LCM(\mathcal{T} :transaction database, θ :support)

1. **call** ENUM_CLOSED PATTERNS(\perp);

Procedure ENUM_CLOSED PATTERNS(P : frequent closed pattern)

2. **if** P is not frequent **then Return**;

2. **output** P ;

3. **for** $i = core_i(P) + 1$ **to** $|\mathcal{I}|$

4. $Q = Clo(P \cup \{i\})$;

5. **If** $P(i - 1) = Q(i - 1)$ **then** // Q is a ppc-extension of P

6. **Call** ENUM_CLOSED PATTERNS(Q);

7. **End for**

FIGURE 2.20 – Description de l'algorithme LCM [Uno et al., 2004]

Proposition 2 ([Uno et al., 2004]) *Etant donné une base SDB, l'algorithme LCM énumère chaque clos fréquent en $O(|\mathcal{T}_{SDB}(P)| \times I)$ avec une occupation mémoire de $O(SDB)$.*

2.1.4 Extraction des règles d'association

Définition 8 (règle d'association) *Une règle d'association R est la donnée de deux motifs A et B . On la note $R = A \rightarrow B$.*

Définition 9 (règle d'association) *Le support d'une règle $A \rightarrow B$ est $\text{support}(A \rightarrow B) = \text{support}(A \cup B)$.*

La confiance d'une règle $A \rightarrow B$ est

$$\text{confiance}(A \rightarrow B) = \frac{\text{support}(A \cup B)}{\text{suppor}(A)} = \frac{|\text{couv}(A) \cap \text{couv}(B)|}{|\text{couv}(A)|}.$$

Notons que $\text{confiance}(A \rightarrow B) \in [0, 1]$.

$\text{confiance}(A \rightarrow B) = 1$ signifie que tous les objets possédant le motif A possèdent le motif B .

$\text{confiance}(A \rightarrow B) = 0$ signifie qu'aucun des objets possédant le motif A ne possède le motif B .

$\text{support}(A \rightarrow B)$ est la probabilité que x possède le motif A et que x possède le motif B .

On peut aussi rapprocher la confiance des probabilités conditionnelles. En effet, $\text{confiance}(A \rightarrow B)$ est la probabilité que x possède le motif B sachant que x possède le motif A .

Définition 10 *Une règle d'association R est valide si elle vérifie :*

- $\text{support}(A \rightarrow B) \geq \sigma_s$ et
- $\text{confiance}(A \rightarrow B) \geq \sigma_c$,

où $\sigma_s, \sigma_c \in [0, 1]$ sont des valeurs seuils prédéfinies.

Une règle d'association est exacte si $\text{confiance}(R) = 1$. Dans le cas contraire, R est approximative.

Ce n'est pas parce qu'une règle est exacte qu'elle est certaine : cela signifie seulement qu'il n'existe aucun contre-exemple dans la base de données fouillée à cette règle. Pour générer les règles d'association, voir l'algorithme de la Figure 2.21.

Finding Association Rules

Two-step approach:

1. Frequent Itemset Generation

- Generate all itemsets whose support $\geq \text{minsup}$

2. Rule Generation

- Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

FIGURE 2.21 – Règles d'association

2.2 Fouille des motifs séquentiels

² Cette section introduit les définitions usuelles utilisées respectivement en fouille de séquences et en programmation par contraintes.

2.2.1 Extraction de motifs séquentiels

Soit \mathcal{I} un ensemble fini de n éléments (*items*). Le langage de séquences correspond à $\mathcal{L}_{\mathcal{I}} = \mathcal{I}^n$ où $n \in \mathbb{N}^*$.

Définition 11 (séquence, base de séquences) Une séquence s définie sur $\mathcal{L}_{\mathcal{I}}$ est une liste ordonnée d'éléments $\langle s_1 s_2 \dots s_n \rangle$, où $s_i \in \mathcal{I}$ est un itemset, $1 \leq i \leq n$. n est appelé la longueur de la séquence s . Une base de séquences SDB est un ensemble de tuples (sid, s) , où s est la séquence et sid représente son identifiant.

Définition 12 (sous-séquence, la relation \preceq) Une séquence $s = \langle s_1 \dots s_m \rangle$ est une sous-séquence d'une séquence $s' = \langle s'_1 \dots s'_n \rangle$ (ou s' est une sur-séquence de s), notée $(s \preceq s')$, ssi $m \leq n$ et il existe des entiers $1 \leq j_1 \leq \dots \leq j_m \leq n$ tels que $s_i \subseteq s'_{j_i}$ pour tout $1 \leq i \leq m$. On dit aussi que s est contenue dans s' . Un tuple (sid', s') contient une séquence s si $s \preceq s'$.

Définition 13 (couverture, support) Soient SDB une base de séquences et s une séquence. La couverture de s dans SDB est l'ensemble de tous les tuples de SDB qui contiennent s : $cover_{SDB}(s) = \{(sid, s') \in SDB \mid s \preceq s'\}$, et son support est défini par $sup_{SDB}(s) = |cover_{SDB}(s)|$.

Définition 14 (motif séquentiel) Soit un seuil de support minimal $minsup$. On dit qu'une séquence s est fréquente dans une base SDB , si son support contient au moins $minsup$ éléments : $sup_{SDB}(s) \geq minsup$. s est appelée motif séquentiel [Agrawal and Srikant, 1995].

sid	Séquence
1	$\langle ABCBC \rangle$
2	$\langle BABC \rangle$
3	$\langle AB \rangle$
4	$\langle BCD \rangle$

TABLE 2.1 – SDB_1 : Un exemple de base de séquences.

Exemple 1 La Table 2.1 représente une base composée de 4 séquences, avec $\mathcal{I} = \{A, B, C, D\}$. Soit la séquence $s = \langle AC \rangle$, alors $cover_{SDB_1}(s) = \{(1, s_1), (2, s_2)\}$. Si on considère $minsup = 2$, alors $s = \langle AC \rangle$ est un motif séquentiel car $sup_{SDB_1}(s) \geq 2$.

Définition 15 (Extraction de motifs séquentiels (EMS)) Etant donnés un seuil de support minimal $minsup \geq 1$ et une base de séquences SDB , le problème d'extraction des motifs séquentiels consiste à trouver toutes les séquences s telles que $sup_{SDB}(s) \geq minsup$.

2. Des extraits sont pris de [?]



Subsequence vs. super sequence

- Given two sequences $\alpha = \langle a_1 a_2 \dots a_n \rangle$ and $\beta = \langle b_1 b_2 \dots b_m \rangle$
- α is called a **subsequence** of β , denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$
- β is a **super sequence** of α



$\beta = \langle a(abc)(ac)d(cf) \rangle$

$\alpha_1 = \langle aa(ac)d(c) \rangle$

$\alpha_2 = \langle (ac)(ac)d(cf) \rangle$

$\alpha_3 = \langle ac \rangle$

$\beta = \langle a(abc)(ac)d(cf) \rangle$

$\alpha_4 = \langle df(cf) \rangle$

$\alpha_5 = \langle (cf)d \rangle$

$\alpha_6 = \langle (abc)dcf \rangle$

FIGURE 2.22 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]



Sequence Support Count

- A **sequence database** is a set of tuples $\langle sid, s \rangle$
- A tuple $\langle sid, s \rangle$ is said to **contain** a sequence α , if α is a subsequence of s , i.e., $\alpha \subseteq s$
- The **support of a sequence** α is the number of tuples containing α



id	Sequence
10	$\langle a(abc)(ac)d(cf) \rangle$
20	$\langle (ad)c(bc)(ae) \rangle$
30	$\langle (ef)(ab)(df)cb \rangle$
40	$\langle eg(af)cbc \rangle$

$\alpha_1 = \langle a \rangle \quad \text{support}(\alpha_1) = 4$

$\alpha_2 = \langle ac \rangle \quad \text{support}(\alpha_2) = 4$

$\alpha_3 = \langle (ab)c \rangle \quad \text{support}(\alpha_3) = 2$

FIGURE 2.23 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]

2.2.2 Projection préfixée et bases projetées

Nous reprenons ici les définitions nécessaires pour introduire le principe de la *projection préfixée* [Pei et al., 2001].

Définition 16 (préfixe, projection, suffixe) Soient $\alpha = \langle \alpha_1 \dots \alpha_m \rangle$ et $\beta = \langle \beta_1 \dots \beta_n \rangle$ deux séquences (avec $m \leq n$).

- La séquence α est un préfixe de β ssi $\forall i \in [1..m-1], \alpha_i = \beta_i$. $\alpha_m \subseteq \beta_m$. Tous les items dans $(\beta_m - \alpha_m)$ sont alphabétiquement après ceux de α_m .
- La séquence $\beta = \langle \beta_1 \dots \beta_n \rangle$ est une projection de la séquence s par rapport à α , ssi (1) $\beta \preceq s$, (2) α est un préfixe de β et (3) il n'existe pas de sur-séquence propre β' de β telle que $\beta' \preceq s$ et α est un préfixe de β' .
- La séquence $\gamma = \langle \beta_{m+1} \dots \beta_n \rangle$ est appelée suffixe de s par rapport au préfixe α . Ainsi, nous avons $\beta = \text{concat}(\alpha, \gamma)$, avec "concat" est l'opérateur standard de concaténation.

Prefix

- Given two sequences $\alpha = \langle a_1 a_2 \dots a_n \rangle$ and $\beta = \langle b_1 b_2 \dots b_m \rangle$, $m \leq n$
- Sequence β is called a **prefix** of α if and only if:
 - $b_i = a_i$ for $i \leq m-1$;
 - $b_m \subseteq a_m$;
 - All the items in $(a_m - b_m)$ are alphabetically after those in b_m

$\alpha = \langle a(abc)(ac)d(cf) \rangle$ $\beta = \langle a(abc)a \rangle$	$\alpha = \cancel{\langle a(abc)(ac)d(cf) \rangle}$ $\beta = \cancel{\langle a(abc)c \rangle}$
---	---

FIGURE 2.24 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]

Définition 17 (Base de séquences projetées) Soit SDB une base de séquences. La base projetée (ou α -projection) de SDB par rapport à α , notée $SDB|_\alpha$, est l'ensemble de tous les suffixes des séquences de SDB ayant pour préfixe α .



Projection

- Given sequences α and β , such that β is a subsequence of α .
- A subsequence α' of sequence α is called a **projection** of α w.r.t. β prefix if and only if
 - α' has prefix β ;
 - There exist no proper super-sequence α'' of α' such that α'' is a subsequence of α and also has prefix β



$\alpha = < a(abc)(ac)d(cf) >$

$\beta = < (bc)a >$

$\alpha' = < (bc)(ac)d(cf) >$

FIGURE 2.25 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]

[Pei et al., 2001] ont proposé un algorithme efficace, appelé **PrefixSpan**, pour l'extraction de motifs séquentiels basé sur le principe de la *projection préfixée*.

Algorithme : *PrefixSpan* ($SDB, minsup$)

Données : SDB : base de séquences ; $minsup$: seuil de fréquence minimale

Résultat : SP : ensemble des motifs séquentiels

début

1 $SP \leftarrow \emptyset$;
 2 $PrefixSpanProj (\langle \rangle, SDB, minsup, SP)$;

fin

Procédure $PrefixSpanProj (\alpha, SDB|_{\alpha}, minsup, SP)$;

début

3 $FreqItems \leftarrow$ items fréquents dans $SDB|_{\alpha}$ par rapport à $minsup$ permettant d'étendre α pour former une nouvelle séquence fréquente ;
 4 **pour chaque** $a \in FreqItems$ **faire**
 5 $\alpha' \leftarrow Ext(\alpha, a)$;
 6 $SP \leftarrow SP \cup \{\alpha'\}$;
 7 $PrefixSpanProj (\alpha', SDB|_{\alpha'}, minsup, SP)$;

fin

FIGURE 2.26 – L'algorithme PrefixSpan sur des séquences d'items (section état de l'art [Kemmar, 2017, Kemmar et al., 2017, Kemmar et al., 2016a, Kemmar et al., 2016b])

Algorithm 1 (PrefixSpan)

Input: A sequence database S , and the minimum support threshold min_sup

Output: The complete set of sequential patterns

Method: Call PrefixSpan($\langle \rangle, 0, S$).

Subroutine PrefixSpan($\alpha, l, S|_{\alpha}$)

Parameters: α : a sequential pattern; l : the length of α ; $S|_{\alpha}$: the α -projected database, if $\alpha \neq \langle \rangle$; otherwise, the sequence database S .

Method:

1. Scan $S|_{\alpha}$ once, find the set of frequent items b such that
 - (a) b can be assembled to the last element of α to form a sequential pattern; or
 - (b) $\langle b \rangle$ can be appended to α to form a sequential pattern.
2. For each frequent item b , append it to α to form a sequential pattern α' , and output α' ;
3. For each α' , construct α' -projected database $S|_{\alpha'}$, and call PrefixSpan ($\alpha', l + 1, S|_{\alpha'}$).

FIGURE 2.27 – L'algorithme PrefixSpan sur des séquences d'itemsets [Pei et al., 2001]

A chaque étape, cet algorithme construit récursivement une nouvelle base de séquences de taille inférieure (base projetée). Ainsi, après avoir extrait tous les motifs fréquents de taille 1, c'est-à-dire contenant un seul item, une base projetée de SDB est calculée pour chaque item. Cette base projetée contient les suffixes des séquences ayant pour préfixe le motif fréquent de taille 1 contenant l'item. L'étape suivante calcule les motifs fréquents de taille 2 à partir des bases projetées obtenues à l'étape 1 ainsi que les bases projetées associées à ces nouveaux motifs. Le processus se répète, la taille des bases projetées se réduisant à chaque étape.

Exemple 2 Prenons l'exemple de la base de séquences de la Table 2.1. Lors de la première itération, la base de séquences fournie est celle de la Table 2.1. Supposons que $\text{minsup} = 2$. *PrefixSpan* commence par trouver les items fréquents. Pour cela, une passe sur la base de séquences SDB_1 va permettre de collecter le nombre de séquences supportant chaque item rencontré et donc d'évaluer le support des items de la base. Les items trouvés sont (sous la forme $\langle \text{item} \rangle : \text{support}$) : $\langle A \rangle : 3$, $\langle B \rangle : 4$ et $\langle C \rangle : 3$. Ensuite, les séquences de SDB_1 sont projetées en trois sous-ensembles disjoints, puisqu'il y a trois préfixes de taille 1 (i.e. les trois items fréquents). Ces sous-ensembles seront : (1) les motifs séquentiels ayant pour préfixe $\langle A \rangle$, (2) ceux ayant pour préfixe $\langle B \rangle$ et (3) ceux ayant pour préfixe $\langle C \rangle$. Par exemple, la base projetée de SDB_1 selon le préfixe $\langle A \rangle$, notée $SDB_1|_{\langle A \rangle}$, est composée de 3 suffixes : $\{(1, \langle BCBC \rangle), (2, \langle BC \rangle), (3, \langle B \rangle)\}$. Une passe sur $SDB_1|_{\langle A \rangle}$ permet alors d'obtenir les motifs séquentiels de longueur 2 ayant $\langle A \rangle$ pour préfixe commun : $\langle AB \rangle : 3$ et $\langle AC \rangle : 2$. Ainsi, de façon récursive, $SDB_1|_{\langle A \rangle}$ peut être partitionnée en deux sous-ensembles : (1) les motifs séquentiels ayant pour préfixe $\langle AB \rangle$ et (2) ceux ayant pour préfixe $\langle AC \rangle$. Ces motifs peuvent alors former de nouvelles bases projetées, et chacune de ces bases peut être utilisée en entrée de l'algorithme, toujours de manière récursive. Ce processus de α -projection des bases projetées se termine lorsque aucun super-motif séquentiel ne peut être obtenu.

La Proposition 3 établit une propriété pour calculer le support d'une séquence γ dans $SDB|_\alpha$ [Pei et al., 2001].

Proposition 3 (Calcul de support) Pour toute séquence γ dans SDB ayant pour préfixe α et pour suffixe β , avec $\gamma = \text{concat}(\alpha, \beta)$, $\text{sup}_{SDB}(\gamma) = \text{sup}_{SDB|_\alpha}(\beta)$.

Cette proposition garantit que seules les séquences dans SDB obtenues par extension du motif α seront considérées pour le calcul du support de la séquence γ . En outre, seuls les suffixes de $SDB|_\alpha$ doivent être considérés pour le calcul de ce support.

2.2.2.1 Illustrations de l'algorithme PrefixSpan sur des séquences d'items

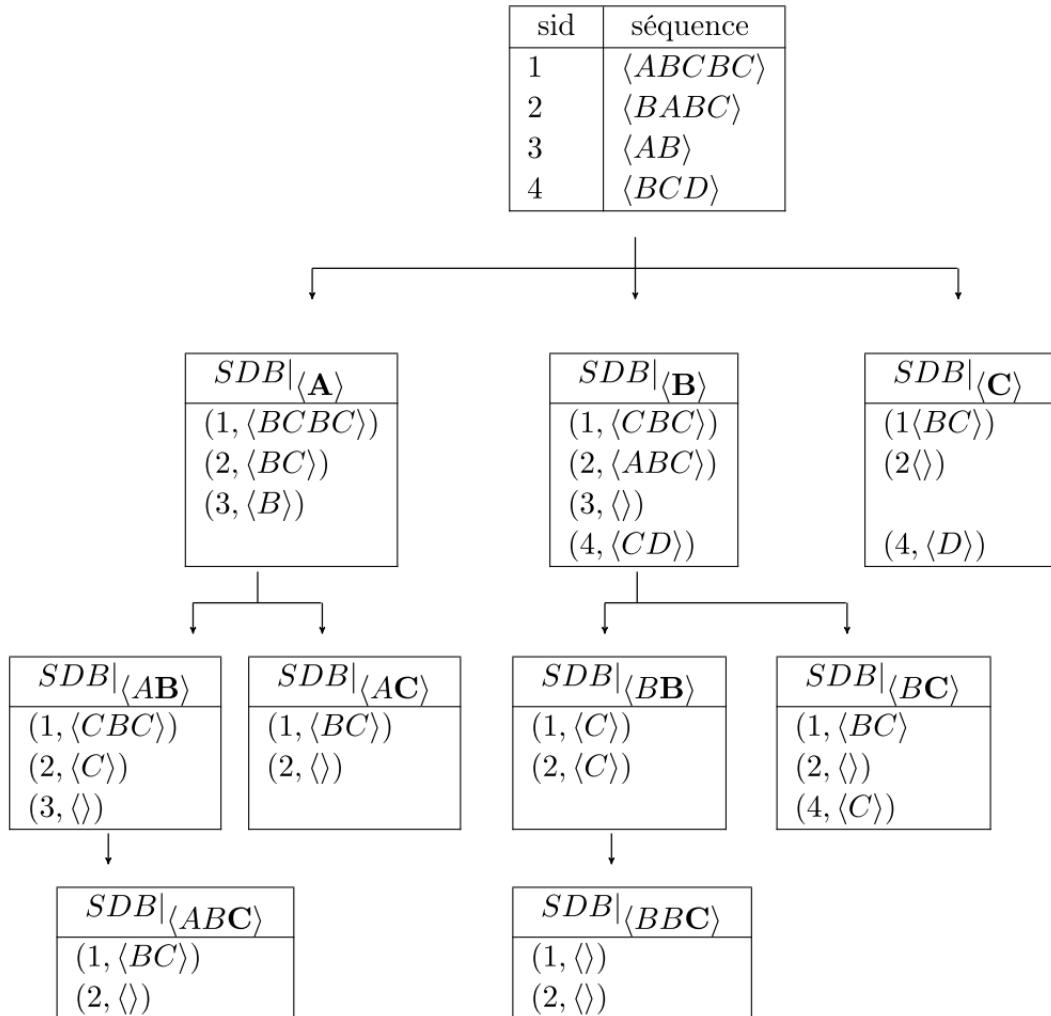


FIGURE 1.5 – Extraction des motifs séquentiels à partir de SDB_1 en utilisant *PrefixSpan* ($minsup = 2$)

FIGURE 2.28 – Illustration de l'algorithme PrefixSpan (section état de l'art [Kemmar, 2017, Kemmar et al., 2017, Kemmar et al., 2016a, Kemmar et al., 2016b])

2.2.2.2 Illustrations de l'algorithme PrefixSpan sur des séquences d'item-sets

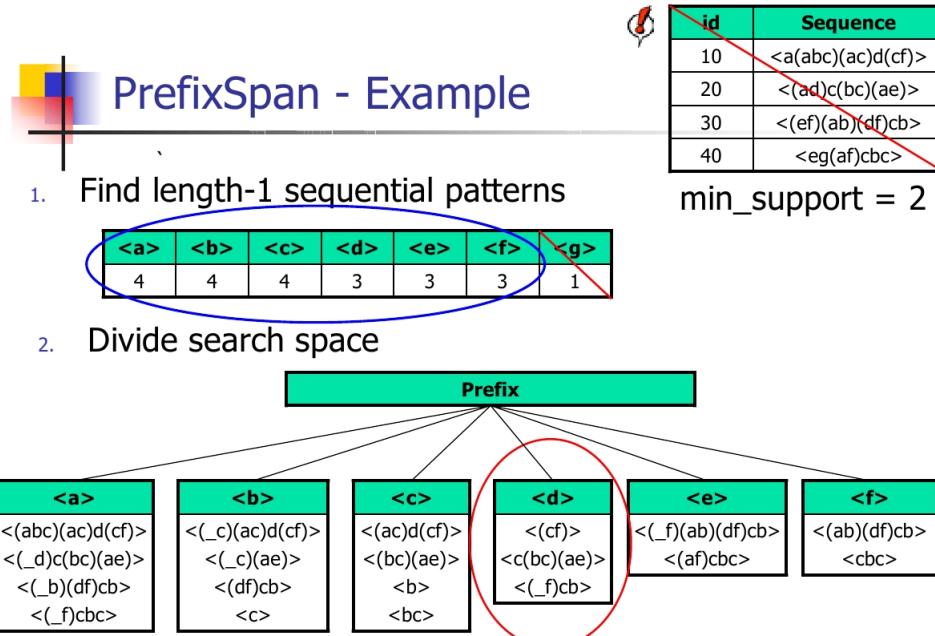


FIGURE 2.29 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]

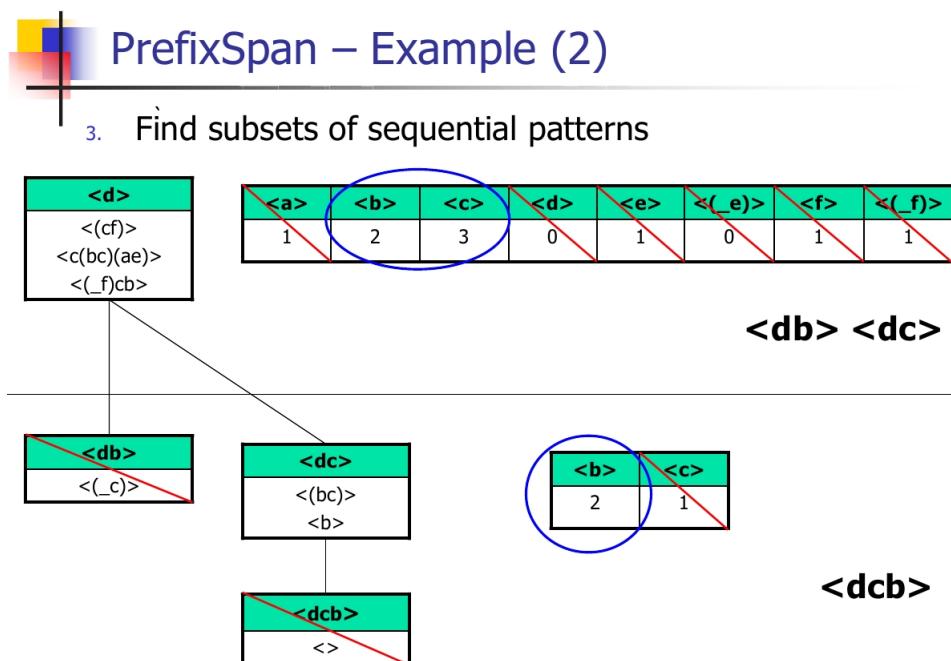


FIGURE 2.30 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]



Pseudo-Projection

- **Observation:** postfixes of a sequence often appear repeatedly in recursive projected databases
- **Method:** instead of constructing *physical* projection by collecting all the postfixes, we can use pointers referring to the sequences in the database as a pseudo-projection
- Every projection consists of two pieces of information: **pointer** to the sequence in database and **offset** to the postfix in the sequence



Pointer	Offset	Postfix
s1	2	<(abc)(ac)d(cf)>
s1	5	<(ac)d(cf)>
s1	6	<(_c)d(cf)>

FIGURE 2.31 – Illustration de l'algorithme PrefixSpan [PrefixSpan-Wojciech.pdf]

2.3 Analyse formel de concepts et treillis de Galois

Définition 18 (relation d'ordre) Soient E un ensemble et \leq une relation binaire sur E . \leq est une relation d'ordre sur E si elle est :

- réflexive ($\forall x \in E, x \leq x$),
- antisymétrique ($\forall x, y \in E, x \leq y \wedge y \leq x \Rightarrow x = y$),
- transitive ($\forall x, y, z, x \leq y \wedge y \leq z \Rightarrow x \leq z$),

Exemples d'ensemble ordonnés : (\mathbb{R}, \leq) , $(2^E, \subseteq)$, $(\mathbb{N}, \text{divise})$, (ensemble des mots, ordre lexicographique), ...

Définition 19 (Diagramme de Hasse) Un ensemble ordonné fini (E, \leq) peut être représenté par un graphe dont les sommets sont les éléments de E et les arcs sont tels que :

- Pour tout x et y de E , $x \leq y$ si et seulement il existe un chemin de x à y dans ce graphe ;
- Ce graphe ne contient pas de boucle (x, x) (on sait que $x \leq x$ car \leq est réflexive) ;
- Ce graphe ne contient pas d'arc de transitivité.

Ce graphe est appelé diagramme de Hasse associé à (E, \leq) .

Définition 20 (minorant, majorant, minimum, maximum) Soit (E, \leq) un ensemble ordonné. Soit $F \subseteq E$.

- Un minorant x de F est un élément de E tel que $\forall y \in F, x \leq y$.
- Un majorant x de F est un élément de E tel que $\forall y \in F, y \leq x$.
- Si x est un minorant de F et que $x \in F$, alors x est le minimum de F .
- On considère l'ensemble m des minorants de F . Si m a un et un seul maximum, ce maximum est appelé borne inférieure de F .
- On considère l'ensemble M des majorants de F . Si m a un et un seul minimum, ce minimum est appelé borne supérieure de F .

Définition 21 (treillis) Soit (E, \leq) un ensemble ordonné. (E, \leq) est un treillis si pour toute paire $\{x, y\}$ d'éléments de E , $\{x, y\}$ possède une borne inférieure (notée $x \wedge y$) et une borne supérieure (notée $x \vee y$).

Exemples de treillis :

- $(2^E, \subseteq)$ avec $\wedge = \cap$ et $\vee = \cup$;
- $(\mathbb{N}, \text{"divise"})$ avec $\wedge = \text{pgcd}$ et $\vee = \text{ppcm}$;

2.3.1 Contexte formel

Un contexte formel est un triplet $K = (O, A, I)$ où O est un ensemble d'objets ou d'individus, A est un ensemble d'attributs ou de propriétés et I est une relation binaire entre S et P vérifiant :

- $I \subseteq O \times A$;
- $(o, a) \in I$ avec $s \in S$ et $a \in A$.

La relation $(o, a) \in I$ signifie que l'objet o possède l'attribut ou la propriété a .

Considérons par exemple la base de données formelle suivante (qui va être utilisée tout au long du cours) :

\mathcal{R}	a	b	c	d	e
x_1	1	0	1	1	0
x_2	0	1	1	0	1
x_3	1	1	1	0	1
x_4	0	1	0	0	1
x_5	1	1	1	0	1
x_6	0	1	1	0	1

Où

- $\mathcal{O} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$,
- $\mathcal{P} = \{a, b, c, d, e\}$,
- $x \mathcal{R} p$ si et seulement si la ligne de x et la colonne de p se croisent sur un 1 (et pas sur un 0), par exemple : $x_1 \mathcal{R} a$, $x_1 \mathcal{R} c$ et $x_1 \mathcal{R} d$.

FIGURE 2.32 – Exemple d'un contexte formel [Slide Jean Lieber]

2.3.2 Connexion de Galois

Soit 2^O (resp. 2^A) l'ensemble des parties de O (resp. A). Considérons les applications suivantes :

Intension

$$f : 2^O \rightarrow 2^A \text{ définie par } f(X) = \{a \in A | \forall o' \in X, (o', a) \in I\}$$

f est l'application qui associe à tout ensemble d'objets de O l'ensemble de leurs attributs communs dans A .

Extension

$$g : 2^A \rightarrow 2^O \text{ définie par } g(Y) = \{o \in O | \forall a' \in Y, (o, a') \in I\}$$

g est l'application qui associe à tout ensemble d'attributs de A l'ensemble de leurs objets communs dans O .

Les fonctions f et g vérifient les propriétés suivantes :

- $\forall O_1, O_2 \in 2^O, O_1 \subseteq O_2 \Rightarrow f(O_2) \subseteq f(O_1)$
- $\forall A_1, A_2 \in 2^A, A_1 \subseteq A_2 \Rightarrow g(A_2) \subseteq g(A_1)$
- $\forall A_1 \in 2^A, A_1 \subseteq f(g(A_1))$
- $\forall O_1 \in 2^O, O_1 \subseteq g(f(O_1))$

Par définition, (f, g) forme une connexion de Galois entre $(2^O, \subseteq)$ et $(2^A, \subseteq)$.

2.3.3 Fermeture

Une fermeture sur un ensemble ordonné (E, \leq) est une application $r : E \rightarrow E$, qui, pour tout $x, y \in E$, vérifie les propriétés suivantes :

- $x \leq r(x)$ (r est extensive)
- si $x \leq y$ alors $r(x) \leq r(y)$ (r est monotone croissante)
- $r(x) = r(r(x))$ (r est idempotente)

Un élément x de E est fermé pour r si et seulement si $x = r(x)$.

Les applications $h = f \circ g$ et $h' = g \circ f$ sont des fermetures sur $(2^A, \leq)$ et $(2^O, \leq)$. Les fermés de h (resp. h') sont les éléments $A_1 \in 2^A$ (resp. $O_1 \in 2^O$) tels que $h(A_1) = A_1$ (resp. $h'(O_1) = O_1$).

2.3.4 Concept formel

Définition 22 (Concept formel) *Un concept formel ayant pour contexte formelle (O, A, I) est défini comme étant un couple $C = (D, B)$ avec*

- $D \subseteq O$,
- $B \subseteq A$,
- $f(D) = B$,
- $D = g(B)$.

Les ensemble D et B sont appelés respectivement extension et intension du concept formel C .

Définition 23 (Rectangle, Rectangle maximal) *Un rectangle d'un contexte formel (O, A, I) est un couple (X, m) où $X \subseteq O$, $m \subseteq A$, et tel que pour tout $x \in X$, x possède le motif ou les attributs m . Un rectangle (X, m) est maximal si ni X ni m ne peuvent être étendus. Autrement dit si on a un rectangle (X', m') tel que $X \subseteq X'$ et $m \subseteq m'$ alors $X' = X$ et $m' = m$.*

Dans la Figure 2.32, on a les rectangles $(\{x_2, x_3\}, \{b, c\})$, $(\{x_1, x_3, x_5\}, \{a\})$ et $(\{x_2, x_4\}, \{b, e\})$.

Définition 24 (Équivalence entre motifs, motifs fermés, motifs clés) *Soit θ la relation d'équivalence entre motifs définie par $m \theta m'$ si $g(m) = g(m')$. Soit $Cl(m)$ la classe d'équivalence pour θ contenant m : $Cl(m) = \{m' \in 2^A | m' \theta m\}$. Un motif est fermé si m est le maximum de $Cl(m)$. Un motif m est clef si m est un minimum de $Cl(m)$.*

On peut démontrer que si (X, m) est un rectangle maximal, alors $g(m) = X$, et m est donc fermé. On peut démontrer que le concept formel (A, B) est un rectangle maximal. m est fermé ssi $(g(m), m)$ est un rectangle maximal. Si m est fermé, on a $f(g(m)) = m$.

2.3.5 Treillis de Galois

Soient L_A et L_O les ensembles des fermés respectifs pour h et h' (voir leurs définitions dans la section 2.3.3). (L_A, \subseteq) est le treillis des fermés pour h et (L_O, \subseteq) est le treillis des fermés pour h' . (L_A, \subseteq) et (L_O, \subseteq) sont deux treillis isomorphes.

Considérons L_O et L_A et $L = L_O \times L_A$. On définit l'opérateur de subsomption de concept formel, qu'on notera \preceq , par :

$$C_1 = (A_1, B_1) \preceq C_2 = (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 (\Leftrightarrow B_2 \subseteq B_1).$$

On dit que C_1 est subsumé par C_2 ou encore C_1 est plus spécifique que C_2 , ou C_2 est plus général que C_1 .

(L, \preceq) est le produit des deux treillis isomorphes (L_O, \subseteq) et (L_A, \subseteq) appelés respectivement treillis des extensions et treillis des intensions. (L, \preceq) est par définition le treillis de Galois associé à la relation binaire I sur $O \times A$. L'ensemble de tous les concepts formels du contexte formel $K = (O, A, I)$ munis de l'ordre partiel \preceq est un treillis complet appelé treillis de concepts de K ou treillis de Galois.

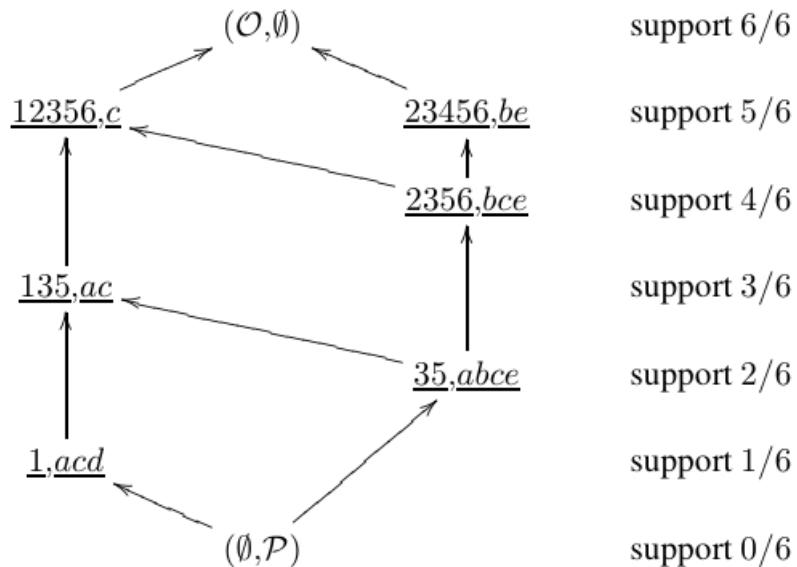


FIGURE 2.33 – Treillis de Galois [Slide Jean Lieber] / Convention : Par exemple, $\underline{135}, \underline{ac}$ représente le rectangle $(\{1, 3, 5\}, \{a, c\})$.

2.3.6 Algorithme de construction des treillis de Galois

La construction du treillis de Galois d'une relation binaire donnée peut être décomposée en trois parties [Claudio Carpineto and Giovanni Romano, ICML'1993] :

1. l'énumération des rectangles maximaux (les fermés),
2. la recherche de la relation d'ordre partiel entre ces rectangles
3. et la représentation du treillis (construction du diagramme de HASSE correspondant au treillis appelé aussi graphe de couverture).

2.4 Fouille de données déclarative

2.4.1 Extraction de motifs ensemblistes sous-contraintes

Soit \mathbf{r} un jeu de données où \mathcal{I} est l'ensemble de ses n items et \mathcal{T} l'ensemble de ses m transactions. \mathbf{r} peut être représenté par une matrice booléenne $\mathbf{d} = (d_{t,i})_{t \in \mathcal{T}, i \in \mathcal{I}}$, tel que $\forall t \in \mathcal{T}, \forall i \in \mathcal{I}, (d_{t,i} = 1) \Leftrightarrow (i \in t)$.

Variables

- Soit M le motif recherché. M est représenté par n variables booléennes $\{X_1, X_2, \dots, X_n\}$ tel que : $\forall i \in \mathcal{I}, (X_i = 1)$ ssi ($i \in M$).
- Le support du motif recherché M est représenté par m variables booléennes $\{T_1, T_2, \dots, T_m\}$ tel que : $\forall t \in \mathcal{T}, (T_t = 1)$ ssi ($M \subseteq t$).

Contraintes La relation entre le motif recherché M , son support et le jeu de données \mathbf{r} est établie par des contraintes réifiées imposant que, pour chaque transaction t , $(T_t = 1)$ ssi M est un sous ensemble de t [Guns et al., 2011] :

$$\forall t \in \mathcal{T}, (\mathbf{T}_t = \mathbf{1}) \Leftrightarrow \sum_{i \in \mathcal{I}} \mathbf{X}_i \times (\mathbf{1} - \mathbf{d}_{t,i}) = \mathbf{0} \quad (2.1)$$

L'encodage booléen permet d'exprimer de façon simple certaines mesures usuelles : $freq(M) = \sum_{t \in \mathcal{T}} T_t$ et $taille(M) = \sum_{i \in \mathcal{I}} X_i$. La contrainte de fréquence minimale $freq(M) \geq min_{fr}$ (ou min_{fr} est un seuil) est encodée par la contrainte $\sum_{t \in \mathcal{T}} T_t \geq min_{fr}$. De la même manière, la contrainte de taille minimale $taille(M) \geq \alpha$ (ou α est un seuil) est encodée par la contrainte $\sum_{i \in \mathcal{I}} X_i \geq \alpha$.

Finalement, la contrainte de fermeture $fermé_m(M)$ (Avec $m = freq$) est encodée par l'équation (2.2).

$$\forall i \in \mathcal{I}, (\mathbf{X}_i = 1) \Leftrightarrow \sum_{t \in \mathcal{T}} \mathbf{T}_t \times (\mathbf{1} - \mathbf{d}_{t,i}) = \mathbf{0}. \quad (2.2)$$

Interprétation de la formule de fermeture : **L'item i est dans le motif SSI l'item i est dans les transactions actives.**

\Rightarrow Si un item est dans un motif, il doit être dans les transactions actives. On force à sélectionner toutes les transactions couvertes par le motif.

\Leftarrow : Tous les items qui sont dans les transactions actives, ils doivent aussi être dans le motif. On force à garder tous les items couverts par les transactions actives, dans le motif.

Lazaar et al. [Lazaar et al., 2016] ont proposé une seule contrainte, la contrainte CLOSED PATTERN, qui permet de prendre en compte toutes les contraintes de fréquence et de clôture. Nous l'introduisons brièvement ci-dessous.

Soit P le motif inconnu. Le motif P est encodé avec les variables booléennes P_1, \dots, P_n . Nous dénotons σ linstanciation partielle des variables P_1, \dots, P_n qui ont un domaine avec une seule valeur. Nous dénotons les trois ensembles suivants :

- items présents : $\sigma^+ = \{j \in 1..n \mid P_j = 1\}$,
- items absents : $\sigma^- = \{j \in 1..n \mid P_j = 0\}$,
- items autres : $\sigma^* = \{1..n\} \setminus (\sigma^+ \cup \sigma^-)$.

σ^* est l'ensemble des items libres (variables non instanciées). Si $\sigma^* = \emptyset$ alors σ est une affectation complète.

La contrainte globale CLOSED PATTERN garantit la contrainte de fréquence et de cloture.

Définition 25 (contrainte globale CLOSED PATTERN) Soient P_1, \dots, P_n des variables binaires. Soit SDB la base transactionnelle et θ le support minimal. Etant donné l'affectation complète σ sur P_1, \dots, P_n , $CLOSEDPATTERN_{SDB,\theta}(\sigma)$ est valide si et seulement si $\text{freq}_{SDB}(\sigma^+) \geq \theta$ et σ^+ est clos.

2.4.2 Extraction de motifs séquentiels sous-contraintes

Dans la pratique, le nombre de motifs séquentiels fréquents peut être important. Une façon de réduire leur nombre (sans perte d'information) est d'utiliser des contraintes. Les contraintes permettent à l'utilisateur de définir plus précisément les motifs qu'il considère pertinents [Pei et al., 2002]. Cette section présente les contraintes les plus utilisées en fouille de séquences [Pei et al., 2002]. D'autres contraintes [Zaki, 2000], qui ne sont pas abordées dans ce travail, peuvent être aussi intéressantes pour l'utilisateur telles que la contrainte de gap, la contrainte d'inclusion ou encore la contrainte d'agrégat.

1. **Contrainte de fréquence.** Cette contrainte, que nous avons déjà introduite, permet de restreindre l'ensemble des motifs extraits aux motifs fréquents selon un seuil de fréquence minimal $minsup$. Pour modéliser formellement cette contrainte, nous adoptons le formalisme de Kemmar et al. [Kemmar et al., 2016a]. Soit P un motif inconnu de longueur ℓ . Le symbole \square représente l'absence d'item et indique la fin de la séquence. Tout motif inconnu P est encodé par une séquence de ℓ variables $\langle P_1, P_2, \dots, P_\ell \rangle$ t.q. $\forall i \in [1 \dots \ell], D(P_i) = \mathbb{I} \cup \{\square\}$, avec les deux règles suivantes sur les domaines :

- Pour ignorer la séquence vide, le premier item de P doit être non vide : $\square \notin D_1$.
- Pour autoriser des motifs de taille inférieure à ℓ : $\forall i \in [1..(\ell-1)], (P_i = \square) \rightarrow (P_{i+1} = \square)$.

La contrainte globale PREFIX-PROJECTION encode à la fois la relation de sous-séquence et la contrainte de fréquence minimale.

Définition 26 (Contrainte PREFIX-PROJECTION) Soient $P = \langle P_1, P_2, \dots, P_\ell \rangle$ la séquence de variables représentant un motif inconnu de longueur ℓ et $minsup$ le seuil de support minimal. L'instanciation $\langle d_1, \dots, d_\ell \rangle \in D(P_1) \times \dots \times D(P_\ell)$ est une solution pour la contrainte PREFIX-PROJECTION($P, SDB, minsup$) si et seulement si $\text{sup}_{SDB}(\langle d_1, \dots, d_\ell \rangle) \geq minsup$.

Proposition 4 (Test de cohérence de PREFIX-PROJECTION) La contrainte PREFIX-PROJECTION($P, SDB, minsup$) admet une solutionssi il existe une instanciation $\sigma = \langle d_1, \dots, d_\ell \rangle$ des variables de P t.q. $SDB|_\sigma$ contient au moins $minsup$ suffixes des projections des séquences de SDB par rapport à σ , i.e. $|SDB|_\sigma| \geq minsup$.

Preuve C'est une conséquence directe de la proposition 3, car $\text{sup}_{SDB}(\sigma) = \text{sup}_{SDB|\sigma}(\langle \rangle) = |SDB|_\sigma|$. Ainsi, les items fréquents dans les éléments de $SDB|_\sigma$ représentent des supports de σ pour la contrainte $\text{PREFIX-PROJECTION}(P, SDB, \text{minsup})$, ce qui assure que $|SDB|_\sigma| \geq \text{minsup}$.

□

La proposition suivante permet de caractériser les valeurs du domaine de la variable non instanciée P_{i+1} (appelée aussi variable future) qui étendent l'instanciation courante des variables $\langle P_1, \dots, P_i \rangle$ en une instanciation cohérente (i.e. satisfaisant la contrainte globale $\text{PREFIX-PROJECTION}(P, SDB, \text{minsup})$).

Proposition 5 (Valeurs consistantes) Soit³ $\sigma = \langle d_1, \dots, d_i \rangle$ une instantiation des variables $\langle P_1, \dots, P_i \rangle$ et P_{i+1} une variable future. Une valeur $d \in D(P_{i+1})$ participe à une solution de $\text{PREFIX-PROJECTION}(P, SDB, \text{minsup})$ si et seulement si d est un item fréquent dans $SDB|_\sigma$:

$$|\{(sid, \gamma) | (sid, \gamma) \in SDB|_\sigma \wedge \langle d_{i+1} \rangle \preceq \gamma\}| \geq \text{minsup}$$

Preuve Supposons que la valeur $d \in D(P_{i+1})$ apparaisse dans $SDB|_\sigma$ plus de minsup fois. De la proposition 3, nous avons $\text{sup}_{SDB}(\text{concat}(\sigma, \langle d \rangle)) = \text{sup}_{SDB|\sigma}(\langle d \rangle)$. Par conséquent, l'instanciation $\text{concat}(\sigma, \langle d \rangle)$ satisfait la contrainte, et la valeur $d \in D(P_{i+1})$ est donc viable (i.e. participe à au moins une solution).

□

2. Contrainte de longueur minimale.

Cette contrainte, notée

$$\text{minSize}(P, \ell_{\min}),$$

spécifie la taille minimale (en nombre d'items) des motifs extraits (i.e. les motifs extraits doivent contenir au moins ℓ_{\min} items).

Exemple 3 Reprenons l'exemple 1 en imposant la contrainte $\text{minSize}(P, 3) \wedge \text{sup}_{SDB_1}(P) \geq 2$. Seuls deux motifs sont extraits $P_1 = \langle ABC \rangle$ et $P_2 = \langle BBC \rangle$.

3. Contrainte de longueur maximale.

Cette contrainte, notée

$$\text{maxSize}(P, \ell_{\max}),$$

spécifie la taille maximale (en nombre d'items) des motifs extraits (i.e. les motifs extraits doivent contenir au plus ℓ_{\max} items).

Exemple 4 Par exemple, en imposant la contrainte

$$\text{maxSize}(P, 2) \wedge \text{sup}_{SDB_1}(P) \geq 3,$$

seuls les motifs suivants sont extraits : $\langle A \rangle$, $\langle B \rangle$, $\langle C \rangle$, $\langle AB \rangle$ et $\langle BC \rangle$.

4. Contrainte d'item.

Cette contrainte spécifie le sous-ensemble d'items qui

doivent apparaître dans les motifs extraits :

$$\text{item}(P, t) \equiv (\exists i \in 1..|P|, p_i = t).$$

3. Nous noterons indifféremment σ soit par $\langle d_1, \dots, d_i \rangle$ ou par $\langle \sigma(P_1), \dots, \sigma(P_i) \rangle$.

Exemple 5 Considérons les trois contraintes suivantes : $\text{sup}_{SDB_1}(P) \geq 3 \wedge \text{maxSize}(P, 2) \wedge \text{item}(P, C)$, définies sur les séquences de la Table 2.1. Seuls deux motifs sont extraits $P_3 = \langle C \rangle$ et $P_4 = \langle BC \rangle$.

5. **Contrainte d'expression régulière.** La contrainte d'expression régulière, notée

$$\text{reg}(P, \text{exp})$$

, assure que le motif P doit être reconnu par un automate d'états finis associé à l'expression régulière exp [Garofalakis et al., 2002].

Exemple 6 Considérons la contrainte d'expression régulière $\text{reg}(P, \text{exp})$, où $\text{exp} = B\{BC|D\}$. Le motif $P_5 = \langle BBC \rangle$ est extrait de SDB_1 .

2.5 Fouille de graphes

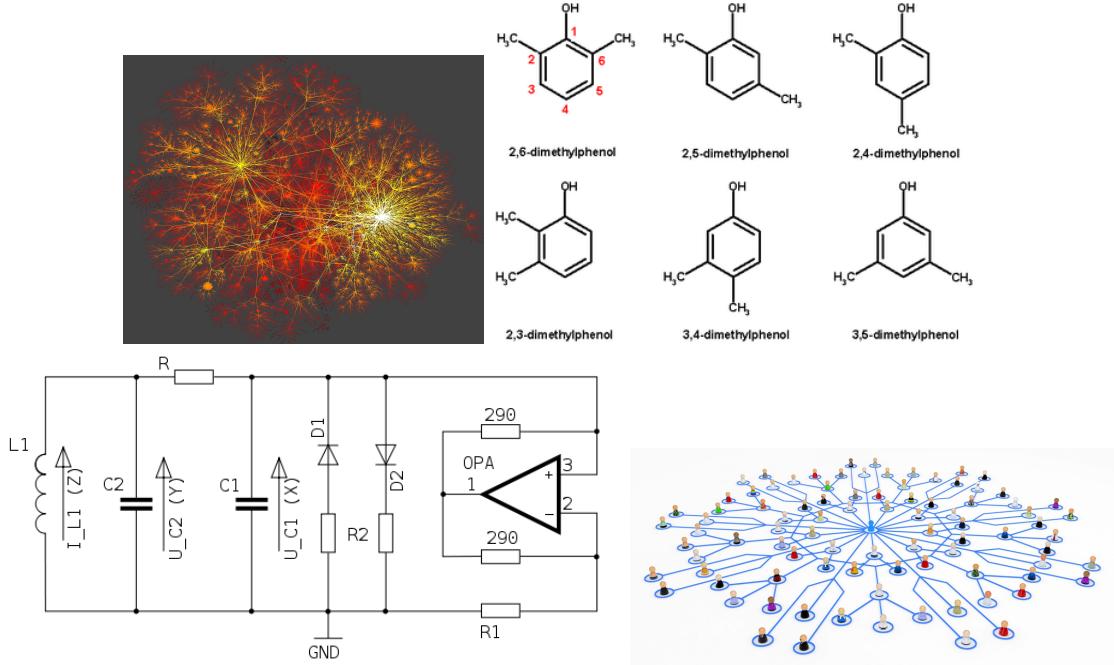


FIGURE 2.34 – Graphs

4

2.5.1 Basic definitions and concepts

Labeled Graph. A *labelled graph* can be represented by a 4-tuple, $G = (V, E, L, l)$, where V is a finite set of vertices, $E \subset V \times V$ is a set of edges, L is a set of labels, $l : V \cup E \rightarrow L$ is a function assigning a label to every element of $V \cup E$, and $\deg : V \rightarrow \mathbb{N}^+$ is a function that calculates the degree of a vertex $v \in V$.

4. Des extraits sont pris de [?]

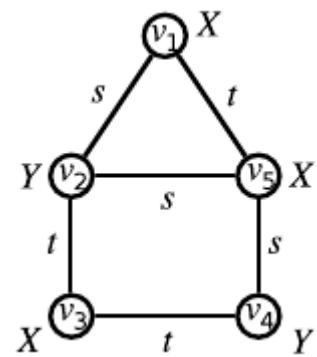


FIGURE 2.35 – Labeled Graph

Cyclic Graph. A graph is *cyclic* if it has many cycles. As the number of cycles increases, the number of edges increases, and then the density of the graph increases too.

Dense Graph. A *dense graph* is a graph in which the number of edges is close to the maximal number of edges. The graph density D is defined as : $D = 2|E|/(|V|(|V| - 1))$. $(|V|(|V| - 1)/2)$ is the number of edges in a complete graph. Clearly, the given formula of D computes the proximity of the number of edges to the maximum number of edges.

Isomorphism and Subgraph Isomorphism. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs.

- G_1 and G_2 are *isomorphic* if there is a bijection function $f : V_1 \rightarrow V_2$ satisfying : (i) $\forall u \in V_1, l_1(u) = l_2(f(u))$, (ii) $\forall \{u, v\} \in E(1), \{f(u), f(v)\} \in E(2)$, and (iii) $\forall \{u, v\} \in E(1), l_1(\{u, v\}) = l_2(\{f(u), f(v)\})$.
- G_2 is a *subgraph* of G_1 , iff $V_2 \subset V_1$, and $E_2 \subset E_1 \wedge (\forall \{v_1, v_2\} \in E_2 \Rightarrow v_1 \in V_2 \text{ and } v_2 \in V_2)$.
- G_1 is *subgraph isomorphic* to G_2 , denoted $G_1 \subseteq G_2$, if G_1 is isomorphic to a subgraph of G_2 .

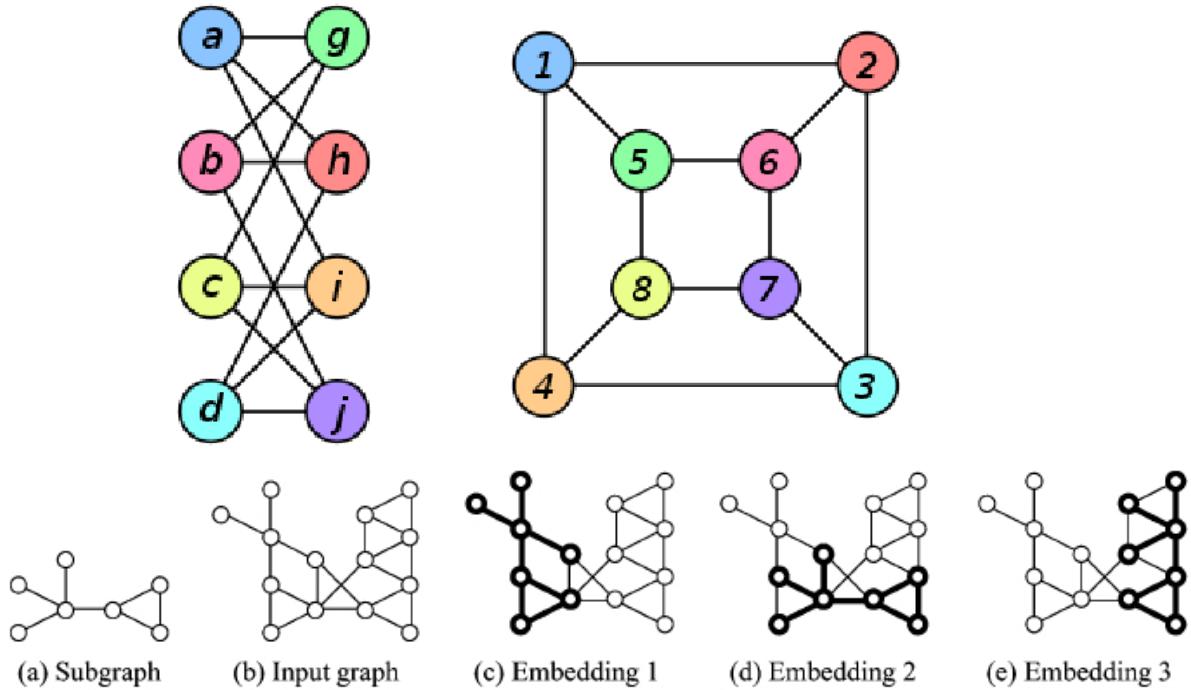
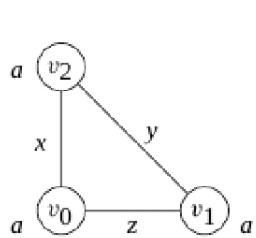


FIGURE 2.36 – Labeled Graph and Graph Isomorphism and subgraph isomorphism

Deciding whether a graph is subgraph isomorphic to another graph is NP-complete ([Read and Corneil, 1972]).

Définition 27 (Canonical encoding) An encoding function ϕ is a function that assigns a code to a given graph. This code represents some structural properties of the graph such as a sequence of bits, a string or a sequence of numbers. The encoding function ϕ is canonical when for any two graphs G_1, G_2 :

G_1 and G_2 are isomorphic iff $\phi(G_1) = \phi(G_2)$.



	v0	v1	v2
v0	a		x
v1	a	z	y
v2	a	x	y

code = **aaa_zxy**

	v1	v0	v2
v1	a		y
v0	a	z	x
v2	a	y	x

code = **aa_zyx**

- Chaque permutation de sommets \Rightarrow nouveau code
- code canonique = minimum (max) de tous ces codes
- Complexité = $O(|V|!)$

FIGURE 2.37 – Canonical encoding

Définition 28 (Frequent subgraph discovery) Given a transaction database \mathcal{D} which contains a family of graphs. The frequency of a graph G in \mathcal{D} is defined by $\text{freq}(G, \mathcal{D}) = |\{G_D \in \mathcal{D} \mid G \subseteq G_D\}|$. The support of a graph is defined by

$$\text{support}(G, \mathcal{D}) = \frac{\text{freq}(G, \mathcal{D})}{|\mathcal{D}|}$$

Let s_{min} be some predefined minimum support. The frequent subgraph discovery problem consists in finding connected undirected graphs G' that are subgraphs of at least $(s_{min} \times |\mathcal{D}|)$ graphs of \mathcal{D} :

$$\mathcal{F} = \{G' \mid \text{support}(G', \mathcal{D}) \geq s_{min}\}.$$

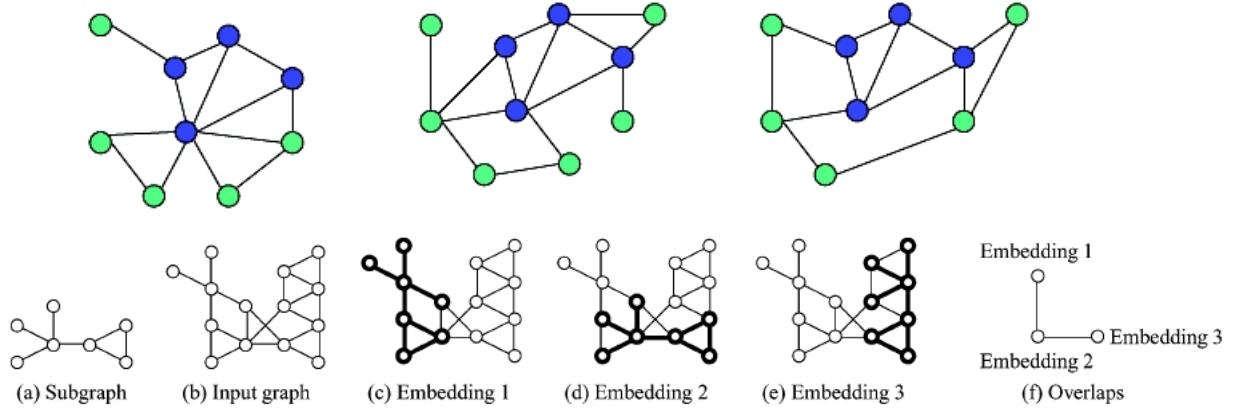


FIGURE 2.38 – Labeled Graph and Graph Isomorphism and subgraph isomorphism

2.5.2 Frequent subgraph mining

Generally, we can distinguish between the frequent subgraph algorithms according to the way the three following problems are handled :

1. **Candidates generation problem** : The candidates are initialized with frequent edges (1-candidates). The k -candidates (i.e., having k edges) are generated, by adding one edge to each $(k - 1)$ -candidate. This process can be done with a breadth-first strategy as well as a depth-first strategy.
2. **Subgraph encoding problem** : A canonical encoding is assigned to each generated graph. Verifying that the candidate is new, consists in checking that its encoding does not belong to the encodings of the already generated candidates.
3. **Frequency computation problem** : Once a new candidate is generated, we have to compute its frequency. It can be achieved by finding all the transactions of the database that contain this new candidate.

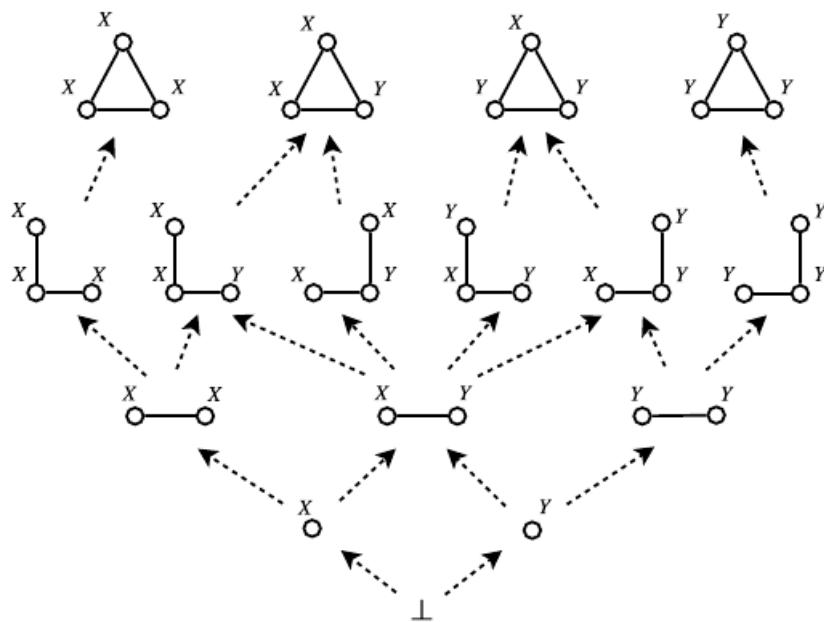


FIGURE 2.39 – Labeled Graph and Graph Isomorphism and subgraph isomorphism

2.6 Partitionnement avec contraintes (constrained clustering)

- 2.6.1 Clustering conceptuel
- 2.6.2 Co-clustering
- 2.6.3 Soft clustering
- 2.6.4 ...

Chapitre 3

Strategies mining and reinforcement learning

- Complete course
 - http://cs330.stanford.edu/fall2021/material/CS330_RLTutorial.pdf
 - <https://awjuliani.medium.com/super-simple-reinforcement-learning-tutorial>
 - <https://www.davidsilver.uk/teaching/>
 - <https://www.youtube.com/watch?v=2pWv7G0vuf0>
- Top illustrated course
 - <https://huggingface.co/learn/deep-rl-course/unit2/introduction>
- Ressources
 - <https://www.kaggle.com/competitions/connectx/discussion/124421>
 - <https://github.com/tsmatz/reinforcement-learning-tutorials>
 - <https://neptune.ai/blog/best-reinforcement-learning-tutorials-examples-practices>
- Youtube
 - <https://www.youtube.com/watch?v=FgzM3zpZ55o&list=PLoROMvodv4r0S0PzutgyCT>
- Taxi et RL https://gymnasium.farama.org/environments/toy_text/taxi/
 - <https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python/>
 - <https://www.kaggle.com/code/karthikcs1/reinforcement-learning-taxi-v3-op>
 - <https://www.kaggle.com/code/charel/learn-by-example-reinforcement-learning-taxi-v3>
- Echecs <https://www.kaggle.com/code/arjanso/reinforcement-learning-chess-1-point-notebook>
- Others
 - <https://www.kaggle.com/code/phunghieu/connectx-with-q-learning/notebook>
 - <https://github.com/dennybritz/reinforcement-learning>
 - <https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial>
 - <https://www.javatpoint.com/reinforcement-learning>
 - <https://www.toptal.com/deep-learning/pytorch-reinforcement-learning-tutorial>
 - <https://blog.paperspace.com/getting-started-with-reinforcement-learning/>
 - <https://nbviewer.org/github/DavidSanwald/ai-notebook/blob/master/index.ipynb>
 - <https://github.com/DavidSanwald/ai-notebook>

https://calvinfeng.gitbook.io/machine-learning-notebook/unsupervised-learning/reinforcement-learning/reinforcement_learning

Chapitre 4

Apprentissage par renforcement

Références :

Reinforcement Learning : State-of-the-Art Traduit de [Wiering and van Otterlo, 2014]

4.1 Introduction

4.2 Processus de décision Markovien

Un Processus de Décision Markovien MDP (Markov Decision Process) consiste en un quadruplet : états, actions, fonction de transition entre états, et finalement une fonction récompense (reward). Nous détaillons chacun ces 4 éléments :

Etats L'ensemble des états environnementaux S est défini comme un ensemble fini d'états

$$S = \{s^1, s^2, \dots, s^N\}$$

où la taille de l'espace des états est N . Un état est une caractérisation unique de tout ce que l'on pourrait qualifier un état de l'environnement. Par exemple, dans le jeu des échecs, un état peut être le contenu de chacune des 8x8 cases, comme ça pourrait être une image de l'échiquier. Il est ainsi nécessaire de distinguer des états possibles (légaux) des états irréalisables (illégaux). Nous prenons l'hypothèse que S contient uniquement des états possibles identifiés par des clés symboliques.

Actions L'ensemble des actions A est défini par l'ensemble

$$A = \{a^1, a^2, \dots, a^K\}$$

où K est la taille de l'espace des actions. Les actions permettent de contrôler les états. Un ensemble restreint d'actions peuvent être effectuées à partir d'un état $s \in S$, dénoté $A(s)$ où $A(s) \subseteq A$. En supposant que toutes les actions sont possibles à partir d'un état, nous adoptons une fonction précondition $pre : S \times A \rightarrow \{\text{true}, \text{false}\}$ précisant si la transition est permis ou non.

Fonction de transition En appliquant une action $a \in A$ à partir d'un état $s \in S$, le système effectue une transition vers un nouvel état $s' \in S$, en

se basant sur une distribution de probabilité par rapport à l'ensemble des transitions possibles. La fonction de transition T est définie

$$T : S \times A \times S \rightarrow [0, 1]$$

i.e., la probabilité pour atteindre l'état s' après exécution d'une action a au niveau d'un état s , dénotée $0 \leq T(s, a, s') \leq 1$. Nous avons nécessairement, $\forall s \in S, a \in A, \sum_{s' \in S} T(s, a, s') = 1$. Ainsi, au lieu d'utiliser une fonction précondition, nous pouvons simplement de supposer $T(s, a, s') = 0$ pour tous les états $s' \in S$ si a n'est pas possible à partir de s .

Pour exprimer un ordre dans lequel les actions sont possibles, nous supposons l'existence d'une horloge globale, $t = 1, 2, \dots$. On utilisera la notation s_t dénotant l'état à l'instant t , et s_{t+1} à l'instance $t+1$, permettant de comparer les différents états (et donc les actions aussi) au cours du temps.

On dit que le système est Markovien si toute action ne dépend pas des états précédent, mais uniquement de l'état courant :

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1}) \quad (4.1)$$

L'idée de la dynamique Markovienne est que l'état courant dispose de suffisamment d'information pour prendre la meilleure décision. Des modèles plus généraux peuvent être caractérisés comme étant k -Markov, i.e. le dernier état dépend des k états précédents, bien que un processus k -Markov peut toujours être transformé en un processus 1-Markov par regroupement d'états. Parmi ces processus généraux, on peut noter les MDPs partiellement observables POMDPs (Partially Observable MDP).

Fonction de récompense (Reward) La fonction Reward spécifie la récompense au niveau d'un état donné ou exécutant une action donnée. La fonction Reward est identifiée

$$R : S \rightarrow \mathbb{R}$$

et spécifie la récompense obtenue dans les états. Deux autres notations sont aussi adoptées

$$R : S \times A \rightarrow \mathbb{R}$$

ou encore

$$R : S \times A \times S \rightarrow \mathbb{R}.$$

La première caractérise la récompense d'un état s , la deuxième l'action a de transition à partir d'un état s , et la troisième l'action a de transition d'un état s vers un autre état s' . Nous utilisons le plus souvent $R(s, a, s')$.

La fonction de transition est une partie importante d'un MDP, car elle spécifie implicitement la fonction objectif de la problématique de décision. Par exemple, dans un jeu (e.g., échec, tic-tac-toe, ...), on peut mettre un Reward positif sur les états gagnants et négatif sur les états perdants.

A partir de ce quadruplet (S, A, T, R) , nous pouvons définir ce qu'est un processus de décision Markovien :

Définition 29 (Processus de décision Markovien MDP) *Un processus de décision Markovien MDP est un tuple $\langle S, A, T, R \rangle$, avec S l'ensemble des états, A l'ensemble des actions, T la fonction de transition de signature $T : S \times A \times S \rightarrow [0, 1]$, R la fonction Reward de signature $R : S \times A \times S \rightarrow \mathbb{R}$.*

Via cette définition, nous pouvons distinguer plusieurs types de systèmes. Par exemple, un système de tâches épisodiques utilisant la notion d'épisode qui consiste à ramener le système d'un état initial donné vers un autre état final. Une distribution d'état initial est donnée

$$I : S \rightarrow [0, 1]$$

permettant d'avoir la probabilité de démarrer à partir d'un état. Dans ce contexte, nous pouvons distinguer un sous-ensemble spécifique $G \subseteq S$, dénotant les états but (goal). On peut aussi distinguer des tâches finies ou à horizon fixe, correspondant à des tâches avec un ensemble fini d'étapes, contrairement aux tâches à horizon infini. Finalement, les tâches infinies qui ne terminent jamais, dites aussi tâches continues.

Pour les tâches épisodiques, ayant des états buts, ces derniers sont modélisés via ce que l'on appelle des états absorbants ou état terminaux.

4.2.1 Stratégies et motifs stratégiques

Etant donné un MDP $\langle S, A, T, R \rangle$, une stratégie est une fonction calculable fournit pour chaque état $s \in S$, une action $a \in A$. Formellement une stratégie π est une fonction

$$\pi : S \rightarrow A.$$

On peut aussi définir une stratégie stochastique

$$\pi : S \times A \rightarrow [0, 1]$$

tel que $\sum_{a \in A} \pi(s, a) = 1$.

L'application d'une stratégie à un MDP est faite comme suit. En premier, un état initial s_0 à partir des états initiaux possibles I est généré. Ainsi, la stratégie π suggère l'action $a_0 = \pi(s_0)$, et cette action est exécutée. En exploitant la fonction de transition T et la fonction Reward R , la transition est exécutée vers l'état s_1 avec la probabilité $T(s_0, a_0, s_1)$ et une récompense $r_0 = R(s_0, a_0, s_1)$. Ce processus continue produisant la séquence

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \dots$$

Si la tâche est épisodique, le processus finit dans l'état s_{but} , et redémarre à partir d'un autre état de I . Si la tâche est continue, le processus ne s'arrêtera pas.

La stratégie est une partie intégrante de l'agent et son objectif est de contrôler l'environnement modélisé par un MDP.

4.2.2 Critère d'optimalité

Nous venons de définir ce qu'est un environnement modélisé sous forme d'un MDP. Nous allons définir ce qu'est une stratégie optimale, ce qu'est un modèle d'optimalité. Nous devons définir quoi optimiser ? Puis comment y arriver ? La réponse à la première question est relative à l'agrégation des récompenses, la deuxième est relative au performance et sera discuté dans la section suivante.

En somme, l'objectif est de rassembler le plus de récompenses possibles. Mais localement à un état, la décision consiste simplement à optimiser

$$E[r_t].$$

Il existe trois modèles d'optimalité :

Horizon fini

$$E\left[\sum_{t=0..h} r_t\right]$$

Horizon discontinu infini

$$E\left[\sum_{t=0..\infty} \gamma^t r_t\right]$$

Récompense moyenne

$$\lim_{h \rightarrow \infty} E\left[\frac{1}{h} \sum_{t=0..h} r_t\right]$$

4.2.3 Fonction d'évaluation et équations de Bellman

La valeur d'un état s suivant une stratégie π , dénotée $V^\pi(s)$ est le retour attendu en démarrant de l'état s en empruntant la stratégie π . Nous supposons un horizon infini et discontinu, menant vers la formulation :

$$V^\pi(s) = E_\pi\left\{\sum_{k=0..\infty} \gamma^k r_{t+k} | s_t = s\right\} \quad (4.2)$$

On peut aussi établir une fonction de retour tenant compte à la fois l'état et l'action :

$$Q^\pi(s, a) = E_\pi\left\{\sum_{k=0..\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\right\} \quad (4.3)$$

On peut reformuler (4.2) :

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{\sum_{k=0..\infty} \gamma^k r_{t+k} | s_t = s\right\} \\ &= E_\pi\{r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = t\} \\ &= E_\pi\{r_t + \gamma V^\pi(s_{t+1}) | s_t = t\} \\ &= \sum_{s'} T(s, \pi(s), s')(R(s, a, s') + \gamma V^\pi(s')) \end{aligned} \quad (4.4)$$

L'objectif pour tout MDP est de trouver la meilleure stratégie, celle recevant la meilleure récompense, i.e. maximisant la valeur de la formule (4.2) pour tous les états $s \in S$. Une stratégie optimale, dénotée π^* , est telle que $V^{\pi^*}(s) \geq V^\pi(s)$ pour

tout $s \in S$ et toute stratégie π . On peut démontrer que la solution optimale $V^* = V^\pi$ satisfait l'équation :

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \quad (4.5)$$

Cette dernière équation est dite l'équation d'optimalité de Bellman. Elle stipule que dans une stratégie optimale, la valeur d'un état est égale à celle de la meilleure action. Pour faire valoir la meilleure action, on peut écrire :

$$V^*(s) = \operatorname{argmax}_a \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \quad (4.6)$$

On appelle cette stratégie, la stratégie gloutonne, dénotée $\pi_{glouton}(V)$. On peut aussi la réécrire en utilisant la formulation sur les actions :

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma \max_{a'} Q^*(s', a')) \quad (4.7)$$

Les fonctions Q sont utiles, car elles évitent la somme pondérée utilisant la fonction de transition. Et donc :

$$\begin{aligned} Q^*(s, a) &= \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \\ V^*(s) &= \max_a Q^*(s, a) \end{aligned} \quad (4.8)$$

4.3 Résoudre un MDP

Résoudre un MDP consiste à rechercher π^* . Des solutions existent dans la littérature, comprenant celle basées sur un modèle préalablement défini, et celles sans modèle prédéfini. Celles basées sur un modèle bien défini, dites par programmation dynamique, ont été largement abordées dans la littérature, notamment en recherche opérationnelle. Nous nous intéressons particulièrement ici aux problématiques RL où le modèle n'est pas totalement défini à priori. En RL, le modèle est souvent itérativement défini en interagissant avec l'environnement, notamment en simulant la stratégie générant des échantillons des états-transitions et des récompenses.

Un problème MDP peut aussi être vu comme un problème de planification. Les fonctions de transition peuvent être vues comme des préconditions. Les fonctions buts en planification $G : S \rightarrow \{\text{true}, \text{false}\}$ sont les états finaux. De point de vue planification, il est question de trouver la suite d'actions qui mènent vers un état final.

4.3.1 Solutions basées sur un modèle prédéfini : programmation dynamique (DP)

...

4.3.2 Solutions basées sur un modèle d'environnement : apprentissage par renforcement (RL)

Dans la section précédente, il était question de trouver la solution à partir d'une modèle précis de l'environnement. L'apprentissage par renforcement RL est concerné par la recherche de la meilleure stratégie quand ce modèle n'existe pas. RL ajoute au MDP le besoin de prendre en compte des informations incomplètes ou approchées, et le besoin d'échantillonnage (ou simulation) et d'exploration. L'absence d'un modèle enclenche le besoin d'échantillonner le MDP afin d'extraire des connaissances statistiques sur le modèle inconnu.

Il existe deux options. La première consiste à apprendre la fonction de transition et la fonction de récompense à partir de l'interaction avec l'environnement. Par la suite, la DP peut être appliquée. Ce type d'apprentissage est dit RL indirect ou RL basée sur un modèle. Le second type, dit RL direct, consiste à estimer les valeurs associées aux actions, sans estimer le modèle.

La plupart des approches de la littérature est de type RL direct, qui fera l'objet de cette section.

```

for each episode do
     $s \in S$  is initialized as the starting state
     $t := 0$ 
    repeat
        choose an action  $a \in A(s)$ 
        perform action  $a$ 
        observe the new state  $s'$  and received reward  $r$ 
        update  $\tilde{T}$ ,  $\tilde{R}$ ,  $\tilde{Q}$  and/or  $\tilde{V}$ 
        using the experience  $\langle s, a, r, s' \rangle$ 
         $s := s'$ 
    until  $s'$  is a goal state

```

FIGURE 4.1 – RL online [Wiering and van Otterlo, 2014]

L'algorithme de la Figure 4.1 montre le schéma général d'un algorithme RL direct. Il consiste simplement à sélectionner une action à prendre à partir de l'état courant, menant le système vers un état suivant. La boucle continue ainsi jusqu'à atteindre un état final. A chaque itération les fonctions T , R , Q et V sont mise à jour. Le défi de cet algorithme consiste à alterner entre exploitation et exploration afin de faire converger le processus global. L'exploitation consiste à profiter de la connaissance courante de l'environnement afin de prendre la meilleure action et donc la meilleure décision, et l'exploration consiste à visiter des actions non nécessairement les meilleures afin d'enrichir le modèle. L'exploration est donc une partie importante de RL pour apprendre le modèle. Il est enfin nécessaire de trouver un bon compromis entre exploration et exploitation.

4.3.2.1 Apprentissage par différence temporelle, TD (Temporal Difference learning)

L'approche TD consiste à apprendre l'estimation des valeurs en se basant sur les anciennes estimations. C'est un apprentissage à partir de l'expérience.

TD(0) est une famille des algorithmes TD, dont la règle de mise à jour de V est la suivante :

$$V_{k+1}(s) \leftarrow V_k(s) + \alpha(r + \gamma V_k(s') - V_k(s)) \quad (4.9)$$

où $\alpha \in [0, 1]$ est le tôt d'apprentissage.

Par la suite, l'algorithme par Q -learning a été proposé qui met à jour la fonction Q au lieu de V (voir la Figure 4.2), via la formule :

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \quad (4.10)$$

Require: discount factor γ , learning parameter α

initialize Q arbitrarily (e.g. $Q(s, a) = 0, \forall s \in S, \forall a \in A$)

for each episode **do**

s is initialized as the *starting* state

repeat

choose an action $a \in A(s)$ based on Q and an exploration strategy

perform action a

observe the new state s' and received reward r

$$Q(s, a) := Q(s, a) + \alpha \left(r + \gamma \cdot \max_{a' \in A(s')} Q(s', a') - Q(s, a) \right)$$

$s := s'$

until s' is a goal state

FIGURE 4.2 – RL online [Wiering and van Otterlo, 2014]

L'algorithme de Q -learning est utilisé dans un environnement sans stratégie au préalable (off-policy). Justement, quand une stratégie existe, la règle de SARSA a été proposé qui consiste à remplacer la règle (4.10) par :

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \quad (4.11)$$

Cette règle est utilisée quand l'action suivante est préalablement choisie par une stratégie donnée.

4.3.2.2 Méthode Monte-Carlo

4.3.3 Algorithmes de bandit

4.3.4 Le dilemme exploration/exploitation

4.4 Illustration sur un jeu ...

Chapitre 5

Fouille d'images

5.1 Rappel : réseaux convolutifs CNN

5.2 Détection d'objets

5.3 Tracking d'objets

5.4 ...

Chapitre 6

Fouille de données spatiotemporelles

6.1 Modélisation des séries temporelles

<https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>

Une série temporelle est une quantité mesurée séquentiellement dans un intervalle de temps. L'étude des séries temporelles, vise à caractériser ce qui s'est passé dans la série courante, et prédire ses valeurs futures.

Nous allons modéliser la série en une quantité aléatoire, en supposant que la série est une séquence de variables aléatoires (discrete-time stochastic process).

Voici quelques exemples de séries temporelles :

- Dans le domaine économique : le taux de croissance, le taux de chômage, les cours d'actions, etc.
- Dans le domaine démographique : le taux de natalité, les flux migratoires, etc.
- Et beaucoup d'autres : un électrocardiogramme, les températures, le trafic routier, l'énergie consommée, le niveau d'un cours d'eau, etc.

Toutes ces séries temporelles ne subiront pas les mêmes traitements, certaines seront simplement décrites (ex : analyse visuelle de courbes), d'autres analysées a posteriori via des modèles déterministes (ex : correction des variations saisonnières), d'autres enfin prévues (à l'aide de modèles issus de la physique, la statistique inférentielle, du machine learning, etc.).

Les méthodes stochastiques (ou probabilistes) : elles émergent au 20ème siècle, et ont connu depuis un véritable essor. On étudiera une de ces méthodes : le modèle ARMA.

Voyons ensemble quelques exemples typiques de séries temporelles.

Le graphique suivant représente le nombre mensuel de passagers aériens, en milliers, de janvier 1949 à décembre 1960. Cette série, fréquemment rencontrée dans les ouvrages traitant des séries temporelles.

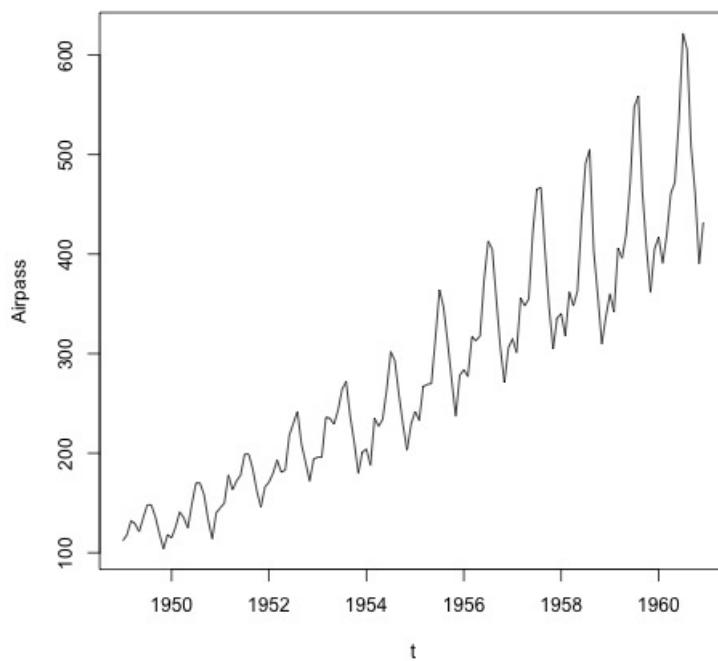


FIGURE 6.1 – Nombre de passagers aériens entre 1949 et 1960
(image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

On constate que son niveau moyen s'accroît avec le temps, de manière quasi-exponentielle ou polynomiale : on traduit encore cette évolution sous le terme de tendance. On observe par ailleurs qu'il y a des variations régulières autour du niveau moyen, de période 12 (correspondant à une période annuelle puisque la série est mensuelle) : on nomme cela l'effet saisonnier.

Le graphique suivant représente le nombre annuel de lynx capturés au Canada de 1821 à 1934.

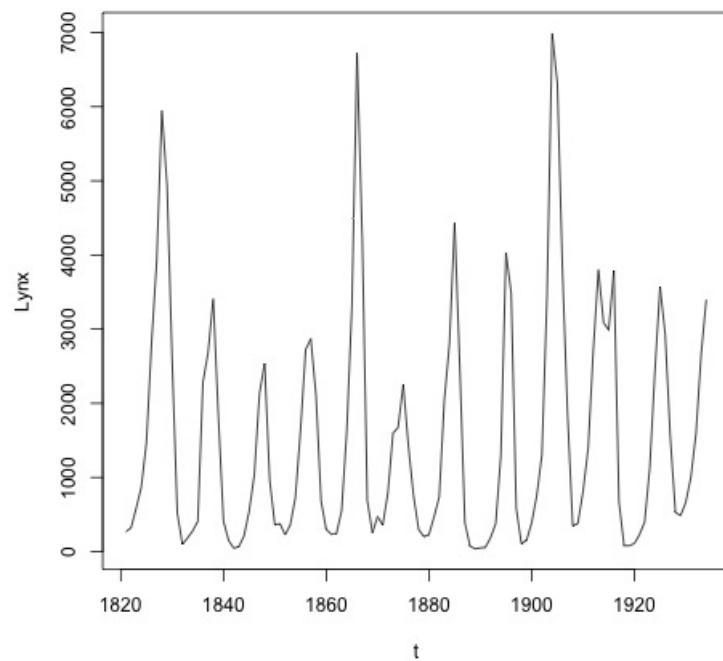


FIGURE 6.2 – Lynx capturés au Canada de 1821 à 1934
(image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

Cette série présente une tendance stable mais une double saisonnalité (saisonnalité et cycle).

Sur la série temporelle relative aux passagers aériens, nous obtiendrons les résultats suivants : Une correction en variations saisonnières (à l'aide de la régression linéaire ou de moyennes mobiles).

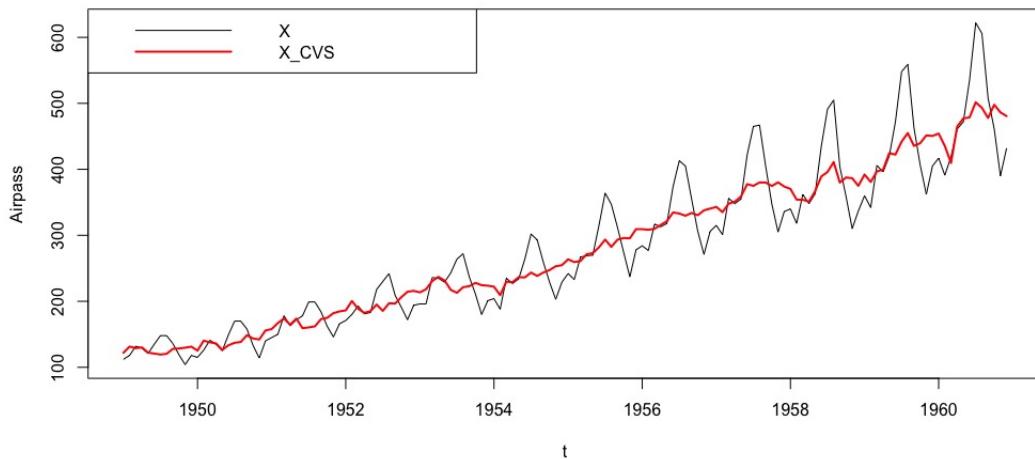


FIGURE 6.3 – Correction en variations saisonnières
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

Pour représenter les séries vues précédemment, on peut utiliser la commande ‘plot’. Vous pouvez lancer ces quelques lignes et vérifier le résultat obtenu.

```
# Série sunspot : nombre annuel de tâches solaires de 1790 à 1970
plot(sunspot.year,xlab="t",ylab="Sunspots")

# Bruit blanc gaussien de loi N(0,3^2)
# Pour les simulations effectuées dans ce document, on fixe arbitrairement
la racine (seed) à 1789.
set.seed(1789)
plot(ts(rnorm(100, sd=3), start=1, end=100), xlab="t", ylab="Bruit blanc gaussien
de variance 9")
abline(h=0)

# Série uspop : population des Etats-Unis, en millions, de 1790 à 1990
(Pas de temps décennal)
plot(uspop,xlab="t",ylab="Uspop")

# Série airpass : nombre mensuel de passagers aériens, en milliers,
de janvier 1949 à décembre 1960
# Série brute :
plot(AirPassengers,xlab="t",ylab="Airpass")
# Logarithme de la série airpass
plot(log(AirPassengers),xlab="t",ylab="Airpass")

# Série lynx : nombre annuel de lynx capturés au Canada, de 1821 à 1934
plot(lynx,xlab="t",ylab="Lynx")
```

```
x=AirPassengers  
y=log(x)
```

6.1.1 Comprenez les variations saisonnières

De nombreuses séries économiques présentent des comportements périodiques, rendant difficile la comparaison de deux instants successifs. On traite ainsi fréquemment de séries trimestrielles (ex : taux de croissance) ou mensuelles (ex : taux de chômage). On fait alors appel à une désaisonnalisation permettant d'obtenir des séries dites corrigées des variations saisonnières (CVS).

La figure suivante illustre le type de courbe désaisonnalisée qu'on peut trouver dans les publications :

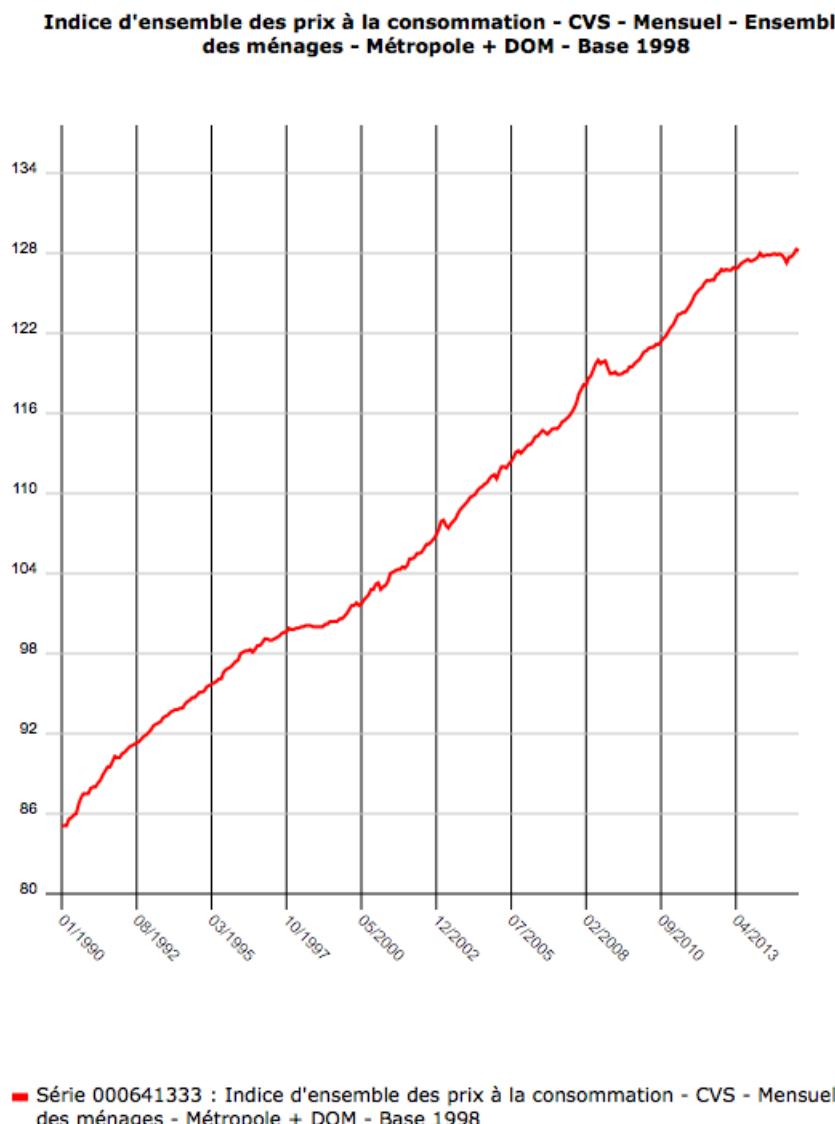


FIGURE 6.4 – Une courbe désaisonnalisée
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

Les méthodes couramment employées pour désaisonnaliser une série temporelle sont la régression linéaire et les filtres moyennes mobiles qu'on retrouve dans des algorithmes tels que Tramo-Seats et la famille des Xnum (dont X11).

On souhaite désaisonnaliser la série temporelle airpass à l'aide de la régression linéaire. On créera à cet effet les bases tendancielle et saisonnière :

```
t=1:144
```

```
for (i in 1:12)
{
```

```

su=rep(0,times=12)
su[i]=1
s=rep(su,times=12)
assign(paste("s",i,sep=""),s)
}

```

On effectue la régression linéaire (le modèle est transformé, comme vu en cours, afin de pallier le problème de colinéarité) sur la série Y_t :

```

reg=lm(y~t+s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12-1)
summary(reg)

```

On obtient l'estimation des différents paramètres :

```

##
## Call:
## lm(formula = y ~ t + s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8 +
## s9 + s10 + s11 + s12 - 1)
##
## Residuals:
## Min 1Q Median 3Q Max
## -0.156370 -0.041016 0.003677 0.044069 0.132324
##
## Coefficients:
## Estimate Std. Error t value Pr(>|t|)
## t 0.0100688 0.0001193 84.4 <2e-16 ***
## s1 4.7267804 0.0188935 250.2 <2e-16 ***
## s2 4.7047255 0.0189443 248.3 <2e-16 ***
## s3 4.8349527 0.0189957 254.5 <2e-16 ***
## s4 4.8036838 0.0190477 252.2 <2e-16 ***
## s5 4.8013112 0.0191003 251.4 <2e-16 ***
## s6 4.9234574 0.0191535 257.1 <2e-16 ***
## s7 5.0273997 0.0192073 261.7 <2e-16 ***
## s8 5.0181049 0.0192617 260.5 <2e-16 ***
## s9 4.8734703 0.0193167 252.3 <2e-16 ***
## s10 4.7353120 0.0193722 244.4 <2e-16 ***
## s11 4.5915943 0.0194283 236.3 <2e-16 ***
## s12 4.7054593 0.0194850 241.5 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ',' 1
##
## Residual standard error: 0.0593 on 131 degrees of freedom
## Multiple R-squared: 0.9999, Adjusted R-squared: 0.9999
## F-statistic: 9.734e+04 on 13 and 131 DF, p-value: < 2.2e-16

```

Les différents coefficients sont contenues dans :

```

reg$coefficients

## t s1 s2 s3 s4 s5 s6
## 0.0100688 4.7267804 4.7047255 4.8349527 4.8036838 4.8013112 4.9234574
## s7 s8 s9 s10 s11 s12
## 5.0273997 5.0181049 4.8734703 4.7353120 4.5915943 4.7054593

```

On revient aux coefficients initiaux :

```

a=mean(reg$coefficients[2:13])
b=reg$coefficients[1]
c=reg$coefficients[2:13]-mean(reg$coefficients[2:13])

```

et on obtient la série corrigée des variations saisonnières (en n'oubliant pas de passer à l'exponentiel pour revenir à X_t) :

```

y_cvs=y-(c[1]*s1+c[2]*s2+c[3]*s3+c[4]*s4+c[5]*s5+c[6]*s6+c[7]*s7+c[8]*s8+
c[9]*s9+c[10]*s10+c[11]*s11+c[12]*s12)
x_cvs=exp(y_cvs)
ts.plot(x,x_cvs,xlab="t",ylab="Airpass",col=c(1,2),lwd=c(1,2))
legend("topleft",legend=c("X","X_CVS"),col=c(1,2),lwd=c(1,2))

```

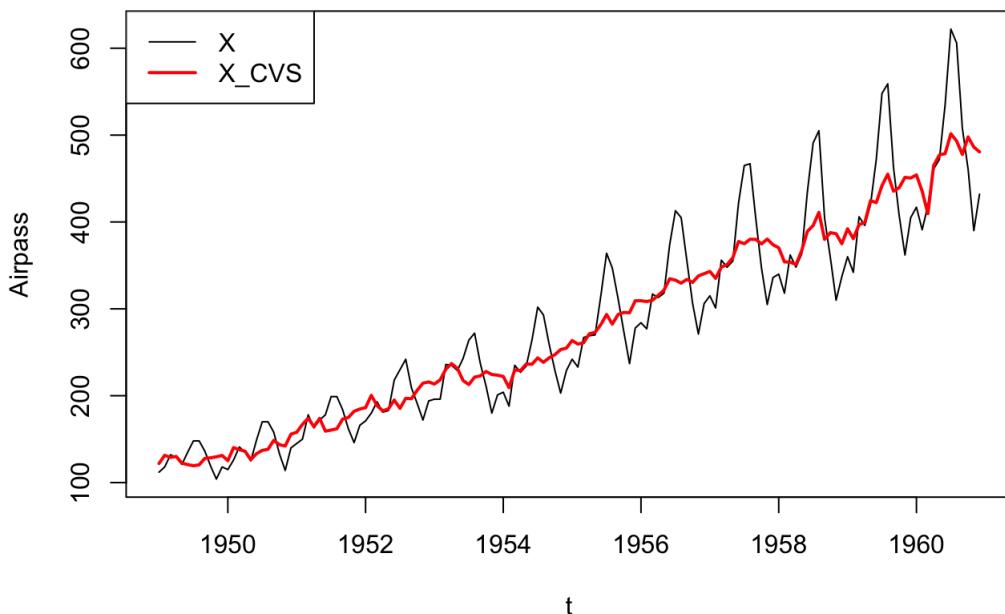


FIGURE 6.5 – ...
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

On constate que la série CVS présente des baisses en 1958 et 1960, pas forcément perceptibles à l'oeil nu sur la série brute. On pourra vérifier que la méthode des moyennes mobiles mettra en évidence les mêmes ruptures.

Si la régression linéaire est une solution "acceptable", on trouve beaucoup plus couramment des méthodes basées sur ce qu'on nomme les moyennes mobiles.

On utilise des filtres afin d'effectuer ces décompositions. De manière peu rigoureuse, un filtre est un "transformateur" d'un signal (temporel). On n'utilise ici que des filtres linéaires, et plus particulièrement les moyennes mobiles.

Une moyenne mobile M est une combinaison linéaire d'instants passés et futurs de la série temporelle :

$$MX_t = \theta_{-m_1}X_{t-m_1} + \dots + \theta_{-1}X_{t-1} + \theta_0X_t + \theta_1X_{t+1} + \dots + \theta_{m_2}X_{t+m_2} .$$

où les coefficients $\theta_{-m_1}, \dots, \theta_{m_2}$ sont tous des réels.

Notons d'ores et déjà que le produit de deux moyennes mobiles M_1 et M_2 ($M_1M_2 = M_1 \circ M_2$) est commutatif, associatif et distributif par rapport à l'addition. Cette propriété nous indique notamment que si l'on doit appliquer plusieurs moyennes mobiles sur une série temporelle, l'ordre d'application est indifférent.

Il existe de nombreuses moyennes mobiles et il nous faudra les caractériser, comprendre leurs propriétés. Dans la littérature, on trouve (et manipule) des moyennes mobiles qui peuvent être :

normalisées (il s'agit alors d'une moyenne au sens où on l'entend habituellement) si :

$$\sum_{i=-m_1}^{m_2} \theta_i = 1$$

centrées (on prend en compte autant d'instants passés que futurs) si ($m_1 = m_2 = m$)

symétriques (on donne des poids similaires aux instants passés et futurs de même ordre) si

$$\forall i \in \{1, \dots, m\} : \theta_{-i} = \theta_i .$$

On désigne par :

$$\Theta(z) = \sum_{i=0}^{m_1+m_2} \theta_{i-m_1} z^i$$

polynôme caractéristique de la moyenne mobile (comme son nom l'indique, il permet de caractériser ses propriétés).

6.1.2 Deux moyennes mobiles classiques

Voici deux des moyennes mobiles les plus employées :

— Pour $p = 2m + 1$, on définit la moyenne mobile simple (arithmétique) :

$$M_p X_t = \frac{1}{p} (X_{t-m} + \dots + X_{t-1} + X_t + X_{t+1} + \dots + X_{t+m})$$

C'est une moyenne mobile symétrique et normalisée.

- Pour $p = 2m$, on définit la moyenne mobile centrée (souvent on la nomme également, abusivement, moyenne mobile simple) :

$$M_{(2\times)p}X_t = \frac{1}{p} \left(X_{t-m+\frac{1}{2}} + \dots + X_{t-\frac{1}{2}} + X_{t+\frac{1}{2}} + \dots + X_{t+m-\frac{1}{2}} \right)$$

$$X_{t-\frac{1}{2}} = \frac{1}{2} (X_{t-1} + X_t)$$

$$M_{(2\times)p}X_t = \frac{1}{2p} (X_{t-m} + 2X_{t-m+1} + \dots + 2X_t + \dots + 2X_{t+m-1} + X_{t+m}) .$$

C'est une moyenne mobile centrée et normalisée.

6.1.3 Utiliser des moyennes mobiles pour corriger des variations saisonnières

L'idée consiste à appliquer une ou plusieurs moyennes mobiles afin de mettre en évidence, estimer, les différentes composantes d'une série temporelle. Supposons que :

$$X_t = T_t + S_t + \varepsilon_t$$

Si on applique une moyenne mobile M sur la série, nous obtenons (grâce à la propriété d'associativité) :

$$MX_t = MT_t + MS_t + M\varepsilon_t .$$

La visualisation de la série nous permet d'émettre des hypothèses sur la tendance (linéaire, quadratique, etc.) et la saisonnalité (période, forme, etc.).

L'enjeu est de trouver une moyenne mobile qui laisse la tendance invariante, qui absorbe la saisonnalité et qui réduit le résidu :

$$MT_t = T_t$$

$$MS_t = 0$$

$M\varepsilon_t$ faible (pouvoir de réduction de variance)

Dans ces conditions MX_t constitue une estimation de la tendance. La saisonnalité peut ensuite être estimée en travaillant sur la différence entre la série et la tendance ainsi estimée.

6.1.4 Evaluer les propriétés d'une moyenne mobile

On constate donc qu'il nous faut connaître les propriétés des moyennes mobiles afin de savoir quelles séries elles peuvent laisser invariantes ou absorber.

Définition 30 (Séries temporelles absorbées et invariantes) — Une série temporelle $(X_t)_{t \in \mathbb{Z}}$ est absorbée (ou annulée) par M si :

$$\forall t \in \mathbb{Z} : MX_t = 0 .$$

— Une série temporelle $(X_t)_{t \in \mathbb{Z}}$ est invariante par M si :

$$\forall t \in \mathbb{Z} : MX_t = X_t$$

Notons qu'une série temporelle invariante par M est absorbée par $I - M$ où I est l'opérateur identité.

L'étude des propriétés des moyennes mobiles repose sur le polynôme caractéristique, on est alors amenés à résoudre une équation de récurrence linéaire.

Il existe de nombreux résultats sur les propriétés, citons notamment :

— M 吸ue une composante saisonnière de période s s'annulant sur une période si et seulement si $\Theta(z)$ est divisible par $1 + z + \dots + z^{s-1}$.

— Les constantes sont invariantes par M si et seulement si M est normalisée.

Revenons à nos deux moyennes mobiles phares :

— M_p absorbe les saisonnalités de période $p = 2m + 1$ et laisse invariantes les tendances linéaires.

— $M_{(2\times)p}$ absorbe les saisonnalités de période $p = 2m$ et laisse invariantes les tendances linéaires.

Une dernière propriété est souvent attendue, celle de réduction de la variance. L'idée est que l'application d'une moyenne mobile sur une série temporelle puisse réduire sa variance, le lisser.

Mathématiquement, ce pouvoir se quantifie en étudiant le rapport de variances entre un bruit blanc fort $(\varepsilon_t)_{t \in \mathbb{Z}}$, c'est-à-dire une suite de variables aléatoires en entrée, et sa résultante en sortie de la moyenne mobile M :

$$\frac{\text{Var}(M\varepsilon_t)}{\text{Var}(\varepsilon_t)} = \sum_{i=-m_1}^{m_2} \theta_i^2 .$$

6.1.5 Algorithmes de traitement des moyennes mobiles

On a comptabilisé pendant trois semaines consécutives le nombre journalier de visiteurs d'un musée dont les jours de fermeture sont le samedi et le dimanche.

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
Première semaine	7	5	35	5	6
Deuxième semaine	8	9	45	8	9
Troisième semaine	10	11	42	9	11

FIGURE 6.6 – Trois semaines consécutives le nombre journalier de visiteurs d'un musée

(image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

Ces données sont encore représentées dans le tableau ci-dessous (avec en sus la série lissée par moyenne mobile).

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X_t	7	5	35	5	6	8	9	45	8	9	10	11	42	9	11
M_5X_t	.	.	11.6	11.8	12.6	?	?	?	?	?	16.0	16.2	16.6	.	.

FIGURE 6.7 – Série lissée par moyenne mobile
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

Le graphe semble mettre en évidence une tendance linéaire ainsi qu'une saisonnalité :

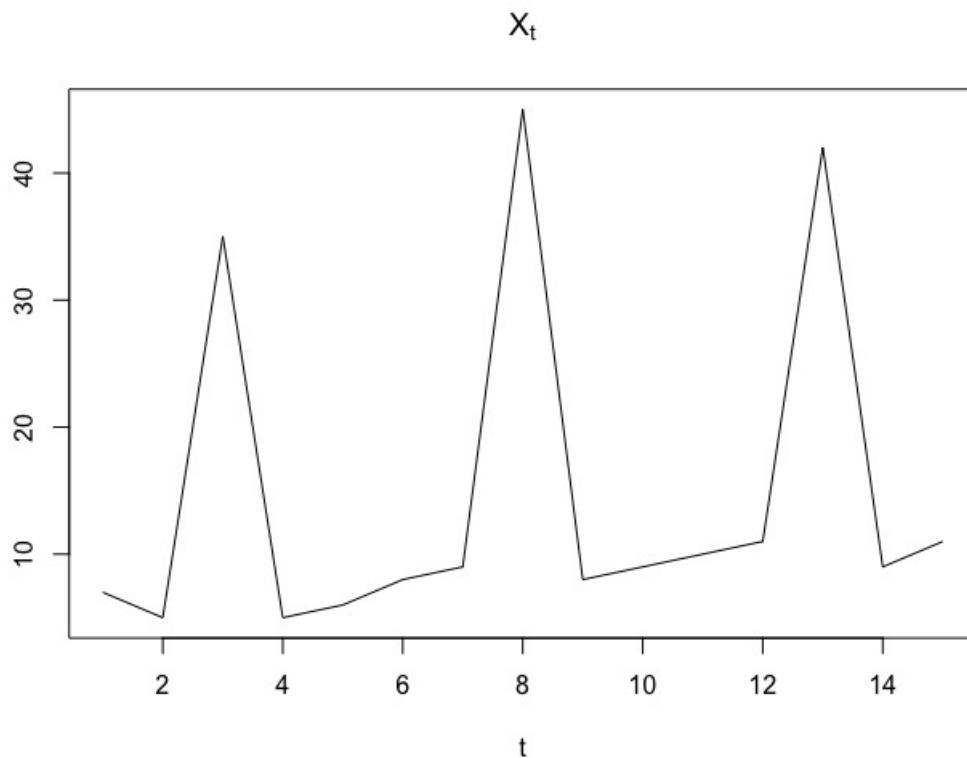


FIGURE 6.8 – Moyenne mobile
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

On choisit un modèle additif car les différences (maximum-minimum) semblent similaires sur chaque période (semaine). Les termes tendanciels pour $t \in \{3, \dots, 13\}$ figurent dans le tableau ci-après.

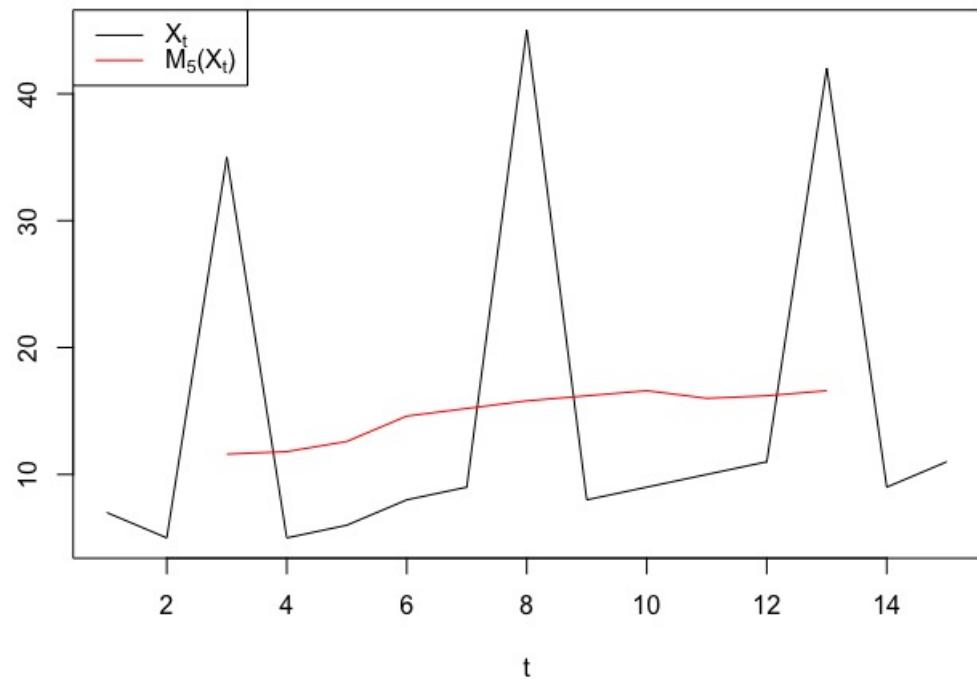


FIGURE 6.9 – Moyenne mobile
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

On calcule les coefficients saisonniers comme suit :

$$\begin{aligned}
 & c_{mercredi} \frac{23,4+29,2+25,4}{3} = 26 \\
 & c_{jeudi} \frac{-6,8-8,2}{2} = -7,5 \\
 & c_{vendredi} \frac{-6,6-7,6}{2} = -7,1 \\
 & c_{lundi} \frac{-6,6-6}{2} = -6,3 \\
 & c_{mardi} \frac{-6,2-5,2}{2} = -5,7
 \end{aligned}$$

On constate que :

$$\bar{c} = -0,12$$

Les coefficients saisonniers centrés sont :

$$\begin{aligned}
 & c'_{mercredi} = 26 - (-0,12) = 26,12 \\
 & c'_{jeudi} = -7,5 - (-0,12) = -7,38 \\
 & c'_{vendredi} = -7,1 - (-0,12) = -6,98 \\
 & c'_{lundi} = -6,3 - (-0,12) = -6,18 \\
 & c'_{mardi} = -5,7 - (-0,12) = -5,58
 \end{aligned}$$

La série corrigée des variations saisonnières, est ainsi représentée :

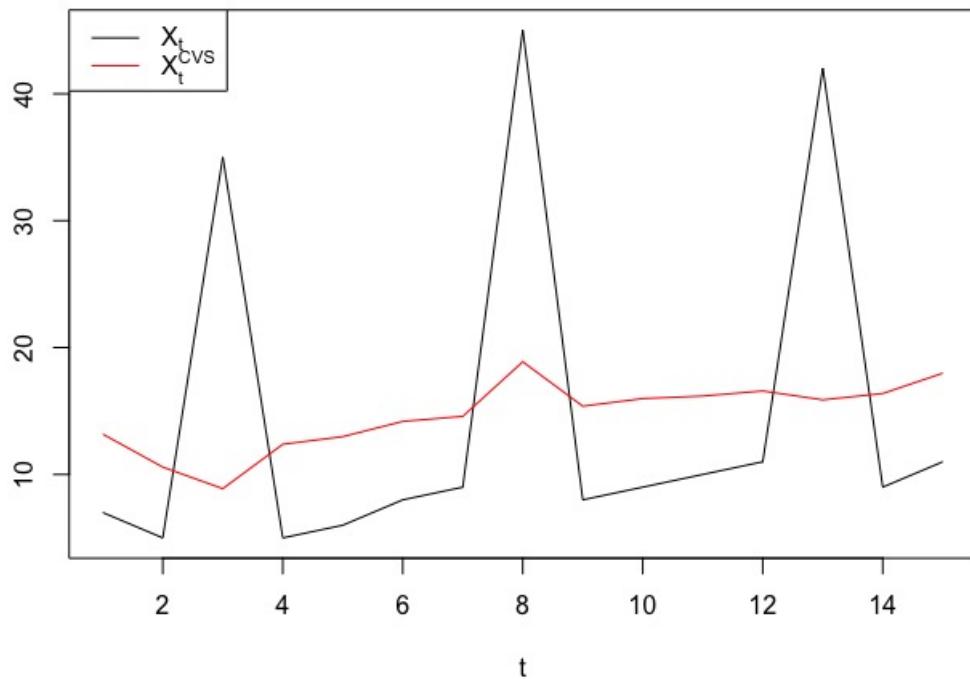


FIGURE 6.10 – Série corrigée des variations saisonnières
(image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

Comme travail pratique, on utilise la fonction ‘decompose’ :

```
decomp.x=decompose(x,type="multiplicative")
plot(decomp.x)
```

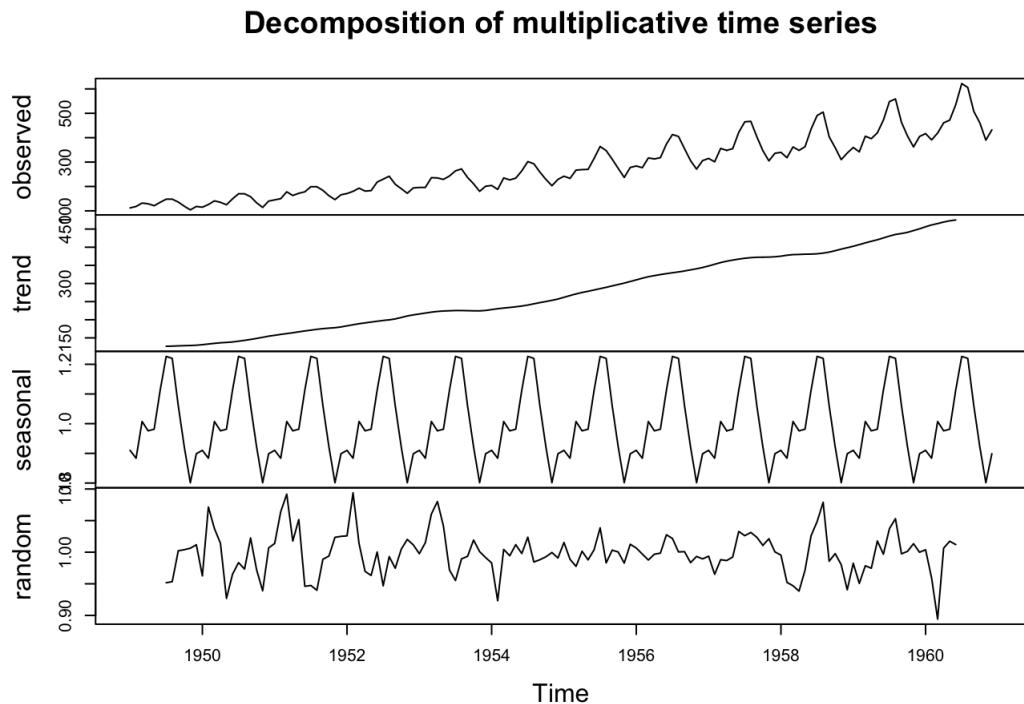


FIGURE 6.11 –
 (image de <https://openclassrooms.com/fr/courses/4525371-analysez-et-modelisez-des-series-temporelles>)

6.1.6 La notion de bruit blanc

A l'issue d'une modélisation, il nous faudrait idéalement obtenir un signal résiduel qui ne contient plus d'information temporelle. Dans le cadre des modèles ARMA, on souhaite que le résidu soit un bruit blanc (faible), c'est-à-dire sans dépendance temporelle linéaire. $(\varepsilon_t)_{t \in \mathbb{Z}}$ est un bruit blanc faible s'il est constitué de v.a.r telles que :

$$\begin{aligned} \forall t \in \mathbb{Z} \mathbb{E}(\varepsilon_t) &= 0 \\ \mathbb{E}(\varepsilon_t^2) &= \sigma^2 \\ \forall (t, t') \in \mathbb{Z}^2 / t \neq t' \operatorname{Cov}(\varepsilon_t, \varepsilon_{t'}) &= 0. \end{aligned}$$

6.1.7 Une notion fondamentale : la stationnarité

Dans de très nombreux cas, on ne peut pas renouveler la suite de mesures dans des conditions identiques. Pour que le modèle déduit à partir d'une suite d'observations ait un sens, il faut que toute portion de la trajectoire observée fournisse des informations sur la loi du processus et que des portions différentes, mais de même longueur, fournissent les mêmes indications. D'où la notion de stationnarité.

Un processus $(X_t)_{t \in \mathbb{Z}}$ est (faiblement) stationnaire si son espérance et ses auto-covariances sont invariantes par translation dans le temps :

$$\forall t \in \mathbb{Z} : \mathbb{E}(X_t) = \mu$$

$$\forall t \in \mathbb{Z}, \forall h \in \mathbb{Z} : \text{Cov}(X_t, X_{t-h})$$

ne dépend que de l'intervalle séparant les 2 instants h , pas de l'instant t .

Un processus stationnaire n'est pas obligatoirement maîtrisable ou sympathique. On sait par exemple que la somme de deux processus stationnaires n'est pas forcément stationnaire.

6.1.8 Les processus AR, MA et ARMA

6.1.8.1 Les processus AR

Sauf mention contraire, on considère dans la suite des processus centrés. Si le processus $(X_t)_{t \in \mathbb{Z}}$ n'est pas centré, on peut se ramener à ce cas de figure en centrant préalablement le processus (on travaillerait alors sur $Y_t = X_t - \mu_X$)

On dit qu'un processus $(X_t)_{t \in \mathbb{Z}}$ (stationnaire) est un processus processus AR (AutoRegressive) d'ordre p , noté AR(p), si

$$\forall t \in \mathbb{Z} : X_t = \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t$$

où

$$(\varphi_1, \dots, \varphi_p) \in \mathbb{R}^p$$

et

$$\varphi_p \neq 0.$$

La modélisation de

$$X_t$$

se résume à une relation linéaire liant aux p derniers instants.

Si $(X_t)_{t \in \mathbb{Z}}$ est un processus AR(p) alors ses autocorrélations partielles s'annulent à partir du rang $p+1$.

6.1.8.2 Les processus MA

On dit qu'un processus $(X_t)_{t \in \mathbb{Z}}$ est un processus MA (Moving Average) d'ordre q , noté MA(q), si

$$\forall t \in \mathbb{Z} : X_t = \varepsilon_t + \sum_{i=1}^q \theta_i \varepsilon_{t-i}$$

où

$$(\theta_1, \dots, \theta_q) \in \mathbb{R}^q$$

et $\theta_q \neq 0$.

On considère ici que le processus est la résultante d'une combinaison linéaire de perturbations decorrélatées (un bruit blanc et son passé).

Si $(X_t)_{t \in \mathbb{Z}}$ est un processus MA(q) alors ses autocorrélations simples s'annulent à partir du rang $q+1$.

6.1.8.3 Les processus ARMA

On dit qu'un processus $(X_t)_{t \in \mathbb{Z}}$ est un processus ARMA (AutoRegressive Moving Average) d'ordre (p, q) , noté ARMA(p, q), si

$(X_t)_{t \in \mathbb{Z}}$ est stationnaire,

$$\forall t \in \mathbb{Z} : X_t = \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t + \sum_{i=1}^q \theta_i \varepsilon_{t-i}$$

où $(\varphi_1, \dots, \varphi_p) \in \mathbb{R}^p$ $\varphi_p \neq 0$, $(\theta_1, \dots, \theta_q) \in \mathbb{R}^q$ et $\theta_q \neq 0$.

Il s'agit ici d'une synthèse des processus AR et MA. Les autocorrélations simples décroissent vers 0.

6.1.8.4 Les processus ARIMA, SARIMA, ...

Les modèles ARIMA permettent de modéliser des séries temporelles qui présentent une tendance polynomiale.

Les modèles SARIMA permettent de modéliser des séries qui présentent une saisonnalité. Estimer un modèle SARIMA se ramène en pratique à l'estimation d'un modèle ARMA sur la série différenciée.

6.2 Fouille des séries temporelles : algorithme SAX

(Réf. <https://www.philippe-fournier-viger.com/spmf/SAXTimeSeries.php>)

Calculer le SAX d'une série temporelle est une méthode réponduue pour transformer les séries temporelles en une représentation symbolique (voir Lin, J., Keogh, E., Wei, L., Lonardi, S. : Experiencing SAX : a novel symbolic representation of time series. Data Mining and Knowledge Discovery 15, 107?144 (2007)).

Par la suite, on peut faire appel aux algorithmes de fouille de séquences.

L'entrée de l'algorithme SAX est une suite de séries temporelles. Comme par exemple :

Name	Data points
ECG1	1,2,3,4,5,6,7,8,9,10
ECG2	1.5,2.5,10,9,8,7,6,5
ECG3	-1,-2,-3,-4,-5
ECG4	-2.0,-3.0,-4.0,-5.0,-6.0

L'algorithme SAX procède comme suit :

1. Chaque série temporelle est décomposée en w segments, chaque segment est remplacé par la valeur moyenne des points. Ce process est appelé "piecewise approximate aggregation (PAA)".
2. Chaque valeur est remplacée par un symbol. (Le nombre de symboles et de segments est fixé par l'utilisateur).
3. Comment les symboles sont choisis ? L'idée de SAX est de supposer que les valeurs suivent une loi normale, et ainsi choisir le symbole représentant l'intervalle de plusieurs valeurs.

Dans l'exemple précédent, le nombre de segments est fixé à 3, et le nombre de symbole à 4. L'algorithme SAX génère la base séquentielle :

```
Name Data points
ECG1_PAA b, c, d,
ECG2_PAA c, d, d
ECG3_PAA a, a, a
ECG4_PAA a, a, a
```

où

```
Symbol Interval of values represented by this symbol
a [-Infinity, -0.9413981789451658]
b [-0.9413981789451658, 2.4642857142857144]
c [2.4642857142857144, 5.869969607516595]
d [5.869969607516595, Infinity]
```

6.3 Fouille de trajectoires

(Réf. *Khatir Mohammed Rachid, Lebbah Yahia and Nourine Rachid. Une démarche incrémentale pour la fouille de trajectoires fréquentes, COSI'2018*)

Les données GPS ont connu récemment une croissance impressionnante induite par la diffusion des appareils dotés de capteurs GPS, et particulièrement les smartphones. Le nombre d'applications qui utilisent ces données ont aussi connu une croissance soutenue par les nombreux besoins qu'on peut exprimer sur la donnée géolocalisée. Les capteurs GPS aident à déterminer la trajectoire d'un usager d'un point de départ vers un point d'arrivée, grâce aux algorithmes efficaces de map-matching qui permettent de trouver le chemin dans le graphe de la carte qui soit le plus proche possible de la trajectoire GPS. Ces applications de navigation GPS ont permis de disposer d'une masse gigantesque de trajectoires GPS, qui renseignent sur différents comportements des usagers. Fouiller dans ces bases permet de trouver les régularités dans ces comportements, qui peuvent être exploitées par les experts. Dans ce papier, nous nous intéressons à la problématique de recherche des sous-trajectoires fréquentes dans une base de trajectoires, qui est une forme de régularité révélant les chemins les plus empruntés par les usagers. Ces sous-trajectoires fréquentées, peuvent être exploitées notamment pour les réaménagements de la circulation urbaine afin de la rendre plus fluide.

Les trajectoires sont exprimées en terme d'une suite de positions GPS. Nous définissons ci-dessous la donnée atomique d'un point GPS, ainsi que les éléments d'une base de données de fouille. Pour illustrer les définitions nous utilisons l'exemple de la base de trajectoires de la Figure 6.13 décrite dans la table ???. La Figure 6.13 montre un exemple de six trajectoires synthétiques distinctes. La carte utilisée dans cette illustration est tirée d'Open Street Map¹.

1. <https://www.openstreetmap.org/>

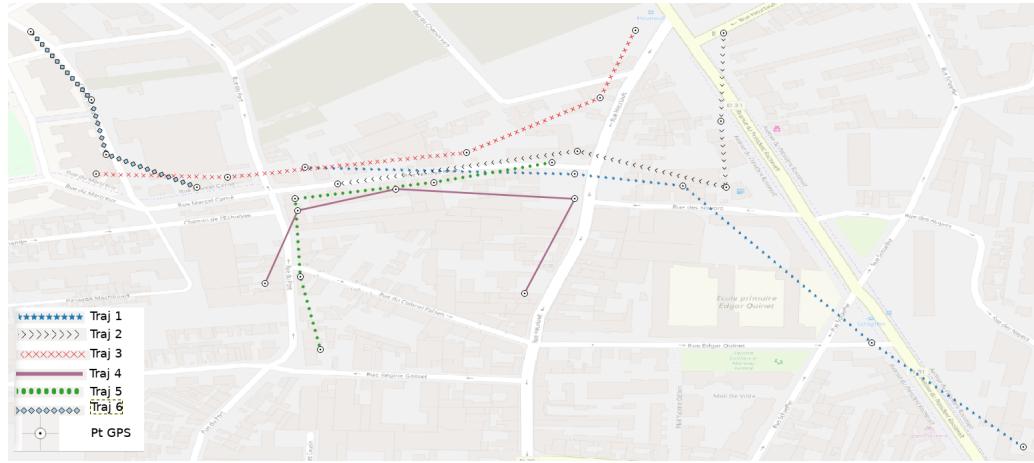


FIGURE 6.12 – Trajectoires synthétiques sur fond OSM

Trajectoires (tid)	Locations Point (Pt.id, Latitude, Longitude)					
Traj 1	Pt.id	pt1	pt2	pt3	pt4	pt5
	Lat	48.915566	48.916455	48.917794	48.917893	48.917949
	Lon	2.381197	2.379909	2.378300	2.377377	2.375081
Traj 2	Pt.id	pt6	pt7	pt8	pt9	pt10
	Lat	48.919092	48.918344	48.917780	48.918090	48.917808
	Lon	2.378643	2.378622	2.378665	2.377399	2.375360
Traj 3	Pt.id	pt11	pt12	pt13	pt14	pt15
	Lat	48.919120	48.918542	48.918076	48.917865	48.917893
	Lon	2.377892	2.377592	2.376455	2.374416	2.3733
Traj 4	Pt.id	pt16	pt17	pt18	pt19	pt20
	Lat	48.916878	48.917682	48.917766	48.917583	48.916962
	Lon	2.376948	2.377377	2.375854	2.375017	2.374738
Traj 5	Pt.id	pt21	pt22	pt23	pt24	pt25
	Lat	48.917992	48.917823	48.917682	48.917019	48.916398
	Lon	2.377184	2.376176	2.374995	2.375038	2.375210
Traj 6	Pt.id	pt26	pt27	pt28	pt29	–
	Lat	48.917780	48.918062	48.918528	48.919106	–
	Lon	2.374159	2.373386	2.373257	2.372742	–

Définition 31 (Point GPS) Une position GPS d'un objet est identifiée par :

Id est l'identifiant de l'objet en mouvement.

Localisation contient les coordonnées spatiales (longitude, latitude, altitude) de l'objet.

Timestamp est l'horodatage du prélèvement de la donnée.

Vitesse est exprimée en Km/h.

Précision au moment du prélèvement par le capteur GPS de la localisation de l'objet qui dépend de la force du signal GPS (nombre de satellites couverts).

Définition 32 (Segment, trajectoire, base de trajectoires) Un Segment d'une trajectoire est défini comme étant le lien de deux points GPS consécutifs d'une même

trajectoire. Une Trajectoire T est une séquence chronologique, constituée de points GPS. $T = [T_1, T_2, \dots, T_{n(T)}]$ où $\forall i, T_i$ est un point GPS. $n(T)$ est la longueur de la trajectoire. Une base de trajectoires TDB est un ensemble de trajectoires (tid, T) , où tid est un identifiant unique et T est une trajectoire.

La démarche souvent adoptée dans la littérature consiste à discréteriser l'espace bidimensionnel englobant l'ensemble des trajectoires sous forme d'un pavage régulier d'éléments, notamment d'hexagones. Par la suite chacune des trajectoires de points GPS est transformée en une suite de pavés contenant la trajectoire en question. Par ce procédé, chaque trajectoire devient une simple séquence d'items, permettant de transformer la problématique de fouille de trajectoires GPS en une fouille de séquences d'items qui peut être appréhendée avec les algorithmes connus de fouille de séquences, particulièrement l'algorithme PrefixSpan.

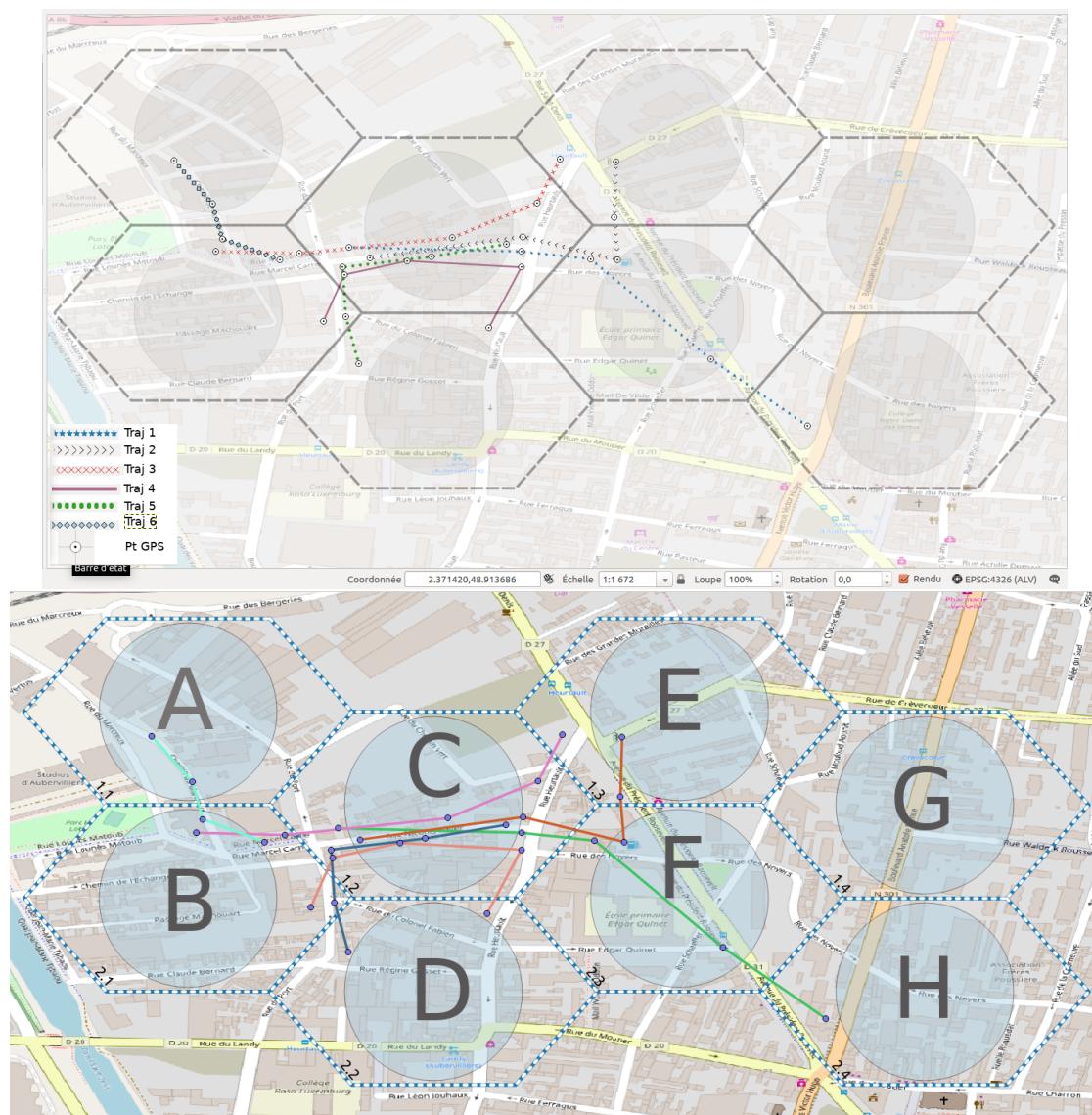


FIGURE 6.13 – Illustrations des démarches de pavage de la littérature

6.3.1 Similarité entre deux trajectoires : distance de Fréchet

Copy/paste from <http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2002/StephanePelletier/>

A polygonal curve $P : [0, N]$ is a continuous and piecewise linear curve made of N connected segments. Such a curve can be parametrized with a parameter $a \in \mathbb{R}$ such that $P(a)$ refers to a given position on the curve, with $P(0)$ referring to the first vertex of the polygonal curve and $P(N)$ referring to its last vertex, as shown in the Figure 6.14.

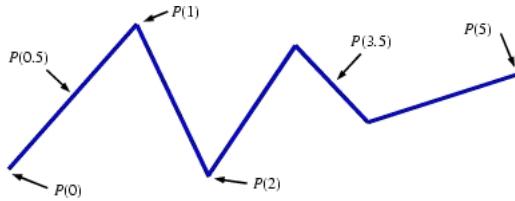


FIGURE 6.14 – Parametrization of a polygonal curve [StephanePelletier]

We consider a polygonal curve Q of length M . The mutual position on both curves P and Q can be modelled as a function of t by using two continuous and increasing functions $\alpha(t)$ and $\beta(t)$, where $\alpha(0) = 0$, $\alpha(1) = N$, $\beta(0) = 0$ and $\beta(1) = M$. The position on P as a function of time is given by $P(\alpha(t))$ and the position of the dog by $Q(\beta(t))$. While many functions $\alpha(t)$ and $\beta(t)$ can be used, some of them will cause the distance between the two curves to be large at some instant, whereas other functions will not. Finding the Fréchet distance actually corresponds to find two parametrization functions $\alpha(t)$ and $\beta(t)$ that minimize the maximum distance between the man and his dog.

Mathematically, the Fréchet distance between two curves is defined as

$$\delta_F(P, Q) = \min_{\substack{\alpha: [0, 1] \rightarrow [0, N] \\ \beta: [0, 1] \rightarrow [0, M]}} \max_{t \in [0, 1]} d(P(\alpha(t)), Q(\beta(t))).$$

where $\alpha(t)$ and $\beta(t)$ range over continuous and increasing functions with $\alpha(0) = 0$, $\alpha(1) = N$, $\beta(0) = 0$ and $\beta(1) = M$ only.

This equation reads like : for every possible function $\alpha(t)$ and $\beta(t)$, find the largest distance between the two curves as two walkers walk along their respective path ; finally, keep the smallest distance found among these maximum distances. For simplicity, we'll take $d(a, b)$ as the Euclidian distance between a and b .

6.3.1.1 Decision problem

Given : polygonal curves P and Q and some $\epsilon \geq 0$

Decide : whether $\delta_F(P, Q) \leq \epsilon$.

In order to solve the decision problem, let us first consider the case where $p = q = 1$, i.e. P and Q are just simple segments. We define

$$F_\epsilon = \{(s, t) \in [0, 1]^2 | d(P(s), Q(t)) \leq \epsilon\}$$

that describes all pairs of points, one on P , one on Q , whose distance is at most ϵ . Figure 6.15 shows line segments P, Q , a distance $\epsilon > 0$, and F_ϵ being the white area within the unit square ; subsequently called the free space.

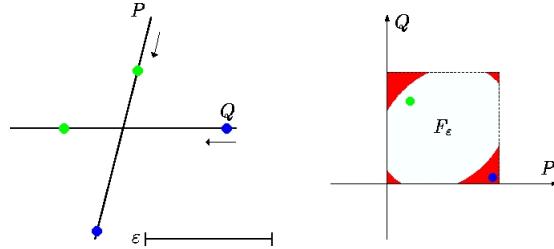


FIGURE 6.15 – Free space of two segments [StephanePelletier]

As the distance between the two green points on the curves is smaller than ϵ , the corresponding green point in the unit square belongs to the free space. Conversely, the blue point in the unit square does not belong to the free space, as the distance between the two corresponding blue points on the curves is larger than ϵ . Also, the free space corresponding to two line segments is the intersection of the unit square with an ellipse, possibly degenerated to the space between two parallel lines. The important thing to remember here is that the free space of two line segments is convex.

We can extend previous equation to arbitrary polygonal curves P, Q , of length p, q respectively :

$$F_\epsilon = \{(s, t) \in [0, p] \times [0, q] | d(P(s), Q(t)) \leq \epsilon\}.$$

The free space corresponding to the curves P and Q is the combination of the free spaces of all pairs containing one segment of P and one segment of Q . Figure 6.16 shows two polygonal curves and their corresponding free space for a given value of ϵ .

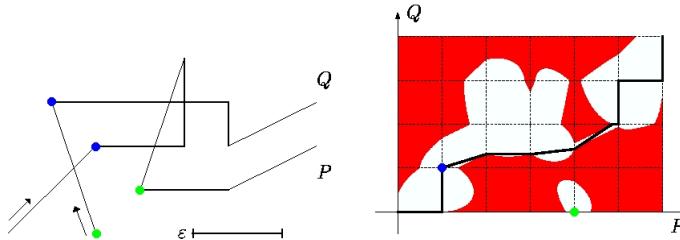


FIGURE 6.16 – Free space of two polygonal curves [StephanePelletier]

The algorithm is based on the following straightforward observation :

Proposition 6 *For polygonal curves P and Q we have $\delta_F(P, Q) \leq \epsilon$, exactly if there exists a curve within the corresponding F_ϵ from $(0, 0)$ to (p, q) which is monotone in both coordinates.*

In the man-dog example, the monotonicity condition corresponds to the fact that none of them is allowed to go backwards. Figure 6.16 shows such a curve proving that for that example the Fréchet distance is lower than ϵ .

Now, let $F_\epsilon(i, j)$ with $1 \leq i \leq p$, $1 \leq j \leq q$ refer to the free cell corresponding to the segment $P(i-1)P(i)$ and the segment $Q(i-1)Q(i)$. What we are interested in at this point is to find the intersection of the free space with the contour of each $F_\epsilon(i, j)$. If the free space intersects the edge of a cell, it means that it is possible to enter in that cell through this edge. As the free space of each unit cell is convex, it is an easy matter to determine if there is a monotone curve that goes from $(0, 0)$ to (p, q) by computing all the intersections of the free space with the contour of each cell. Figure 6.17 illustrates the values that must be calculated in order to identify the passages between neighboring cells. $L_{i,j}^F$ refers to the left line segment bounding $F_\epsilon(i, j)$ and $B_{i,j}^F$ refers to the bottom line segment.

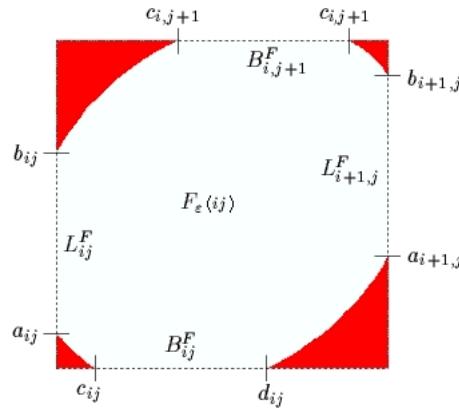


FIGURE 6.17 – Values that must be calculated [StephanePelletier]

Furthermore, we define

$$R_\epsilon = \{(s, t) \in F_\epsilon \mid \exists \text{a monotone curve within } F_\epsilon \text{ from } (0, 0) \text{ to } (s, t)\}.$$

The following figure shows the difference between F_ϵ and R_ϵ . One can see that all the zones that were not reachable by a monotone curve in F_ϵ have been removed in R_ϵ . Hereafter, R_ϵ will be referred to as the monotone free space.

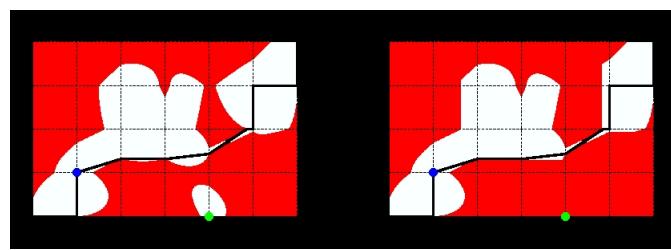


FIGURE 6.18 – Monotone free space of two polygonal curves [StephanePelletier]

R_ϵ can easily be computed from F_ϵ . We will use almost the same notation as in figure 6.17 to designate the values that have to be calculated for the monotone free

space. That is, $L_{i,j}^R$ will refer to the left line segment bounding $R_\epsilon(i,j)$ and $B_{i,j}^R$ will refer to the bottom line segment.

Algorithm 1 Fréchet existence

```

1: for each feasible pair  $(i, j)$  do
2:   compute  $L_{i,j}^F$  and  $B_{i,j}^F$ 
3: end for
4: for  $i = 1$  to  $p$  do
5:   determine  $B_{i,1}^R$ 
6: end for
7: for  $i = 1$  to  $q$  do
8:   determine  $L_{1,j}^R$ 
9: end for
10: for  $i = 1$  to  $p$  do
11:   for  $j = 1$  to  $q$  do
12:     construct  $L_{i+1,j}^R$  and  $B_{i,j+1}^R$  from  $L_{i,j}^R$ ,  $B_{i,j}^R$ ,  $L_{i+1,j}^F$ ,  $B_{i,j+1}^F$ 
13:   end for
14: end for
15: answer yes if  $(p, q)$  in  $L_{p+1,q}^R$ , no otherwise.

```

Above algorithm decides in $O(pq)$.

6.3.1.2 Frechet distance

Now, let us consider the problem of really computing the Frechet distance. Assume that we start with $\epsilon = 0$ and continuously increase ϵ . Then the free space becomes larger and larger and we want to determine the smallest ϵ such that it contains a monotone curve from $(0, 0)$ to (p, q) . Observe that this can occur only in one of the following cases :

1. ϵ is minimal with $(0, 0) \in F_\epsilon$ and $(p, q) \in F_\epsilon$
2. ϵ is minimal with $L_{i,j}^F$ or $B_{i,j}^F$ becomes nonempty for some pair (i, j) .
3. ϵ is minimal with $a_{i,j} = b_{k,j}$ or $c_{i,j} = d_{i,k}$ for some i, j, k .

Clearly, case 1 states that $\delta_F(P, Q)$ must be greater or equal to the distance between the starting points of P and Q and also greater or equal to the distance between their ending points. Case 1 corresponds to the opening of a new passage between two neighboring cells and case 3 corresponds to the opening of a new vertical or horizontal passage within the diagram. The following figure illustrates case 3 for a horizontal passage.



FIGURE 6.19 – Horizontal passage that opens [StephanePelletier]

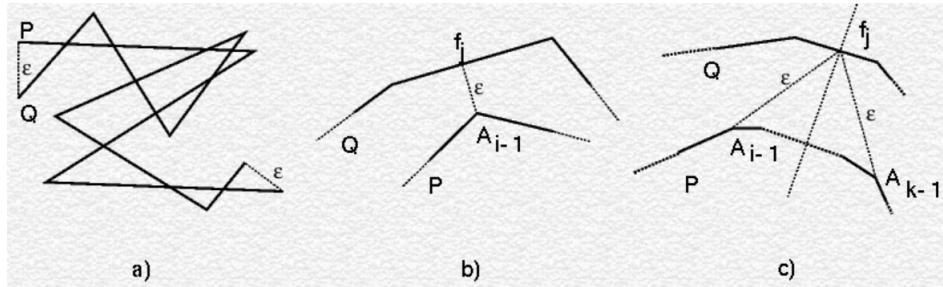


FIGURE 6.20 – Geometric meaning of critical values [StephanePelletier]

The following figure shows the geometric meaning of ϵ in cases 1, 2 and 3.

There are $O(p^2q + pq^2)$ such critical values of ϵ , namely the distance between starting points and endpoints of P and Q (case a), the distances between vertices of one curve and edges of the other (case b), and the common distance of two vertices of one curve to the intersection point of their bisector with some edge of the other (case c). Each one of these values can be computed in $O(1)$ time. So, we obtain the following algorithm for computing $\delta_F(P, Q)$.

Algorithm 2 Continuous Frechet distance

- 1: Determine all critical values of ϵ
 - 2: Sort them
 - 3: Do a binary search on the sorted sequence in each search step solving the decision problem, continuing with the half containing smaller critical values if it has a positive answer and with the half containing larger critical values otherwise.
-

6.3.1.3 Discrete Frechet

[Definitions taken from "Computing Discrete Fréchet Distance", Thomas Eiter, Heikki Mannila, University of Helsinki, Finland.]

We define a curve as a mapping $f : [a, b] \rightarrow V$, where $a, b \in \mathbb{R}$ and $a \leq b$. (V, d) is a metric space. Consider two curves $f : [a, b] \rightarrow V$ and $g : [a', b'] \rightarrow V$. Their Frechet distance is defined as :

$$\delta_F(f, g) = \underset{\substack{\alpha: [0,1] \rightarrow [a,b] \\ \beta: [0,1] \rightarrow [a',b']}}{\preceq} \max_{t \in [0,1]} d(f(\alpha(t)), g(\beta(t))).$$

Let $P : [0, n] \rightarrow V$ and $Q : [0, n] \rightarrow V$ be two polygonal curves. We denote $\sigma(P) = (P(0), P(1), \dots, P(p))$ (resp. $\sigma(Q) = (Q(0), Q(1), \dots, Q(q))$) the sequence of endpoints of the line segments of P (resp. Q).

A coupling L between P and Q is a sequence :

$$(P_{a_1}, Q_{b_1}), (P_{a_2}, Q_{b_2}), \dots, (P_{a_m}, Q_{b_m})$$

of distinct pairs from $\sigma(P) \times \sigma(Q)$ such that $a_1 = 1, b_1 = 1, a_m = p, b_m = q$, and for all $i = 1, \dots, q$, we have $a_{i+1} = a_i$ or $a_{i+1} = a_i + 1$, and $b_{i+1} = b_i$ or $b_{i+1} = b_i + 1$. The

length $\|L\|$ of coupling L is

$$\|L\| = \max_{i=1..m} d(P_{a_i}, Q_{b_i})$$

Given polygonal curves P and Q , their discrete Fréchet distance is defined

$$\delta_{dF}(P, Q) = \min\{\|L\| \mid L \text{ is a coupling between } P \text{ and } Q\}.$$

It can be proven that $\delta_F(P, Q) \leq \delta_{dF}(P, Q)$.

The dynamic programming algorithm is given in Figure 6.21.

Function $dF(P, Q)$: **real**;

input: polygonal curves $P = (u_1, \dots, u_p)$ and $Q = (v_1, \dots, v_q)$.

return: $\delta_{dF}(P, Q)$

ca : **array** [1.. p , 1.. q] of **real**;

function $c(i, j)$: **real**;

begin

if $ca(i, j) > -1$ **then return** $ca(i, j)$

elsif $i = 1$ **and** $j = 1$ **then** $ca(i, j) := d(u_1, v_1)$

elsif $i > 1$ **and** $j = 1$ **then** $ca(i, j) := \max\{c(i-1, 1), d(u_i, v_1)\}$

elsif $i = 1$ **and** $j > 1$ **then** $ca(i, j) := \max\{c(1, j-1), d(u_1, v_j)\}$

elsif $i > 1$ **and** $j > 1$ **then** $ca(i, j) :=$

$\max\{\min(c(i-1, j), c(i-1, j-1), c(i, j-1)), d(u_i, v_j)\}$

else $ca(i, j) = \infty$

return $ca(i, j)$;

end; /* function c */

begin

for $i = 1$ **to** p **do for** $j = 1$ **to** q **do** $ca(i, j) := -1.0$;

return $c(p, q)$;

end.

FIGURE 6.21 – Algorithm computing the coupling measure [Thomas Eiter and Heikki Mannila, Computing Discrete Fréchet Distance, CD-TR 94/64, April 25, 1994]

6.3.2 Mapping d'une trajectoire sur une carte

Chapitre 7

Compléments et applications

7.1 Détection d'anomalies (outlier detection)

Réf. : <https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-anomalies.pdf>

La détection d'anomalies (outliers) est un enjeu ancien et majeur des applications industrielles de la Statistique notamment pour la détection d'une défaillance ou défaut de fabrication.

Quelque soit la méthode utilisée, une anomalie des données est toujours définie, implicitement ou explicitement relativement à un modèle sous-jacent. Le choix de la méthode et donc de ce modèle dépend complètement du contexte, de l'objectif visé, des données disponibles, de leurs propriétés.

- 7.1.1 Valeurs manquantes et valeurs aberrantes ?
 - 7.1.2 Analyse des valeurs extrêmes
 - 7.1.3 Méthodes probabilistes
 - 7.1.4 Méthodes à base de clustering
 - 7.1.5 Méthodes pour des données catégorielles
 - 7.1.6 ...
- 7.2 Apprentissage profond (deep learning) et découverte de motifs
 - 7.3 Clustering : Manifold learning
 - 7.4 Modèles de mélange et algorithme EM (Expectation Maximization)
 - 7.4.1 Modèle de mélange
 - 7.4.2 Modèle de mélange gaussien
 - 7.4.3 Algorithme EM (Expectation Maximization)
 - 7.5 Kernel density estimation (Estimation par noyau)
 - 7.6 ...

Chapitre 8

Rappels : classification et clustering

8.1 Régression linéaire [Biernat and Lutz, 2015]

On recherche la meilleure fonction hypothèse qui approximera les données d'entrée :

$$\text{valeur d'entrée } x \xrightarrow{\text{hypothèse } h} \text{valeur de sortie } y.$$

On offre plusieurs variables en entrée du problème, qui constituent autant de degrés de liberté à la fonction h pour approximer au mieux les données d'entrée.

Avec ces nouvelles hypothèses, h prend une forme plus générale pour n variables d'entrée :

$$h(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \dots + \theta_n X_n.$$

Les données en entrée se représentent sous forme d'une matrice de dimension $m \times n$:

$$X = \begin{pmatrix} X_{1,1}, \dots, X_{1,n} \\ \dots \\ X_{m,1}, \dots, X_{m,n} \end{pmatrix}$$

$X_{i,j}$ correspond à la valeur prise par la variable j de l'observation i .

La fonction coût est la suivante :

$$J(\theta) = \frac{1}{2m} \sum_{i=1..m} (h(x_i) - y_i)^2 \quad (8.1)$$

Il est aussi nécessaire de normaliser les données X via la formule standard :

$$X_{std} = \frac{X - \min(X)}{\max(X) - \min(X)}.$$

Pour trouver le min de cette fonction, il suffit de trouver les racines de sa dérivée, via la méthode de gradient/Newton.

Mais au fait, on peut trouver le min d'une façon analytique comme suit.

Si on pose $\forall i \in 1..m, x_{i,0} = 1$, on peut réécrire la forme de la régression comme suit :

$$h(x_i) = \theta_0 x_{i,0} + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \dots + \theta_n x_{i,n}.$$

On peut écrire :

$$X = \begin{pmatrix} 1, X_{1,1}, \dots, X_{1,n} \\ 1, \dots \\ 1, X_{m,1}, \dots, X_{m,n} \end{pmatrix}$$

Posons :

$$\Theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{pmatrix}$$

et

$$Y = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_m \end{pmatrix}$$

On peut définir l'erreur :

$$r_i = y_i - \sum_{j=0..n} x_{i,j} \theta_j.$$

J peut maintenant s'exprimer :

$$J = \sum_{i=1..m} r_i^2.$$

Sa dérivée par rapport à θ_j est :

$$\frac{\partial J}{\partial \theta_j} = 2 \sum_{i=1..m} r_i \frac{\partial r_i}{\partial \theta_j}$$

Si on pose la condition nécessaire pour atteindre le min :

$$\frac{\partial J}{\partial \theta_j} = 0$$

On a :

$$2 \sum_{i=1..m} r_i \frac{\partial r_i}{\partial \theta_j}$$

On a donc

$$\sum_{i=1..m} \sum_{k=0..n} x_{i,j} x_{i,k} \theta_k = \sum_{i=1..m} \sum_{k=0..n} x_{i,j} y_i$$

D'une façon plus compact :

$$(X^T X) \theta = X^T Y$$

On obtient la solution :

$$\theta = (X^T X)^{-1} X^T Y$$

Dite méthode des "moindres carrés".

Pour une telle résolution analytique, les statisticiens imposent des hypothèses fortes sur les caractéristiques du vecteur r (par exemple : un modèle statistique repose également sur d'autres hypothèses de modélisation). Il doit en effet se comporter comme une suite de variables indépendantes de même loi normale $N(0, \sigma^2)$.

Lorsque les hypothèses concernant r ne peuvent être respectées (non normalité des résidus, non indépendance ? comme c'est souvent le cas pour les séries temporelles ?, etc.), on emploie des alternatives aux moindres carrés ordinaires telles que les moindres carrés généralisés ou le maximum de vraisemblance, pour les plus connues d'entre elles.

Pour finir, remarquons que la méthode analytique a une complexité de type $O(n^3)$. Ce qu'il faut comprendre, c'est qu'elle peut devenir instables numériquement et couteuses à mettre en oeuvre pour des instances de grandes dimensions (c'est-à-dire avec beaucoup de variables). Rappelons que la méthode de la descente de gradient reste une bonne alternative, généralisable et très appropriée à des analyses à grande échelle.

8.2 Régression polynomiale [Biernat and Lutz, 2015]

La régression polynomiale est une extension de la régression linéaire multivariée. Elle permet de lier les variables par un polynôme de degré k . Par exemple, un modèle polynomial de degré 2 à deux variables explicatives :

$$h(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_1^2 + \theta_4 X_2^2.$$

L'introduction de termes polynomiaux dans un modèle de régression permet donc de modéliser simplement des relations potentiellement très complexes.

8.3 Arbres de décision

Nous illustrons ci-dessous un exemple d'un arbre de décision.

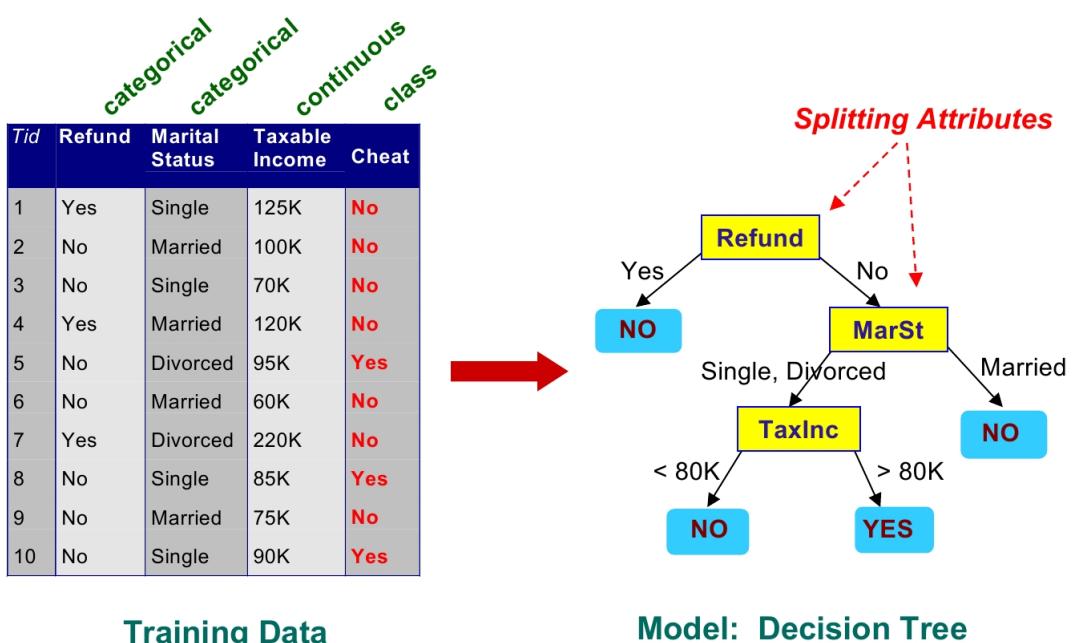


FIGURE 8.1 – Illustration d'un exemple d'un arbre de décision [Tan et al., 2005]

On peut produire un deuxième arbre de décision sur les mêmes données :

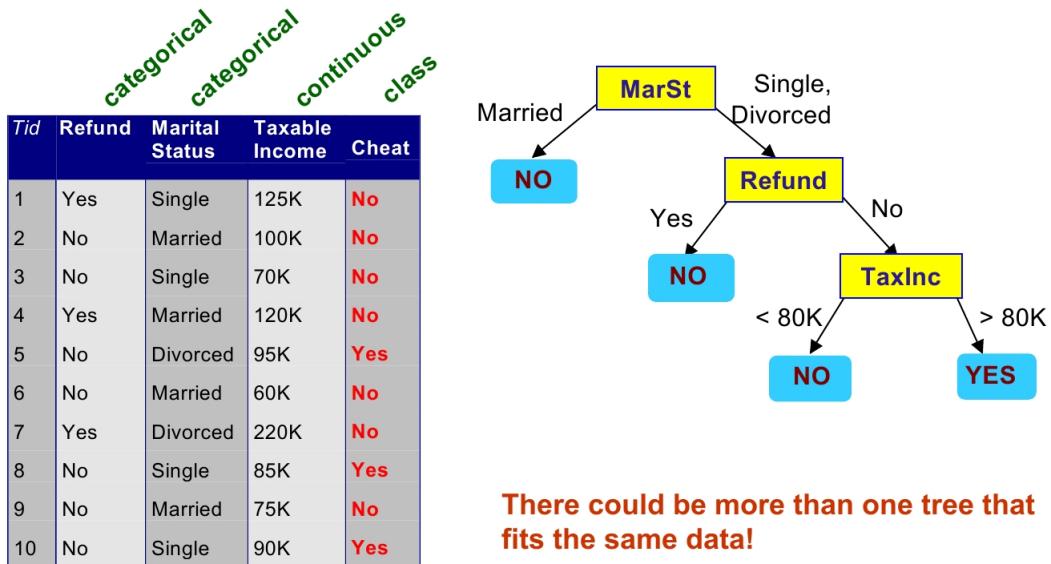


FIGURE 8.2 – Illustration d'un 2ème arbre de décision pour les mêmes données [Tan et al., 2005]

Le processus global de la méthode des arbres de décision est illustrée comme suit :

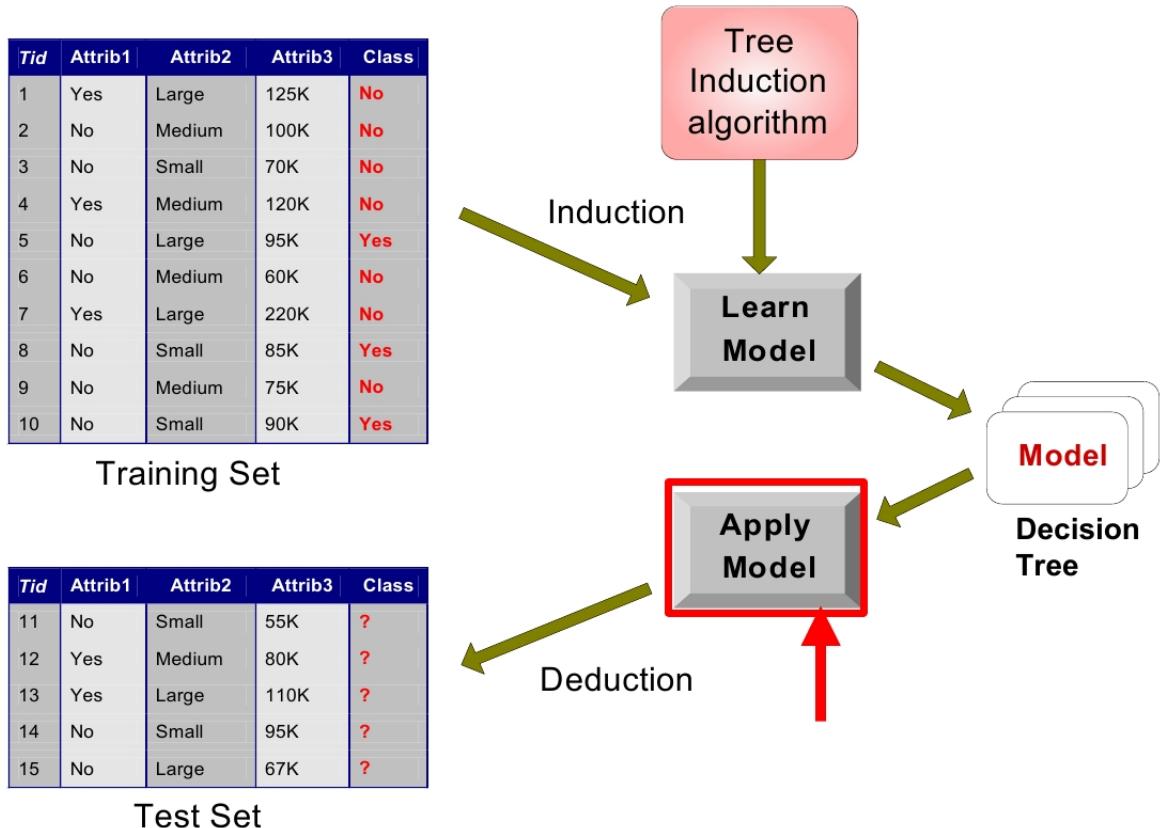


FIGURE 8.3 – Illustration de la méthode des arbres de décision [Tan et al., 2005]

8.3.1 Comment construire l'arbre de décision ? [Ph Preux 2011]

La construction d'un arbre de décision optimal est un problème NP-complet. En théorie, il existe un nombre exponentiel d'arbres de décision sur un exemple de données. Repérer l'arbre de décision optimal est un problème difficile et insurmontable en pratique à cause du nombre exponentiel de combinaisons à explorer. La solution est d'adopter un algorithme approché générant un arbre de décision qui soit le plus pertinent possible sans garantir pour autant l'optimalité. Les algorithmes existants adoptent donc la stratégie gloutonne. Parmi les algorithmes, on cite CART en 1980 (Breiman et al. 1984), l'algorithme ID3 de Quinlan en 1986, et C4.5 (puis C5) de Quinlan en 1993. D'un point de vue expérimental, ces algorithmes sont très performants et permettent de construire rapidement des arbres de décision qui prédisent avec une grande fiabilité la classe de nouvelles données.

ID3 ne prend en compte que les attributs nominaux, alors que C4.5 prend en charge des attributs numériques.

ID3 fonctionne récursivement : il détermine un attribut à placer en racine de l'arbre. Cette racine prend autant de branches que cet attribut prend de valeurs. A chaque branche est associé un ensemble d'exemples dont l'attribut prend la valeur qui étiquette cette branche ; on accroche alors au bout de cette branche l'arbre de décision construit sur ce sous-ensemble des exemples et en considérant tous les attributs excepté celui qui vient d'être mis à la racine. Par cette procédure, l'ensemble des exemples ainsi que l'ensemble des attributs diminuent petit à petit au long de la descente dans l'arbre.

L'arbre de décision est construit récursivement en partitionnant les données de la base d'apprentissage en sous-ensembles. Soient S l'ensemble des données de la base d'apprentissage (training set), A l'ensemble des attributs, et y l'attribut classe. L'algorithme est donné comme suit [Maimon and Rokach, 2005] :

TreeGrowing ($S, A, y, SplitCriterion, StoppingCriterion$)

where:

S - Training Set

A - Input Feature Set

y - Target Feature

$SplitCriterion$ - the method for evaluating a certain split

$StoppingCriterion$ - the criteria to stop the growing process

Create a new tree T with a single root node.

IF $StoppingCriterion(S)$ THEN

 Mark T as a leaf with the most
 common value of y in S as a label.

ELSE

 Find attribute a that obtains the best $SplitCriterion(a_i, S)$.

 Label t with a

 FOR each outcome v_i of a :

 Set $Subtree_i = \text{TreeGrowing } (\sigma_{a=v_i} S, A, y)$.

 Connect the root node of t_T to $Subtree_i$ with
 an edge that is labelled as v_i

 END FOR

END IF RETURN TreePruning (S, T, y)

TreePruning (S, T, y) Where: S - Training Set

y - Target Feature T - The tree to be pruned

DO

 Select a node t in T such that pruning it
 maximally improve some evaluation criteria

 IF $t \neq \emptyset$ THEN $T = \text{pruned}(T, t)$

UNTIL $t = \emptyset$

RETURN T

Top-Down Algorithmic Framework for Decision Trees Induction.

FIGURE 8.4 – Algorithme de construction de l'arbre de décision
[Maimon and Rokach, 2005]

L'algorithme pose plusieurs questions :

1. Comment sélectionner l'attribut (*SplitCriterion* dans l'algorithme Figure 8.4) ?
2. Comment partitionner les valeurs d'un attribut ?
3. Comment arrêter le processus de partitionnement des données (*StoppingCriterion* dans l'algorithme Figure 8.4) ? La stratégie évidente est d'arrêter une fois le noeud est nul ou que l'ensemble des données couverts par le noeud appartient à la même classe. Nous considérerons aussi la stratégie qui consiste à ne pas continuer le partitionnement pour éviter un coût tout en garantissant un certain niveau de performance.

8.3.2 Stratégies de partitionnement de l'attribut

Suivant le type de l'attribut, différentes stratégies peuvent adoptées :

Attribut binaires Il existe deux sorties possibles, une pour chacune des deux valeurs booléennes.

Attribut nominal Ce type d'attribut a un nombre fini de valeurs sans relation d'ordre. Les deux procédés sont illustrés dans la Figure 8.5. La stratégie qui consiste à produire au plus deux sorties est adopté par la méthode CART.

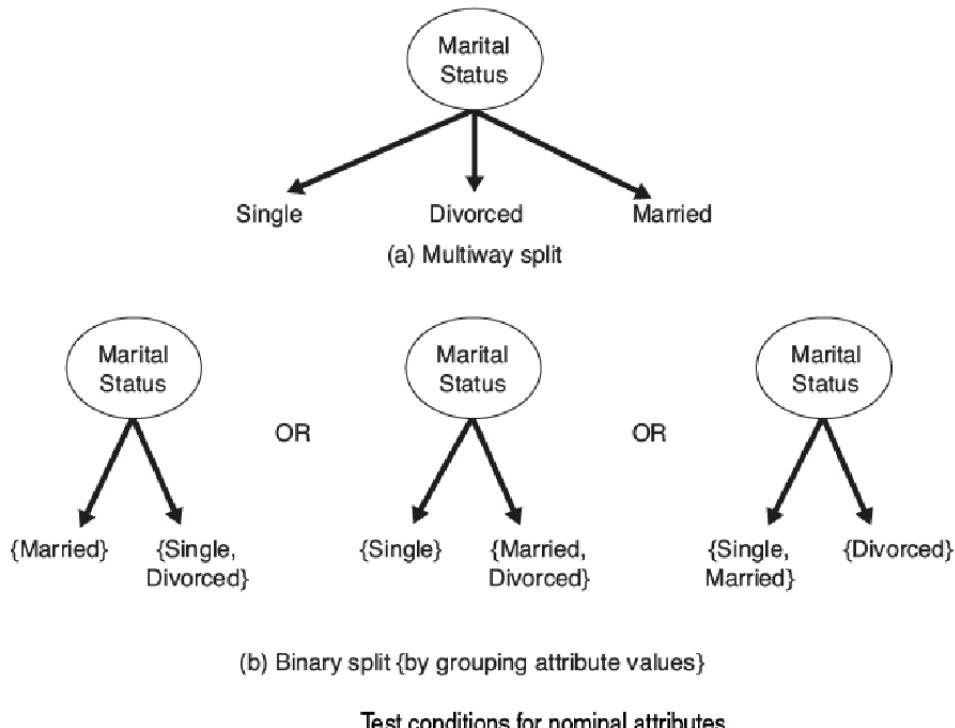


FIGURE 8.5 – Partitionnement d'un noeud de type domaine fini [Tan et al., 2005]

Attribut ordinal Ce type d'attribut a un nombre fini de valeurs avec une relation d'ordre. Le partitionnement peut adopter les mêmes stratégies que celles du type nominal, en exploitant la relation d'ordre entre les valeurs.

Attribut continu Le partitionnement des attributs continus se fait d'une façon binaire $A < v$ ou $A \geq v$ ou par partitionnement sur plusieurs intervalles $v_i \leq A < v_{i+1}$ for $i = 1..k$. Ces procédés sont illustrés dans la Figure 8.9.

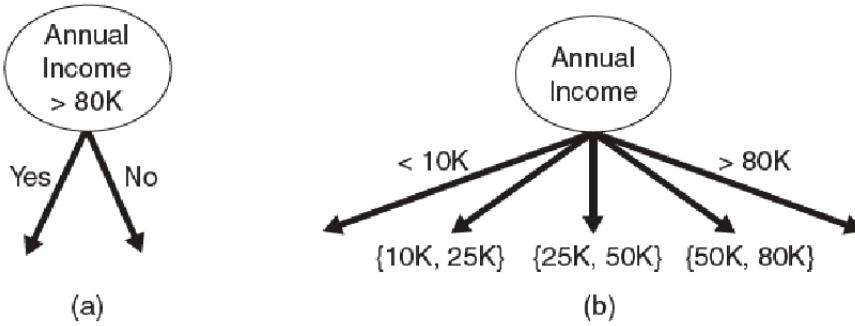


FIGURE 8.6 – Partitionnement d'un noeud de type continu [Tan et al., 2005]

8.3.3 Problème du choix de l'attribut [Preux, 2006, Maimon and Rokach, 2005]

Ayant l'idée de l'algorithme, il reste à résoudre une question centrale : quel attribut placer en racine ? Une fois cette question résolue, on itérera le raisonnement pour les sous-arbres.

L'intuition de la réponse à cette question est que l'on tente de réduire l'hétérogénéité à chaque noeud : les données qui atteignent un certain noeud de l'arbre de décision doivent être plus homogènes que les données atteignant un noeud ancêtre. Pour cela, on a besoin de pouvoir l'homogénéité d'un ensemble de données. En physique, on mesure l'homogénéité par l'entropie.

Plusieurs critères de choix peuvent être adoptés [Maimon and Rokach, 2005] :

Critère de minimisation de l'impureté (Impurity-based Criteria) Soit x une variable aléatoire ayant k valeurs discrètes, ayant les probabilités $P = (p_1, \dots, p_k)$. Une mesure d'impureté est une fonction $\phi : [0, 1]^k \rightarrow \mathbb{R}$ qui satisfait les conditions suivantes :

- $\phi(P) \geq 0$
- $\phi(P)$ est minimum si $\exists i, p_i = 1$.
- $\phi(P)$ est maximum si $\forall i, p_i = 1/k$.
- $\phi(P)$ est symétrique relativement aux composantes de P (fonction invariante par permutation de ses variables).
- $\phi(P)$ est différentiable partout dans son domaine. (On appelle différentielle d'ordre 1 d'une fonction en un point a (ou dérivée de cette fonction au point a) la partie linéaire de l'accroissement de cette fonction entre a et $a + h$ lorsque h tend vers 0. Une fonction possédant une différentielle est appelée une fonction différentiable.)

Etant donné un exemple de données S , le vecteur des probabilités de l'attribut cible y est défini comme suit :

$$P_y(S) = \left(\frac{|\sigma_{y=c_1} S|}{|S|}, \dots, \frac{|\sigma_{y=c_{\text{dom}(y)}} S|}{|S|} \right) \quad (8.2)$$

où $\sigma_{a_i=v_{i,j}} S$ renvoie les données ayant comme valeur d'attribut $a_i = v_{i,j}$, $v_{i,j} \in \text{dom}(a_i)$.

La qualité d'un split (partitionnement) est définie comme étant la maximisation de la réduction dans l'impureté de l'attribut cible après le partitionnement S relativement à une valeur $v_{i,j} \in \text{dom}(a_i)$:

$$\Delta\Phi(a_i, S) = \phi(P_y(S)) - \sum_{j=1}^{|\text{dom}(a_i)|} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} \cdot \phi(P_y(\sigma_{a_i=v_{i,j}} S)) \quad (8.3)$$

Gain en information (information gain) C'est un critère d'impureté (proposée par [Quinlan, 1987]) utilisant la mesure d'entropie.

L'entropie de Shannon [E.Shannon, 1948], due à Claude Shannon, est une fonction mathématique qui, intuitivement, correspond à la quantité d'information contenue ou délivrée par une source d'information. Cette source peut être un texte écrit dans une langue donnée, un signal électrique ou encore un fichier informatique quelconque (collection d'octets). Du point de vue d'un récepteur, plus la source émet d'informations différentes, plus l'entropie (ou incertitude sur ce que la source émet) est grande. Ainsi, si une source est réputée envoyer toujours le même symbole, par exemple la lettre 'a', alors son entropie est nulle, c'est-à-dire minimale. En effet, un récepteur qui connaît seulement les statistiques de transmission de la source est assuré que le prochain symbole sera un 'a'. Par contre, si la source est réputée envoyer un 'a' la moitié du temps et un 'b' l'autre moitié, le récepteur est incertain de la prochaine lettre à recevoir. L'entropie de la source dans ce cas est donc non nulle (positive) et représente quantitativement l'incertitude qui règne sur l'information émanant de la source. L'entropie indique alors la quantité d'information nécessaire pour que le récepteur puisse déterminer sans ambiguïté ce que la source a transmis. Plus le récepteur reçoit d'information sur le message transmis, plus l'entropie (incertitude) vis-à-vis de ce message décroît. En particulier, plus la source est redondante, moins elle contient d'information. En l'absence de contraintes particulières, l'entropie est maximale pour une source dont tous les symboles sont équiprobables. [Wikipedia https://fr.wikipedia.org/wiki/Entropie_de_Shannon]

$$\begin{aligned} \text{InformationGain}(a_i, S) &= \\ \text{Entropy}(y, S) - \sum_{v_{i,j} \in \text{dom}(a_i)} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} \cdot \text{Entropy}(y, \sigma_{a_i=v_{i,j}} S) \end{aligned}$$

où

$$\text{Entropy}(y, S) = \sum_{c_i \in \text{dom}(y)} -\frac{|\sigma_{y=c_i} S|}{|S|} \cdot \log_2 \left(\frac{|\sigma_{y=c_i} S|}{|S|} \right) \quad (8.4)$$

L'entropie mesure l'hétérogénéité d'une population du point de vue de la classe de ses membres. Elle se mesure en bits pour coder l'information. C'est en fait une mesure de la quantité d'information qu'il y a dans S du point de vue de la classe y . Intuitivement, avoir des sous-ensembles dont l'entropie est minimale est intéressant : cela signifie que l'attribut placé à la racine

discrimine les exemples en fonction de leur classe. Il est naturel de sommer ces entropies en les pondérant en fonction de la proportion d'exemples dans chacun des sous-ensembles.

Dans le cas où l'on dispose d'un nombre N de symboles de la forme $N = 2^n$, avec n entier, et où les N symboles sont équiprobables, il suffit de n questions, en procédant par dichotomie, pour déterminer le symbole envoyé par la source. Dans ce cas, la quantité d'information contenue par le symbole est exactement $n = \log_2(N)$. [Wikipedia https://fr.wikipedia.org/wiki/Entropie_de_Shannon]

Le gain basé sur l'entropie sera notre mesure par défaut : $Gain(a_i, S) = InformationGain(a_i, S)$.

Index de Gini Il mesure la divergence entre les probabilités des valeurs de la cible.

$$Gini(y, S) = 1 - \sum_{c_i \in dom(y)} \left(\frac{|\sigma_{y=c_i} S|}{|S|} \right)^2$$

En utilisant cette mesure, la sélection de l'attribut se fait en prenant en maximisant la mesure suivante :

$$GiniGain(a_i, S) = Gini(y, S) - \sum_{v_{i,j} \in dom(a_i)} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} \cdot Gini(y, \sigma_{a_i=v_{i,j}} S) \quad (8.5)$$

Rapport de gain En présence d'attribut numérique ou d'attribut d'arité élevée, ceux ci sont automatiquement favorisés pour être sélectionnés comme test dans les noeuds. Pour contrecarrer cet effet, C4.5 utilise le rapport de gain au lieu du gain d'information pour déterminer l'attribut à utiliser dans un noeud.

$$GainRatio(a_i, S) = \frac{InformationGain(a_i, S)}{Entropy(a_i, S)}.$$

Critère des attributs multiples Plusieurs attributs sont sélectionnés au niveau du noeud.

De nombreux autres indices existent dans la littérature, nous recommandons la référence [Maimon and Rokach, 2005].

8.3.4 Exemple de déroulement

Soit l'ensemble des données :

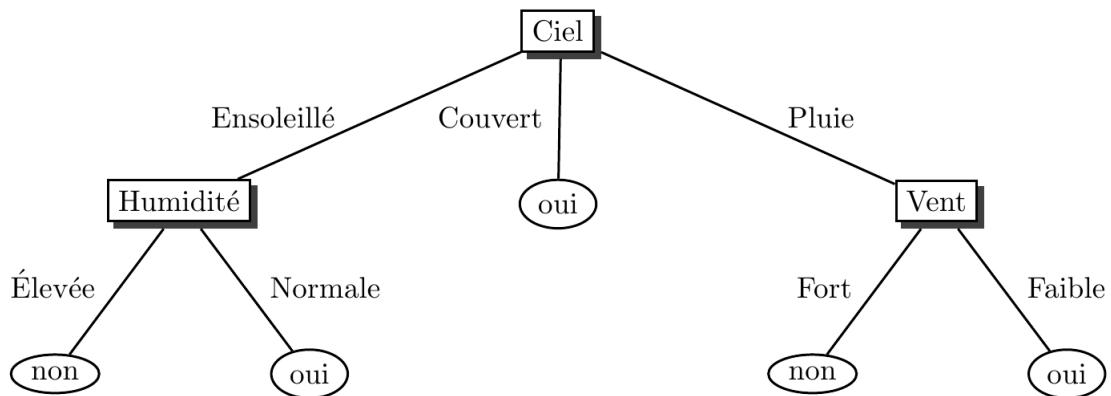
Jeu de données « jouer au tennis ? »

Jour	Ciel	Température	Humidité	Vent	Jouer au tennis ?
1	Ensoleillé	Chaud	Élevée	Faible	Non
2	Ensoleillé	Chaud	Élevée	Fort	Non
3	Couvert	Chaud	Élevée	Faible	Oui
4	Pluie	Tiède	Élevée	Faible	Oui
5	Pluie	Fraîche	Normale	Faible	Oui
6	Pluie	Fraîche	Normale	Fort	Non
7	Couvert	Fraîche	Normale	Fort	Oui
8	Ensoleillé	Tiède	Élevée	Faible	Non
9	Ensoleillé	Fraîche	Normale	Faible	Oui
10	Pluie	Tiède	Normale	Faible	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui
12	Couvert	Tiède	Élevée	Fort	Oui
13	Couvert	Chaud	Normale	Faible	Oui
14	Pluie	Tiède	Élevée	Fort	Non

Attribut	Gain
Ciel	0,246
Humidité	0,151
Vent	0,048
Température	0,029

FIGURE 8.7 – Exemple de données et leurs gains [Preux, 2006]

L'attribut sélectionné est "Ciel". L'attribut "Ciel" peut prendre trois valeurs. Pour Ensoleillé", l'algorithme est appelé avec 5 données {1, 2, 8, 9, 11} ; le gains des trois attributs restants sont :



Arbre de décision obtenu pour l'exemple du texte « jouer au tennis ? ».

FIGURE 8.8 – Gain après sélection de "Ciel" sur la valeur "Ensoleillé" [Preux, 2006]

Nous continuons ainsi jusqu'au feuille.

On peut utiliser cet arbre de décision pour classer de nouvelles données, comme illustré ci-dessous.

Algorithme 2 Classification d'une donnée dans un arbre de décision

Nécessite: 2 paramètres : arbre de décision AD, exemple x

```

// on utilise une variable nc (nœud courant) pour parcourir l'arbre
nc ← racine (AD)
tant-que nc ≠ feuille faire
  en fonction de l'attribut testé dans nc et de sa valeur dans  $x$ , suivre l'une
  des branches de nc. Le nœud atteint devient nc
fin tant-que
retourner étiquette(nc)
  
```

Par exemple, on peut prédire la classe pour les données suivantes :

- (Ensoleillé, Fraîche, Élevée, Fort) est classée comme « non » ;
- (Ensoleillé, Fraîche, Normale, Fort) est classée comme « oui » ;
- (Pluie, Chaude, Normale, Faible) est classée comme « oui » ;
- (Pluie, Fraîche, Élevée, Fort) est classée comme « oui » .

FIGURE 8.9 – Algorithme et exemples de classification de nouvelles données [Preux, 2006]

8.3.5 Elagage [Preux, 2006]

L'élagage consiste à simplifier un arbre de décision en coupant des branches :

- simplifier l'arbre
- diminuer le sur-apprentissage (augmentation de la capacité de généralisation)

Deux types d'élagage sont effectués dans C4.5 :

- remplacement d'un sous-arbre : consiste à remplacer un sous-arbre par une feuille ;
- promotion d'un sous-arbre : consiste à rassembler deux noeuds dans un seul noeud.

Exemple : on suppose qu'un noeud possède trois fils ; le premier couvre 6 exemples, dont 2 sont mal classés ; le deuxième couvre 2 exemples, 1 étant mal classé ; le dernier couvre 6 exemples dont, à nouveau, 2 sont mal classés. Les taux d'erreur estimés de ces trois fils sont respectivement 0,47, 0,72 et 0,47 (avec un taux de confiance de 75 %). La combinaison des trois donne : $\frac{6}{14} \times 0,47 + \frac{2}{14} \times 0,72 + \frac{6}{14} \times 0,47 = 0,504$. Le taux d'erreur du nœud lui-même est donc calculé avec les données : 14 exemples, dont 5 mal classés, ce qui fournit un taux d'erreur estimé de 0,46. Ce taux d'erreur étant plus petit, C4.5 remplace le nœud par une feuille étiquetée avec la valeur la plus présente parmi ces 14 exemples.

FIGURE 8.10 – Exemple de déroulement d'un élagage de remplacement d'un sous-arbre [Preux, 2006]

8.4 Forêts d'arbres décisionnels (Random forest)

8.5 Régression logistique

8.6 Validation des méthodes supervisées : illustration sur les arbres de décision [Preux, 2006]

Une fois un arbre de décision construit, il est essentiel de le valider en estimant la probabilité que la classe prédictive pour une donnée quelconque soit correcte. Dépendant de l'ensemble de données qui est utilisé pour la mesurer, cette quantité est donc une variable aléatoire dont il faut estimer la valeur.

En fait, cette section ne concerne pas uniquement les arbres de décision : à quelque détail près, elle peut s'appliquer à tous les algorithmes de classification qui seront vus par la suite.

Définition 33 *L'erreur de classification E d'un classeur est la probabilité que ce classeur ne prédise pas correctement la classe d'une donnée de l'espace de donnée. Le taux de succès est égal à $1 - E$.*

L'erreur apparente E_{app} est mesurée avec les exemples utilisés pour la construction du classeur : c'est la proportion d'exemples dont la classe est mal prédite par le classeur.

E_{app} n'est pas un bon estimateur de l'erreur qui serait commise face à de nouvelles données ; en effet, l'enjeu est bien là : à partir des exemples avec lesquels l'arbre de décision a été construit, l'apprentissage doit pouvoir être généralisé à de nouvelles données. C'est là l'objectif des algorithmes d'apprentissage : un algorithme ne peut être qualifié d'algorithme d'apprentissage que s'il est capable de généraliser ce qu'il a appris.

L'estimation de la qualité de l'arbre de décision construit en tant que classeur de nouvelles données est donc un point crucial. On distingue :

jeu d'exemples d'apprentissage X_{app}

jeu d'exemples de test X_{test} qui permet d'estimer les erreurs de classification : pour ces exemples, on connaît leur classe. On les classe avec l'arbre de décision construit avec X_{app} puis on regarde s'ils sont classés correctement.

Etant donné un classeur de données de cible binaire.

Définition 34

- VN : le nombre de vrais négatifs ($(\text{classe-test}=0)$ et $(\text{classe-prédite}=0)$)
- FN : le nombre de faux positifs ($(\text{classe-test}=0)$ et $(\text{classe-prédite}=1)$)
- FP : le nombre de faux négatifs ($(\text{classe-test}=1)$ et $(\text{classe-prédite}=0)$)
- VP : le nombre de vrais positifs ($(\text{classe-test}=1)$ et $(\text{classe-prédite}=1)$)

On peut définir aussi deux mesures statistiques, la précision et le rappel :

- précision pour les positifs : $\frac{VP}{VP+FP}$; précision pour les négatifs = $\frac{VN}{VN+FN}$;
- rappel pour les positifs : $\frac{VP}{VP+FN}$; rappel pour les négatifs = $\frac{VN}{VN+FP}$.

Validation croisée Soit un partitionnement de l'ensemble des données X en trois sous-ensembles disjoints A, B et C . On construit l'arbre de décision $AD_{B \cup C}$ et on mesure l'erreur E_A . Nous faisons de même avec $A \cup C$ et $A \cup B$. Le taux d'erreur E est alors estimé par la moyenne de ces trois mesures : $E = \frac{E_A+E_B+E_C}{3}$. Habituellement, on prend $n = 10$.

Technique du leave-one-out Elle consiste à effectuer une validation croisée à $n = N$ en laissant à chaque fois un seul exemple de côté. L'erreur est estimée par la moyenne des N erreurs mesurées.

Technique de bootstrap (bagging) Le jeu d'apprentissage est constitué en effectuant N tirages avec remise parmi l'ensemble des exemples. Cela entraîne que certains exemples du jeu d'apprentissage seront vraisemblablement sélectionnés plusieurs fois, et donc que d'autres ne le seront jamais. En fait, la probabilité qu'un certain exemple ne soit jamais tiré est simplement : $(1 - 1/N)^N$. La limite quand $N \rightarrow \infty$ de cette probabilité est $e^{-1} = 0.368$. Les exemples qui n'ont pas été sélectionnés constituent le jeu de test. Le jeu d'apprentissage contient 63.2% des exemples du jeu d'exemples initial. L'erreur est calculée en combinant l'erreur d'apprentissage E_{app} et l'erreur de test E_{test} par la formule suivante : $E = 0.632 \times E_{test} + 0.368 \times E_{app}$. Ensuite cette procédure est itérée plusieurs fois et une erreur moyenne est calculée.

Confiance dans l'estimation de l'erreur **Donnons nous un seuil de confiance c et déterminons la probabilité avec laquelle E est dans un certain intervalle de valeurs centré sur E_{test} .** On présente un exemple au classeur. On compte 0 si l'exemple est bien classé, 1 sinon. Si on a N exemples, on obtient ainsi une estimation de la valeur de E_{test} . Nous sommes en présence d'une loi binomiale. Aussi, on sait que cette estimation de E_{test} tend vers E . On sait aussi que sa variance tend vers $E(1 - E)/N$. Donc, $\frac{E_{test} - E}{\sqrt{\frac{E(1-E)}{N}}}$ est une variable aléatoire centrée réduite.

Rappelons que la probabilité qu'un variable aléatoire v distribuée normalement, centrée et réduite prenne une valeur dans un intervalle $x \pm dx$ est donnée par

$$Pr[v = x] = \frac{1}{2\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

La probabilité que la valeur de v soit comprise dans un intervalle $[-z, +z]$ est donnée par :

$$Pr[-z \leq v \leq +z] = \frac{1}{2\sqrt{\pi}} \int_{-z}^{+z} e^{-\frac{x^2}{2}} dx$$

Quel est l'intervalle de valeurs $[-z, +z]$ autout de E_{test} dans lequel nous avons la probabilité c que E s'y trouve : C'est z tel que :

$$Pr[-z \leq \frac{E_{test} - E}{\sqrt{\frac{E(1-E)}{N}}} \leq +z] = c$$

dans laquelle E_{test} est connu, c est fixé (donc z s'en déduit) et E est cherché. Donc, on cherche bien l'intervalle $E \in [E_{inf}, E_{sup}]$ tel que la probabilité que E appartienne à cet intervalle soit c . On trouve z dans une table de distribution normale, centrée, réduite. Ce type de table, on sait que $Pr[v \leq -z] = Pr[v \geq z]$. Pour une distribution normale, on sait que $Pr[v \geq -z] = Pr[v \geq z]$. **La confiance c dans l'intervalle est $c = Pr[-z \leq v \leq +z] = 1 - 2 \times Pr[v \geq z]$.**

Donc, c étant fixé, on détermine z tel que

$$Pr[v \geq z] = (1 - c)/2.$$

$$\left\{ \begin{array}{l} E_{inf} = \frac{E_{\text{test}} + \frac{z^2}{2N} - z \sqrt{\frac{E_{\text{test}}}{N} - \frac{E_{\text{test}}^2}{N^2} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \\ E_{sup} = \frac{E_{\text{test}} + \frac{z^2}{2N} + z \sqrt{\frac{E_{\text{test}}}{N} - \frac{E_{\text{test}}^2}{N^2} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \end{array} \right.$$

FIGURE 8.11 – [Preux, 2006]

$Pr[v \geq z]$	0,001	0,005	0,01	0,05	0,1	0,2	0,4
z	3,09	2,58	2,33	1,65	1,28	0,84	0,25

Par exemple, si $E_{\text{test}} = 0,25$, $N = 1000$ et $c = 0,8$, on obtient un intervalle de confiance :

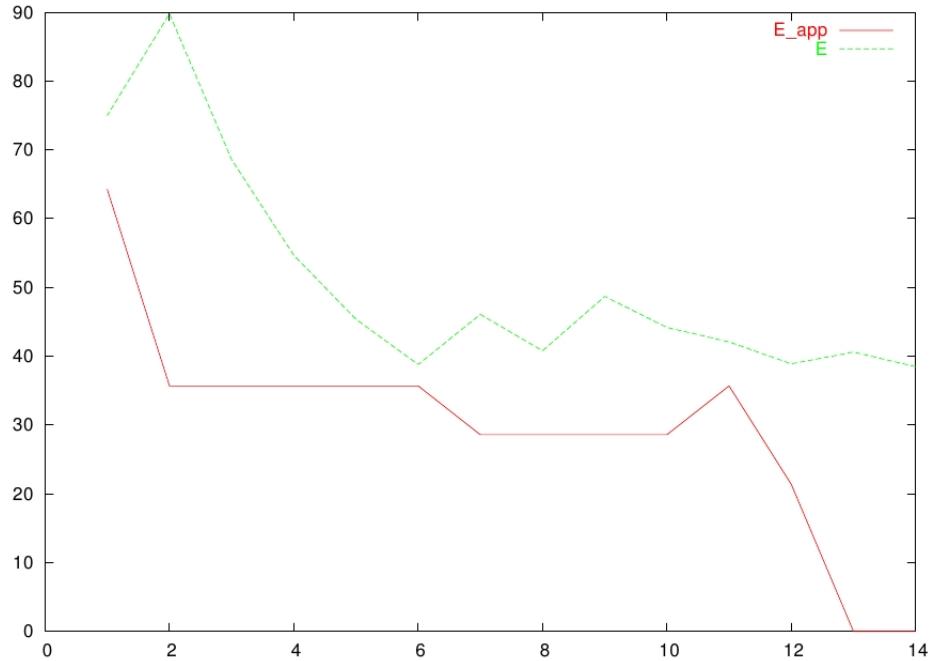
$$E \in \frac{0,25 + \frac{1,28^2}{2000} \pm 1,28 \sqrt{\frac{0,25}{1000} - \frac{0,25^2}{1000^2} + \frac{1,28^2}{4 \times 1000^2}}}{1 + \frac{1,28^2}{1000}} = [0,233, 0,268]$$

De même, si $N = 100$, on a $E \in [0,20, 0,31]$. On constate, on s'y attendait, que plus N est grand, plus l'intervalle est resserré.

FIGURE 8.12 – Extrait de la table de probabilité et exemple de calcul d'un intervalle de confiance [Preux, 2006]

8.6.1 Surapprentissage

La Figure suivante trace le comportement de E et de E_{app} en fonction du nombre d'exemples.



Variation de l'erreur mesurée sur l'ensemble de tous les exemples et de l'erreur E (estimée) en fonction du nombre d'exemples utilisés pour construire l'arbre de décision. Les deux erreurs diminuent. Au bout d'un certain temps, alors que E_{app} continue de diminuer, E stagne. E est estimée ici avec une confiance $c = 0.75$. On utilise le jeu de données « jouer au tennis ? ».

FIGURE 8.13 – Evolution de E et de E_{app} en fonction du nombre d'exemples [Preux, 2006]

Il est très important de comprendre ce que ce schéma illustre. Plus le nombre d'exemples utilisés pour construire le modèle augmente, plus l'erreur sur ce jeu diminue et tend vers zéro. **Cependant ce qui compte vraiment, c'est l'erreur de généralisation. Justement, quand le nombre d'exemples utilisés augmente, l'erreur en généralisation commence à diminuer puis elle augmente : c'est précisément là où elle est minimale que l'on construit le meilleur modèle, celui qui fait une erreur minimale. C'est ce modèle qui est capable de produire la meilleure généralisation de l'apprentissage. Au delà de ce modèle optimal, quand l'apprentissage se poursuit, le modèle se complique, la probabilité d'erreur augmente, et le modèle produit du sur-apprentissage. Le modèle alors construit colle de plus en plus près aux exemples et sa capacité à prédire correctement la classe d'autres données diminue ; le modèle manque de recul.**

8.7 Classeur bayésien

"EXT-BAYES-NAIF.pdf".

8.8 Classification par réseaux de neurones (Perceptron)

"[EXT-SVM-PERCEPTRON.pdf](#)".

8.9 Classification par SVM

["EXT-SVM-PERCEPTRON.pdf".](#)
["EXT-CLUSTERING.pdf".](#)

8.10 *k*-means

["EXT-CLUSTERING.pdf".](#)

Chapitre 9

Travaux pratiques

9.1 Environnement SPMF

1. Télécharger la librairie SPMF à partir de son site officiel
<http://www.philippe-fournier-viger.com/spmf>
2. Créer un projet sous Eclipse
3. Intégrer la librairie SPMF en suivant les instructions données dans "how_to_install.txt". Vous trouverez dans le fichier [SPMF_A Java Open-Source Data Mining Library.html](#) les algorithmes de fouille implémentés.
4. Afficher le programme java "[MainTestApriori_saveToMemory.java](#)" qui fait appel à la méthode Apriori.
5. Lancer la fouille sur le fichier exemple "contextPasquier99.txt". Commenter les résultats donnés.
6. Saisir l'exemple de fouille ensembliste donné en cours, et lancer sa fouille. Commenter les résultats donnés.

9.2 Fouille itemset

Une entreprise commerciale a décidé de lancer une opération de promotion. Le gérant vous demande d'utiliser la fouille ensembliste pour trouver des motifs intéressants pour ses futures promotions. Il va donc réutiliser le bilan des achats de l'année dernière à la même date :

Achats	Produit 1	Produit 2	Produit 3	Produit 4	Produit 5
Client 1	X			X	X
Client 2	X	X			X
Client 3					X
Client 4			X	X	X
Client 5	X	X	X	X	X
Client 6	X				X
Client 7	X			X	X
Client 8		X	X		

1. Extraire les motifs fréquents avec un seuil de, 20%, **30%**, 40%, 50%, 60%, puis 100%
2. Extraire les motifs clos

9.3 Fouille de séquences

Soit la base de séquences donnée dans la Figure 9.1.

1. Utiliser la librairie SPMF pour trouver les motifs séquentiels fréquents avec $\text{minsup}=4$. (N.B. *The input file format is defined as follows. It is a text file where each line represents a sequence from a sequence database. Each item from a sequence is a positive integer and items from the same itemset within a sequence are separated by single space. Note that it is assumed that items within a same itemset are sorted according to a total order and that no item can appear twice in the same itemset. The value "-1" indicates the end of an itemset. The value "-2" indicates the end of a sequence (it appears at the end of each line).*)
2. Lesquels des motifs trouvés sont clos ?
3. Lesquels des motifs trouvés sont maximaux ?

SID	Sequence
1	$\langle \{a, b\}, \{c\}, \{f, g\}, \{g\}, \{e\} \rangle$
2	$\langle \{a, d\}, \{c\}, \{b\}, \{a, b, e, f\} \rangle$
3	$\langle \{a\}, \{b\}, \{f, g\}, \{e\} \rangle$
4	$\langle \{b\}, \{f, g\} \rangle$

FIGURE 9.1 – Exemple d'une base de séquences

9.4 Familiarisation Python

Le but de cette section est de se familiariser avec le langage Python. En premier, il est nécessaire d'installer Python (3.7 actuellement), et Anaconda (environnement prédisposant déjà de la majeure partie des packages dont on va se servir). Nous proposons de suivre les manipulations suivantes pour s'initier à la programmation sous Python. Certaines parties consistent en une série de codes et d'exécutions, dont l'intérêt est commenté au niveau des lignes du code donné.

Indentation Le langage Python exige une indentation stricte pour déclarer les blocs du programme. Exemple :

```
for i in [1, 2, 3, 4, 5]:
    print i                  # first line in "for i" block
    for j in [1, 2, 3, 4, 5]:
        print j              # first line in "for j" block
        print i + j          # last line in "for j" block
    print i                  # last line in "for i" block
print "done looping"
```

Fonctions

```
def double(x):
    """this is where you put an optional docstring
    that explains what the function does.
    for example, this function multiplies its input by 2"""
    return x * 2
```

Le langage Python est du premier ordre ; on peut introduire des fonctions en argument. Exemple :

```
def apply_to_one(f):
    """calls the function f with 1 as its argument"""
    return f(1)
```

et par la suite, on peut écrire :

```
my_double = double      # refers to the previously defined function
x = apply_to_one(my_double)  # equals 2
```

Chaines de caractères

Les chaines sont avec cotes ou en double-cotes :

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
```

```
tab_string = "\t"      # represents the tab character
len(tab_string)       # is 1
```

```
multi_line_string = """
    This is the first line.
    and this is the second line
    and this is the third line"""
```

Exception

```
try:
    print 0 / 0
except ZeroDivisionError:
    print "cannot divide by zero"
```

List

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [ integer_list, heterogeneous_list, [] ]
list_length = len(integer_list)      # equals 3
list_sum   = sum(integer_list)      # equals 6

x = range(10)  # is the list [0, 1, ..., 9]
zero = x[0]    # equals 0, lists are 0-indexed
one = x[1]     # equals 1
nine = x[-1]   # equals 9, 'Pythonic' for last element
eight = x[-2]  # equals 8, 'Pythonic' for next-to-last element
x[0] = -1      # now x is [-1, 1, 2, 3, ..., 9]

first_three  = x[:3]                # [-1, 1, 2]
three_to_end = x[3:]               # [3, 4, ..., 9]
one_to_four  = x[1:5]              # [1, 2, 3, 4]
last_three   = x[-3:]              # [7, 8, 9]
without_first_and_last = x[1:-1]    # [1, 2, ..., 8]
copy_of_x    = x[:]                 # [-1, 1, 2, ..., 9]

1 in [1, 2, 3]  # True
0 in [1, 2, 3]  # False

x = [1, 2, 3]
x.extend([4, 5, 6])  # x is now [1,2,3,4,5,6]

x = [1, 2, 3]
y = x + [4, 5, 6]  # y is [1, 2, 3, 4, 5, 6]; x is unchanged

x = [1, 2, 3]
x.append(0)  # x is now [1, 2, 3, 0]

y = x[-1]    # equals 0
z = len(x)   # equals 4

x, y = [1, 2]  # now x is 1, y is 2
_, y = [1, 2]  # now y == 2, didn't care about the first element
```

Tuples

```
my_list = [1, 2]
my_tuple = (1, 2)
other_tuple = 3, 4
```

```

my_list[1] = 3      # my_list is now [1, 3]

try:
    my_tuple[1] = 3
except TypeError:
    print "cannot modify a tuple"

def sum_and_product(x, y):
    return (x + y),(x * y)

sp = sum_and_product(2, 3)  # equals (5, 6)
s, p = sum_and_product(5, 10) # s is 15, p is 50

x, y = 1, 2      # now x is 1, y is 2
x, y = y, x      # Pythonic way to swap variables; now x is 2, y is 1

```

Dictionnaires

```

empty_dict = {}                  # Pythonic
empty_dict2 = dict()             # less Pythonic
grades = { "Joel" : 80, "Tim" : 95 }  # dictionary literal

joels_grade = grades["Joel"]      # equals 80

try:
    kates_grade = grades["Kate"]
except KeyError:
    print "no grade for Kate!"

joel_has_grade = "Joel" in grades  # True
kate_has_grade = "Kate" in grades   # False

joels_grade = grades.get("Joel", 0)  # equals 80
kates_grade = grades.get("Kate", 0)  # equals 0
no_ones_grade = grades.get("No One") # default default is None

grades["Tim"] = 99                # replaces the old value
grades["Kate"] = 100               # adds a third entry
num_students = len(grades)        # equals 3

tweet = {
    "user" : "joelgrus",
    "text" : "Data Science is Awesome",
    "retweet_count" : 100,
    "hashtags" : ["#data", "#science", "#datascience", "#awesome", "#yolo"]
}

tweet_keys = tweet.keys()      # list of keys
tweet_values = tweet.values()  # list of values

```

```

tweet_items = tweet.items()    # list of (key, value) tuples

"user" in tweet_keys      # True, but uses a slow list in
"user" in tweet          # more Pythonic, uses faster dict in
"joelgrus" in tweet_values # True

dd_list = defaultdict(list)      # list() produces an empty list
dd_list[2].append(1)            # now dd_list contains {2: [1]}
dd_dict = defaultdict(dict)     # dict() produces an empty dict
dd_dict["Joel"]["City"] = "Seattle" # { "Joel" : { "City" : Seattle" } }
dd_pair = defaultdict(lambda: [0, 0])
dd_pair[2][1] = 1              # now dd_pair contains {2: [0,1]}

```

Ensembles

```

s = set()
s.add(1)      # s is now { 1 }
s.add(2)      # s is now { 1, 2 }
s.add(2)      # s is still { 1, 2 }
x = len(s)    # equals 2
y = 2 in s   # equals True
z = 3 in s   # equals False

stopwords_list = ["a", "an", "at"] + hundreds_of_other_words + ["yet", "you"]
"zip" in stopwords_list    # False, but have to check every element
stopwords_set = set(stopwords_list)
"zip" in stopwords_set    # very fast to check

item_list = [1, 2, 3, 1, 2, 3]
num_items = len(item_list)      # 6
item_set = set(item_list)       # {1, 2, 3}
num_distinct_items = len(item_set) # 3
distinct_item_list = list(item_set) # [1, 2, 3]

```

Instruction de flux

```

if 1 > 2:
    message = "if only 1 were greater than two..."
elif 1 > 3:
    message = "elif stands for 'else if'"
else:
    message = "when all else fails use else (if you want to)"

parity = "even" if x % 2 == 0 else "odd"

x = 0
while x < 10:
    print x, "is less than 10"
    x += 1

for x in range(10):

```

```

print x, "is less than 10"

for x in range(10):
    if x == 3:
        continue # go immediately to the next iteration
    if x == 5:
        break   # quit the loop entirely
    print x

```

Logique

```

one_is_less_than_two = 1 < 2      # equals True
true_equals_false = True == False  # equals False

x = None
print x == None    # prints True, but is not Pythonic
print x is None    # prints True, and is Pythonic

```

Tri

```

x = [4,1,2,3]
y = sorted(x)    # is [1,2,3,4], x is unchanged
x.sort()         # now x is [1,2,3,4]

# sort the list by absolute value from largest to smallest
x = sorted([-4,1,-2,3], key=abs, reverse=True) # is [-4,3,-2,1]
# sort the words and counts from highest count to lowest
wc = sorted(word_counts.items(),
            key=lambda (word, count): count,
            reverse=True)

```

Liste en compréhension

```

even_numbers = [x for x in range(5) if x % 2 == 0] # [0, 2, 4]
squares     = [x * x for x in range(5)]          # [0, 1, 4, 9, 16]
even_squares = [x * x for x in even_numbers]      # [0, 4, 16]

square_dict = { x : x * x for x in range(5) } # { 0:0, 1:1, 2:4, 3:9, 4:16 }
square_set  = { x * x for x in [1, -1] }       # { 1 }

pairs = [(x, y)
          for x in range(10)
          for y in range(10)] # 100 pairs (0,0) (0,1) ... (9,8), (9,9)

increasing_pairs = [(x, y)                  # only pairs with x < y,
                     for x in range(10)    # range(lo, hi) equals
                     for y in range(x + 1, 10)] # [lo, lo + 1, ..., hi - 1]

```

Générateurs et itérateurs

L'opérateur `yield` génère les valeurs.

```

def lazy_range(n):
    """a lazy version of range"""
    i = 0
    while i < n:
        yield i
        i += 1

for i in lazy_range(10):
    do_something_with(i)

def natural_numbers():
    """returns 1, 2, 3, ..."""
    n = 1
    while True:
        yield n
        n += 1

```

Générateur aléatoire

```

import random
four_uniform_randoms = [random.random() for _ in range(4)]
# [0.8444218515250481, # random.random() produces numbers
# 0.7579544029403025, # uniformly between 0 and 1
# 0.420571580830845, # it's the random function we'll use
# 0.25891675029296335] # most often

random.seed(10)      # set the seed to 10
print random.random() # 0.57140259469
random.seed(10)      # reset the seed to 10
print random.random() # 0.57140259469 again

random.randrange(10) # choose randomly from range(10) = [0, 1, ..., 9]
random.randrange(3, 6) # choose randomly from range(3, 6) = [3, 4, 5]

up_to_ten = range(10)
random.shuffle(up_to_ten)
print up_to_ten
# [2, 5, 1, 9, 7, 3, 8, 6, 4, 0] (your results will probably be different)

my_best_friend = random.choice(["Alice", "Bob", "Charlie"]) # "Bob" for me

lottery_numbers = range(60)
winning_numbers = random.sample(lottery_numbers, 6) # [16, 36, 10, 6, 25, 9]

four_with_replacement = [random.choice(range(10))
                        for _ in range(4)]
# [9, 4, 4, 2]

```

Expressions régulières

```
import re
```

```

print all([
    not re.match("a", "cat"),           # * 'cat' doesn't start with 'a'
    re.search("a", "cat"),              # * 'cat' has an 'a' in it
    not re.search("c", "dog"),          # * 'dog' doesn't have a 'c' in it
    3 == len(re.split("[ab]", "carbs")), # * split on a or b to ['c','r','s']
    "R-D-" == re.sub("[0-9]", "-", "R2D2") # * replace digits with dashes
]) # prints True

```

Programmation Objet

```

# by convention, we give classes PascalCase names
class Set:
    # these are the member functions
    # every one takes a first parameter "self" (another convention)
    # that refers to the particular Set object being used
    def __init__(self, values=None):
        """This is the constructor.
        It gets called when you create a new Set.
        You would use it like

        s1 = Set()      # empty set
        s2 = Set([1,2,2,3]) # initialize with values"""
        self.dict = {} # each instance of Set has its own dict property
                       # which is what we'll use to track memberships
    if values is not None:
        for value in values:
            self.add(value)
    def __repr__(self):
        """this is the string representation of a Set object
        if you type it at the Python prompt or pass it to str()
        return "Set: " + str(self.dict.keys())
    # we'll represent membership by being a key in self.dict with value True
    def add(self, value):
        self.dict[value] = True
    # value is in the Set if it's a key in the dictionary
    def contains(self, value):
        return value in self.dict
    def remove(self, value):
        del self.dict[value]

```

Par la suite :

```

s = Set([1,2,3])
s.add(4)
print s.contains(4)    # True
s.remove(3)
print s.contains(3)    # False

```

Enumérer

```
# not Pythonic
```

```

for i in range(len(documents)):
    document = documents[i]
    do_something(i, document)
# also not Pythonic
i = 0
for document in documents:
    do_something(i, document)
    i += 1

for i, document in enumerate(documents):
    do_something(i, document)

for i in range(len(documents)): do_something(i)      # not Pythonic
for i, _ in enumerate(documents): do_something(i)    # Pythonic

```

Compression

```

list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
zip(list1, list2)      # is [('a', 1), ('b', 2), ('c', 3)]

pairs = [('a', 1), ('b', 2), ('c', 3)]
letters, numbers = zip(*pairs)

zip(('a', 1), ('b', 2), ('c', 3))

def add(a, b): return a + b
add(1, 2)      # returns 3
add([1, 2])    # TypeError!
add(*[1, 2])  # returns 3

```

Arguments

```

def doubler(f):
    def g(x):
        return 2 * f(x)
    return g

def f1(x):
    return x + 1
g = doubler(f1)
print g(3)      # 8 (== ( 3 + 1) * 2)
print g(-1)     # 0 (== (-1 + 1) * 2)

def f2(x, y):
    return x + y
g = doubler(f2)
print g(1, 2)   # TypeError: g() takes exactly 1 argument (2 given)

def magic(*args, **kwargs):
    print "unnamed args:", args

```

```

        print "keyword args:", kwargs
magic(1, 2, key="word", key2="word2")
# prints
# unnamed args: (1, 2)
# keyword args: {'key2': 'word2', 'key': 'word'}
```

```

def other_way_magic(x, y, z):
    return x + y + z
x_y_list = [1, 2]
z_dict = { "z" : 3 }
print other_way_magic(*x_y_list, **z_dict)  # 6
```

```

def doubler_correct(f):
    """works no matter what kind of inputs f expects"""
    def g(*args, **kwargs):
        """whatever arguments g is supplied, pass them through to f"""
        return 2 * f(*args, **kwargs)
    return g
g = doubler_correct(f2)
print g(1, 2) # 6
```

9.5 Prétraitement des données

9.5.1 Visualisation

Nous utilisons la librairie `matplotlib` pour différentes visualisations.

```

from matplotlib import pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')
# add a title
plt.title("Nominal GDP")
# add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()
```

Courbes bars

```

movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story"]
num_oscars = [5, 11, 3, 8, 10]
# bars are by default width 0.8, so we'll add 0.1 to the left coordinates
# so that each bar is centered
xs = [i + 0.1 for i, _ in enumerate(movies)]
# plot bars with left x-coordinates [xs], heights [num_oscars]
plt.bar(xs, num_oscars)
plt.ylabel("# of Academy Awards")
plt.title("My Favorite Movies")
```

```
# label x-axis with movie names at bar centers
plt.xticks([i + 0.5 for i, _ in enumerate(movies)], movies)
plt.show()

Puis,

grades = [83,95,91,87,70,0,85,82,100,67,73,77,0]
decile = lambda grade: grade // 10 * 10
histogram = Counter(decile(grade) for grade in grades)
plt.bar([x - 4 for x in histogram.keys()], # shift each bar to the left by 4
        histogram.values(),           # give each bar its correct height
        8)                           # give each bar a width of 8
plt.axis([-5, 105, 0, 5])           # x-axis from -5 to 105,
                                    # y-axis from 0 to 5
plt.xticks([10 * i for i in range(11)]) # x-axis labels at 0, 10, ..., 100
plt.xlabel("Decile")
plt.ylabel("# of Students")
plt.title("Distribution of Exam 1 Grades")
plt.show()
```

Courbes

```
variance    = [1, 2, 4, 8, 16, 32, 64, 128, 256]
bias_squared = [256, 128, 64, 32, 16, 8, 4, 2, 1]
total_error  = [x + y for x, y in zip(variance, bias_squared)]
xs = [i for i, _ in enumerate(variance)]
# we can make multiple calls to plt.plot
# to show multiple series on the same chart
plt.plot(xs, variance,    'g-', label='variance')    # green solid line
plt.plot(xs, bias_squared, 'r-.', label='bias^2')     # red dot-dashed line
plt.plot(xs, total_error, 'b:', label='total error') # blue dotted line
# because we've assigned labels to each series
# we can get a legend for free
# loc=9 means "top center"
plt.legend(loc=9)
plt.xlabel("model complexity")
plt.title("The Bias-Variance Tradeoff")
plt.show()
```

Scatterplots

```
friends = [ 70, 65, 72, 63, 71, 64, 60, 64, 67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
plt.scatter(friends, minutes)
# label each point
for label, friend_count, minute_count in zip(labels, friends, minutes):
    plt.annotate(label,
                 xy=(friend_count, minute_count), # put the label with its point
                 xytext=(5, -5),                  # but slightly offset
                 textcoords='offset points')
```

```

plt.title("Daily Minutes vs. Number of Friends")
plt.xlabel("# of friends")
plt.ylabel("daily minutes spent on the site")
plt.show()

```

Puis,

```

test_1_grades = [ 99, 90, 85, 97, 80]
test_2_grades = [100, 85, 60, 90, 70]
plt.scatter(test_1_grades, test_2_grades)
plt.title("Axes Aren't Comparable")
plt.xlabel("test 1 grade")
plt.ylabel("test 2 grade")
plt.show()

```

9.5.2 Importation des données

stdin et stdout Soit le programme "egrep.py" qui affiche les lignes qui contiennent une chaîne spécifiée par le premier argument.

```

# egrep.py
import sys, re
# sys.argv is the list of command-line arguments
# sys.argv[0] is the name of the program itself
# sys.argv[1] will be the regex specified at the command line
regex = sys.argv[1]
# for every line passed into the script
for line in sys.stdin:
    # if it matches the regex, write it to stdout
    if re.search(regex, line):
        sys.stdout.write(line)

```

Soit le programme "line_count.py" qui comptabilise le nombre de lignes d'une entrée :

```

# line_count.py
import sys
count = 0
for line in sys.stdin:
    count += 1
# print goes to sys.stdout
print(count)

```

La ligne de commande suivante, effectue ce comptage sur un fichier "SomeFile.txt" :

```
type SomeFile.txt | python egrep.py "[0-9]" | python line_count.py
```

Soit un programme d'affichage des mots les plus fréquents dans un fichier :

```
# most_common_words.py
import sys
from collections import Counter
# pass in number of words as first argument
try:
    num_words = int(sys.argv[1])
except:
    print("usage: most_common_words.py num_words")
    sys.exit(1) # non-zero exit code indicates error
counter = Counter(word.lower()) # lowercase words
    for line in sys.stdin #
        for word in line.strip().split() # split on spaces
            if word) # skip empty 'words'
for word, count in counter.most_common(num_words):
    sys.stdout.write(str(count))
    sys.stdout.write("\t")
    sys.stdout.write(word)
    sys.stdout.write("\n")
```

L'appel se fait via :

```
type aa.txt | python most_common_words.py 10
```

Fichiers texte # 'r' means read-only

```
file_for_reading = open('reading_file.txt', 'r')
# 'w' is write -- will destroy the file if it already exists!
file_for_writing = open('writing_file.txt', 'w')
# 'a' is append -- for adding to the end of the file
file_for_appending = open('appending_file.txt', 'a')
# don't forget to close your files when you're done
file_for_writing.close()
```

Données tabulaires

Soit un fichier contenant les données tabulaires :

```
6/20/2014 AAPL 90.91
6/20/2014 MSFT 41.68
6/20/2014 FB 64.5
6/19/2014 AAPL 91.86
6/19/2014 MSFT 41.51
6/19/2014 FB 64.34
```

Son traitement se fait comme suit :

```
import csv
with open('tab_delimited_stock_prices.txt', 'rt') as f:
    reader = csv.reader(f, delimiter='\t')
    for row in reader:
        date = row[0]
        symbol = row[1]
        closing_price = float(row[2])
        print(date, symbol, closing_price)
```

On peut aussi nommer les colonnes et traiter via les noms :

```

date:symbol:closing_price
6/20/2014:AAPL:90.91
6/20/2014:MSFT:41.68
6/20/2014:FB:64.5

with open('colon_delimited_stock_prices.txt', 'rb') as f:
    reader = csv.DictReader(f, delimiter=':')
    for row in reader:
        date = row["date"]
        symbol = row["symbol"]
        closing_price = float(row["closing_price"])
        print(date, symbol, closing_price)

```

Données web Les données disponibles à distance, via web, peuvent aussi être chargées. La librairie `BeautifulSoup` permet de récupérer le contenu web avec toutes ses balises. Il est nécessaire d'installer le paquetage `html5lib`.

Par exemple, la récupérer des données peut se faire comme suit :

```

from bs4 import BeautifulSoup
import requests
html = requests.get("http://www.example.com").text
soup = BeautifulSoup(html, 'html5lib')

```

Pour repérer le premier contenu de la balise `<p>`, il suffit d'invoquer :

```
first_paragraph = soup.find('p')      # or just soup.p
```

On peut aussi y accéder directement :

```

first_paragraph_text = soup.p.text
first_paragraph_words = soup.p.text.split()

```

On peut récupérer tous les contenus via :

```

all_paragraphs = soup.find_all('p')  # or just soup('p')
paragraphs_with_ids = [p for p in soup('p') if p.get('id')]

```

Exemple : parcourir les livres O'Reilly Les livres sont accessibles via :

```
http://shop.oreilly.com/category/browse-subjects/data.do?sortby=publicationDate&page=1
```

La récupération des données des livres se fera via :

```

# you don't have to split the url like this unless it needs to fit in a book
url = "http://shop.oreilly.com/category/browse-subjects/" + \
      "data.do?sortby=publicationDate&page=1"
soup = BeautifulSoup(requests.get(url).text, 'html5lib')

```

Chaque livre est tagué comme suit :

```

<td class="thumbtext">
  <div class="thumbcontainer">
    <div class="thumbdiv">
      <a href="/product/9781118903407.do">
        

```

```

</a>
</div>
</div>
<div class="widthchange">
  <div class="thumbheader">
    <a href="/product/9781118903407.do">Getting a Big Data Job For Dummies</a>
  </div>
  <div class="AuthorName">By Jason Williamson</div>
  <span class="directorydate"> December 2014 </span>
  <div style="clear:both;">
    <div id="146350">
      <span class="pricelabel">
        Ebook:
        <span class="price">&nbsp;$29.99</span>
      </span>
    </div>
  </div>
</div>
</td>

```

Pour récupérer le nombre de livres :

```

tds = soup('td', 'thumbtext')
print len(tds)
# 30

```

Pour avoir le titre, auteur, ... :

```

title = td.find("div", "thumbheader").a.text
author_name = td.find('div', 'AuthorName').text
authors = [x.strip() for x in re.sub("^By ", "", author_name).split(",")]
isbn_link = td.find("div", "thumbheader").a.get("href")
# re.match captures the part of the regex in parentheses
isbn = re.match("/product/(.*).do", isbn_link).group(1)
date = td.find("span", "directorydate").text.strip()

```

On peut retourner tout ça dans une seule structure :

```

def book_info(td):
    """given a BeautifulSoup <td> Tag representing a book,
    extract the book's details and return a dict"""
    title = td.find("div", "thumbheader").a.text
    by_author = td.find('div', 'AuthorName').text
    authors = [x.strip() for x in re.sub("^By ", "", by_author).split(",")]
    isbn_link = td.find("div", "thumbheader").a.get("href")
    isbn = re.match("/product/(.*).do", isbn_link).groups()[0]
    date = td.find("span", "directorydate").text.strip()
    return {
        "title" : title,
        "authors" : authors,
        "isbn" : isbn,
        "date" : date
    }

```

On peut explorer d'une façon répétitive les 30 pages comme suit :

```
from time import sleep
base_url = "http://shop.oreilly.com/category/browse-subjects/" + \
           "data.do?sortby=publicationDate&page="
books = []
NUM_PAGES = 31    # at the time of writing, probably more by now
for page_num in range(1, NUM_PAGES + 1):
    print("souping page", page_num, ", ", len(books), " found so far")
    url = base_url + str(page_num)
    soup = BeautifulSoup(requests.get(url).text, 'html5lib')
    for td in soup('td', 'thumbtext'):
        if not is_video(td):
            books.append(book_info(td))
# now be a good citizen and respect the robots.txt!
sleep(30)
```

Maintenant, on peut visualiser le nombre de livres par années :

```
def get_year(book):
    """book["date"] looks like 'November 2014' so we need to
    split on the space and then take the second piece"""
    return int(book["date"].split()[1])
# 2014 is the last complete year of data (when I ran this)
year_counts = Counter(get_year(book) for book in books
                      if get_year(book) <= 2014)
import matplotlib.pyplot as plt
years = sorted(year_counts)
book_counts = [year_counts[year] for year in years]
plt.plot(years, book_counts)
plt.ylabel("# of data books")
plt.title("Data is Big!")
plt.show()
```

Parser les fichiers JSON On peut extraire le contenu d'un fichier JSON dans une structure comme suit :

```
import json
serialized = """{ "title" : "Data Science Book",
                  "author" : "Joel Grus",
                  "publicationYear" : 2014,
                  "topics" : [ "data", "science", "data science" ] }"""
# parse the JSON to create a Python dict
deserialized = json.loads(serialized)
if "data science" in deserialized["topics"]:
    print(deserialized)
```

Soit l'exemple de l'invocation via web du dict Python :

```
import requests, json
endpoint = "https://api.github.com/users/joelgrus/repos"
repos = json.loads(requests.get(endpoint).text)
```

On peut extraire plusieurs infos comme suit :

```
from collections import Counter
from dateutil.parser import parse
dates = [parse(repo["created_at"]) for repo in repos]
month_counts = Counter(date.month for date in dates)
weekday_counts = Counter(date.weekday() for date in dates)
```

9.5.3 Exploration des données

Exploration des données en 1D Soit la génération d'un vecteur 1D :

```
import random
random.seed(0)
# uniform between -100 and 100
uniform = [200 * random.random() - 100 for _ in range(10000)]
```

Soit les trois fonctions pour uniformiser la visualisation des données :

```
def bucketize(point, bucket_size):
    """floor the point to the next lower multiple of bucket_size"""
    return bucket_size * math.floor(point / bucket_size)
def make_histogram(points, bucket_size):
    """buckets the points and counts how many in each bucket"""
    return Counter(bucketize(point, bucket_size) for point in points)
def plot_histogram(points, bucket_size, title=""):
    histogram = make_histogram(points, bucket_size)
    plt.bar(histogram.keys(), histogram.values(), width=bucket_size)
    plt.title(title)
    plt.show()
```

On peut maintenant visualiser nos données 1D comme suit :

```
plot_histogram(uniform, 10, "Uniform Histogram")
```

Exploration des données en 2D Soit la fonction qui génère les valeurs suivant la loi normale¹ :

1. https://fr.wikipedia.org/wiki/Loi_normale

En théorie des probabilités et en statistique, la loi normale est l'une des lois de probabilité les plus adaptées pour modéliser des phénomènes naturels issus de plusieurs événements aléatoires. Plus formellement, c'est une loi de probabilité absolument continue qui dépend de deux paramètres : son espérance, un nombre réel noté μ , et son écart type, un nombre réel positif noté σ . La densité de probabilité de la loi normale est donnée par :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}.$$

La courbe de cette densité est appelée courbe de Gauss ou courbe en cloche, entre autres. C'est la représentation la plus connue de cette loi. La loi normale de moyenne nulle et d'écart type unitaire est appelée loi normale centrée réduite ou loi normale standard. Lorsqu'une variable aléatoire X suit la loi normale, elle est dite gaussienne ou normale et il est habituel d'utiliser la notation avec la variance σ^2 :

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

Parmi les lois de probabilité, elle prend une place particulière grâce au théorème central limite.

```

def normal_cdf(x, mu=0,sigma=1):
    return (1 + math.erf((x - mu) / math.sqrt(2) / sigma)) / 2

def inverse_normal_cdf(p, mu=0, sigma=1, tolerance=0.00001):
    """find approximate inverse using binary search"""
    # if not standard, compute standard and rescale
    if mu != 0 or sigma != 1:
        return mu + sigma * inverse_normal_cdf(p, tolerance=tolerance)
    low_z, low_p = -10.0, 0           # normal_cdf(-10) is (very close to) 0
    hi_z, hi_p = 10.0, 1             # normal_cdf(10) is (very close to) 1
    while hi_z - low_z > tolerance:
        mid_z = (low_z + hi_z) / 2    # consider the midpoint
        mid_p = normal_cdf(mid_z)     # and the cdf's value there
        if mid_p < p:
            # midpoint is still too low, search above it
            low_z, low_p = mid_z, mid_p
        elif mid_p > p:
            # midpoint is still too high, search below it
            hi_z, hi_p = mid_z, mid_p
        else:
            break
    return mid_z

```

Soit la génération de deux vecteurs sur des valeurs provenant de la loi normale :

```

def random_normal():
    """returns a random draw from a standard normal distribution"""
    return inverse_normal_cdf(random.random())
xs = [random_normal() for _ in range(1000)]
ys1 = [x + random_normal() / 2 for x in xs]
ys2 = [-x + random_normal() / 2 for x in xs]

```

On peut visualiser en 2D sur les deux vecteurs :

```

plt.scatter(xs, ys1, marker='.', color='black', label='ys1')
plt.scatter(xs, ys2, marker='.', color='gray', label='ys2')
plt.xlabel('xs')
plt.ylabel('ys')
plt.legend(loc=9)
plt.title("Very Different Joint Distributions")
plt.show()

```

On peut visualiser leurs corrélations :

```

print correlation(xs, ys1)      # 0.9
print correlation(xs, ys2)      # -0.9

```

En effet, elle correspond au comportement, sous certaines conditions, d'une suite d'expériences aléatoires similaires et indépendantes lorsque le nombre d'expériences est très élevé. Grâce à cette propriété, la loi normale permet d'approcher d'autres lois et ainsi de modéliser de nombreuses études scientifiques comme des mesures d'erreurs ou des tests statistiques, en utilisant par exemple les tables de la loi normale.

Proba ... les basiques Soit la table `num_friends` (combien d'amis ont les membres d'une communauté de data scientists) :

```
num_friends = [100, 49, 41, 40, 25,
               # ... and lots more
               ]
```

Servant d'exemple, on les génère aléatoirement :

```
import random
random.seed(0)
num_friends = [math.ceil(100*random.random()) for _ in range(100)]
```

On peut visualiser comme suit :

```
friend_counts = Counter(num_friends)
xs = range(101)                      # largest value is 100
ys = [friend_counts[x] for x in xs]    # height is just # of friends
plt.bar(xs, ys)
plt.axis([0, 101, 0, 25])
plt.title("Histogram of Friend Counts")
plt.xlabel("# of friends")
plt.ylabel("# of people")
plt.show()
```

Les propriétés de la table peuvent être calculées simplement via :

```
num_points = len(num_friends)
largest_value = max(num_friends)        #
smallest_value = min(num_friends)       #
sorted_values = sorted(num_friends)
smallest_value = sorted_values[0]        # 1
second_smallest_value = sorted_values[1] # 1
second_largest_value = sorted_values[-2] # 49

# this isn't right if you don't from __future__ import division
def mean(x):
    return sum(x) / len(x)

mean(num_friends) # 7.333333

def median(v):
    """finds the 'middle-most' value of v"""
    n = len(v)
    sorted_v = sorted(v)
    midpoint = n // 2
    if n % 2 == 1:
        # if odd, return the middle value
        return sorted_v[midpoint]
    else:
        # if even, return the average of the middle values
        lo = midpoint - 1
```

```

        hi = midpoint
        return (sorted_v[lo] + sorted_v[hi]) / 2

median(num_friends) # 6.0

def quantile(x, p):
    """returns the pth-percentile value in x"""
    p_index = int(p * len(x))
    return sorted(x)[p_index]
quantile(num_friends, 0.10) # 1
quantile(num_friends, 0.25) # 3
quantile(num_friends, 0.75) # 9
quantile(num_friends, 0.90) # 13

```

Proba : dispersion

```

# "range" already means something in Python, so we'll use a different name
def data_range(x):
    return max(x) - min(x)
data_range(num_friends) # 99

```

La variance² peut être calculée comme suit :

```

from numpy import dot
def sum_of_squares(v):
    """v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)
def de_mean(x):
    """translate x by subtracting its mean (so the result has mean 0)"""
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]
def variance(x):
    """assumes x has at least two elements"""
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)
variance(num_friends) # 81.54

```

Puis l'écart type :

2. [https://fr.wikipedia.org/wiki/Variance_\(statistiques_et_probabilitÃ's](https://fr.wikipedia.org/wiki/Variance_(statistiques_et_probabilitÃ's)

En statistique et en théorie des probabilités, la variance est une mesure servant à caractériser la dispersion d'un échantillon ou d'une distribution. Pour calculer la variance d'une série statistique ou d'une variable aléatoire, on calcule les écarts entre la série (ou la variable) et sa moyenne (ou espérance) puis on prend la moyenne (ou l'espérance) de ces écarts élevés au carré. Synthétiquement, on écrit :

$$V(X) = E[(X - E(X))^2].$$

La variance correspond au carré de l'écart-type (noté?), et les notations pour la désigner sont multiples. On trouvera par exemple :

$$V(X) = Var(X) = \sigma^2(X).$$

```
def standard_development(x):
    return math.sqrt(variance(x))
standard_development(num_friends) # 9.03
```

La covariance³ entre deux vecteurs :

```
def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)

daily_minutes = [10, 9, 4, 4, 2, 6, 7, 10]
# Pour l'exemple, on les génère aléatoirement ...
import random
random.seed(0)
daily_minutes = [math.ceil(120*random.random()) for _ in range(100)]

covariance(num_friends, daily_minutes) # 22.43
```

où `daily_minutes` est le nombre de minutes que passe la personne sur le forum.

La corrélation⁴ entre deux vecteurs se calcule :

3. <https://fr.wikipedia.org/wiki/Covariance>

En théorie des probabilités et en statistique, la covariance entre deux variables aléatoires est un nombre permettant de quantifier leurs écarts conjoints par rapport à leurs espérances respectives. Elle s'utilise également pour deux séries de données numériques (écarts par rapport aux moyennes).

$$Cov(X, Y) = E[(X - E(X))(Y - E(Y))]$$

La variance de X est donc $Var(X) = Cov(X, X)$. La covariance de deux variables aléatoires indépendantes est nulle, bien que la réciproque ne soit pas toujours vraie. Ce concept se généralise naturellement à plusieurs variables (vecteur aléatoire) par la matrice de covariance (ou matrice de variance-covariance) qui, pour un ensemble de p variables aléatoires réelles X_1, \dots, X_p est la matrice carrée dont l'élément de la ligne i et de la colonne j est la covariance des variables X_i et X_j . Cette matrice permet de quantifier la variation de chaque variable par rapport à chacune des autres. La forme normalisée de la matrice de covariance est la matrice de corrélation.

4. [https://fr.wikipedia.org/wiki/CorrÃ©lation_\(statistiques\)](https://fr.wikipedia.org/wiki/CorrÃ©lation_(statistiques))

En probabilités et en statistiques, étudier la corrélation entre deux ou plusieurs variables aléatoires ou statistiques numériques, c'est étudier l'intensité de la liaison qui peut exister entre ces variables. Le type le plus simple de liaison est la relation affine. Dans le cas de deux variables numériques, elle se calcule à travers une régression linéaire. La mesure de la corrélation linéaire entre les deux se fait alors par le calcul du coefficient de corrélation linéaire, noté r. Ce coefficient est égal au rapport de leur covariance et du produit non nul de leurs écarts types. Le coefficient de corrélation est compris entre -1 et 1.

Le fait que deux variables soient « fortement corrélées » ne démontre pas qu'il y ait une relation de causalité entre l'une et l'autre. Le contre-exemple le plus typique est celui où elles sont en fait liées par une causalité commune. Cette confusion est connue sous l'expression Cum hoc ergo propter hoc.

Le coefficient de corrélation entre deux variables aléatoires réelles X et Y ayant chacune une variance finie, noté $Cor(X, Y)$ ou parfois ρ_{XY} ou simplement r_p , est défini par

$$r = \frac{Cov(X, Y)}{\sigma_X \sigma_Y}.$$

Il est égal à 1 dans le cas où l'une des variables est une fonction affine croissante de l'autre variable,

```

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return 0    # if no variation, correlation is zero
correlation(num_friends, daily_minutes) # 0.25

```

La personne qui a 100 amis et qui passe une minute par jour est une valeur aberrante. La corrélation est sensible à ces valeurs aberrante. Que se passe-t-il si on les ignore ?

```

outlier = num_friends.index(100)    # index of outlier
num_friends_good = [x
                     for i, x in enumerate(num_friends)
                     if i != outlier]
daily_minutes_good = [x
                      for i, x in enumerate(daily_minutes)
                      if i != outlier]
correlation(num_friends_good, daily_minutes_good)

```

Plusieurs dimensions On définit la forme de la données suivant ses dimensions :

```

def shape(A):
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0    # number of elements in first row
    return num_rows, num_cols

```

On utilisera la corrélation entre les dimensions de la donnée :

```

def correlation_matrix(data):
    """returns the num_columns x num_columns matrix whose (i, j)th entry
    is the correlation between columns i and j of data"""
    _, num_columns = shape(data)
    def matrix_entry(i, j):
        return correlation(get_column(data, i), get_column(data, j))
    return make_matrix(num_columns, num_columns, matrix_entry)

```

On peut visualiser sur toutes les dimensions :

```

import matplotlib.pyplot as plt
_, num_columns = shape(data)
fig, ax = plt.subplots(num_columns, num_columns)
for i in range(num_columns):

```

à -1 dans le cas où une variable est une fonction affine et décroissante. Les valeurs intermédiaires renseignent sur le degré de dépendance linéaire entre les deux variables. Plus le coefficient est proche des valeurs extrêmes -1 et 1, plus la corrélation linéaire entre les variables est forte ; on emploie simplement l'expression "fortement corrélées" pour qualifier les deux variables. Une corrélation égale à 0 signifie que les variables ne sont pas corrélées linéairement, elles peuvent néanmoins être corrélées non-linéairement.

```

for j in range(num_columns):
    # scatter column _j on the x-axis vs column _i on the y-axis
    if i != j: ax[i][j].scatter(get_column(data, j), get_column(data, i))
    # unless i == j, in which case show the series name
    else: ax[i][j].annotate("series " + str(i), (0.5, 0.5),
                           xycoords='axes fraction',
                           ha="center", va="center")
    # then hide axis labels except left and bottom charts
    if i < num_columns - 1: ax[i][j].xaxis.set_visible(False)
    if j > 0: ax[i][j].yaxis.set_visible(False)
    # fix the bottom right and top left axis labels, which are wrong because
    # their charts only have text in them
    ax[-1][-1].set_xlim(ax[0][-1].get_xlim())
    ax[0][0].set_ylim(ax[0][1].get_ylim())
plt.show()

```

Nettoyage des données

Exemple de manipulation

Mise à l'échelle

Réduction des dimensions

9.6 Prise en main sur les méthodes de fouille de données

L'objectif de la fouille de données (et de l'apprentissage automatique (machine learning)) est la construction d'un modèle sur un ensemble de données. Le volet "apprentissage" vient du fait que le modèle contient des paramètres qui doivent être fixés, ou plus précisément "appris". Une fois ces paramètres "appris" sur des données observées en entrée, le modèle peut être utilisé pour appréhender de nouvelles données. Il y a deux grandes catégories d'apprentissage :

Apprentissage supervisé L'objectif est de dresser un modèle entre les attributs observés en entrée, et un attribut disponible particulier appelé étiquette ou classe. Une fois ce modèle est construit, on peut l'utiliser pour trouver l'étiquette d'une nouvelle donnée. Deux grandes méthodes non supervisées sont développées : (1) méthodes par régression, quand l'étiquette est un attribut continu ; (2) méthodes par classification quand l'étiquette est une valeur discrète.

Apprentissage non-supervisé Ici, on veut construire un modèle sans connaissance à priori d'une quelconque étiquette ou classe sur les données disponibles. Parmi ces méthodes, on peut citer : (1) Le clustering qui vise à partitionner les données en plusieurs groupes ; (2) méthodes de réduction des dimensions, qui visent à représenter les données d'une façon plus compacte ; (3) méthodes orientées motifs, qui vont repérer des régularités appelées motifs, dans les données.

9.6.1 Introduction à SciKit-Learn

[SciKit-Learn](#) est une librairie Python qui contient les algorithmes implémentés de fouille. La forme standard de représentation des données en vue d'un traitement par un algorithme de fouille, est la forme tabulaire en deux dimensions. [seaborn](#) est une librairie de visualisation, qui contient aussi notamment des exemples de données. Soit par exemple le dataset [Iris](#) :

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

On peut visualiser ces données comme suit :

```
%matplotlib inline
import seaborn as sns; sns.set()
sns.pairplot(iris, hue='species', size=1.5);
```

Si on veut éliminer la classe :

```
X_iris = iris.drop('species', axis=1)
X_iris.shape
```

Et si on s'intéresse uniquement à la classe :

```
y_iris = iris['species']
y_iris.shape
```

Pour construire le modèle de fouille en vue dans [SciKit-Learn](#), on procède comme suit :

- (1) *Choix type de modèle* Repérer le type de modèle adapté aux données disponibles.
 - (2) *Fixer le modèle* Fixer les paramètres usuels du modèle.
 - (3) *Formatage des données* Mettre en forme les données dans le format d'entrée de l'algorithme de fouille.
 - (4) *Génération du modèle* Appliquer l'algorithme de fouille.
 - (5) *Application du modèle* Appliquer le modèle sur de nouvelles données.
- Nous donnons ci-dessous les différents types de fouille sur des exemples.

Fouille par régression Soit un dataset sur deux attributs.

```
import matplotlib.pyplot as plt
import numpy as np
rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)
plt.scatter(x, y);
```

Procédons à l'utilisation de la régression sur nos données :

Choix type de modèle Via
`from sklearn.linear_model import LinearRegression`

Fixer le modèle Dès lors que nous avons choisi une régression, cette dernière nécessite certaines options comme par exemple : (1) veut-on matcher toutes les données ? (2) veut-on travailler sur une forme normalisée ? (3) y-a-t-il nécessité de prétraiter les données ? (4) quelle degré de régularisation ?

Dans notre exemple, on voudrait que l'option `fit_intercept` soit vraie.
`model = LinearRegression(fit_intercept=True)`

Formatage des données Les données sont souvent dans une forme tabulaire en 2D, et les étiquette/sortie souvent un vecteur. Ici `y` est déjà bien formaté. Par contre les données en entrée ne sont pas encore mis en forme tabulaire. On procèdera comme suit en ajoutant une dimension supplémentaire :

```
X = x[:, np.newaxis]
X.shape
```

Génération du modèle Nous appliquons maintenant le modèle :

```
model.fit(X, y)
```

Suite à cette exécution, le modèle est généré avec toutes ses données propres au modèle, comme par exemple les coefficients de la régression :
`model.coef_`

Application du modèle Dès lors que c'est une méthode supervisée, on pourra prédire avec le modèle généré l'étiquette d'une nouvelle donnée :

```
xfit = np.linspace(-1, 11)
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

On peut visualiser le tout comme suit :

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```

Classification supervisée sur l'exemple Iris On se pose ici une question supplémentaire : de combien notre modèle par régression est bon pour prédire la classe (type) ?

Nous utiliseront cette fois-ici une autre méthode celle du classifieur bayésien naïf.

Pour évaluer les performances de notre modèle, nous décomposons nos données en deux sous-ensembles : sous-ensemble d'apprentissage, un sous-ensemble de test.

```
from sklearn.cross_validation import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris, random_state=1)
```

Nous appliquons notre protocole d'apprentissage en 5 étapes :

```
from sklearn.naive_bayes import GaussianNB # 1. choose model class
model = GaussianNB()                      # 2. instantiate model
model.fit(Xtrain, ytrain)                  # 3. fit model to data
y_model = model.predict(Xtest)
```

Nous pouvons maintenant faire appel à une méthode qui calcule la précision de notre prédiction :

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Apprentissage non supervisée : réduction des dimensions Nous ferons appel à une PCA pour réduire les 4 dimensions en deux comme convenu, via les 5 étapes :

```
from sklearn.decomposition import PCA # 1. Choose the model class
model = PCA(n_components=2) # 2. Instantiate the model with hyperparameters
model.fit(X_iris) # 3. Fit to data. Notice y is not specified!
X_2D = model.transform(X_iris) # 4. Transform the data to two dimensions
```

On pourra visualiser les résultats :

```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False);
```

On voit bien que les 3 classes sont assez bien regroupées.

Apprentissage non supervisée : clustering L'objectif du clustering est de partitionner les données dans des groupes distincts. Au lieu de faire appel au classique **k-means**, nous ferons plutôt appel à une toute autre méthode, celle GMM (Gaussian mixture model). Appliquons les 5 étapes :

```
from sklearn.mixture import GMM # 1. Choose the model class
model = GMM(n_components=3,
            covariance_type='full') # 2. Instantiate the model w/ hyperparameters
model.fit(X_iris) # 3. Fit to data. Notice y is not specified!
y_gmm = model.predict(X_iris) # 4. Determine cluster labels
```

Visualisons :

```
iris['cluster'] = y_gmm
sns.lmplot("PCA1", "PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```

Remarquons que les deux premiers clusters sont nets, alors que le dernier contient un bruit ... à commenter.

9.6.2 Application à des chiffres écrits à la main

Chargement et visualisation des données Nous importons les images :

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

Il y a 1 797 échantillons, où chaque image est de taille 8x8. Visualisons les 100 premières images :

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```

Chargeons la matrice des données et le vecteur classe :

```
X = digits.data
X.shape
y = digits.target
y.shape
```

Réduction des dimensions Comme nous avons 64 dimensions, il est insurmontable de visualiser les points. Nous réduirons les dimensions à 2 en faisant appel à la méthode Isomap (manifold learning).

```
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
data_projected = iso.transform(digits.data)
data_projected.shape
```

Visualisons :

```
plt.scatter(data_projected[:, 0], data_projected[:, 1], c=digits.target,
edgecolor='none', alpha=0.5,
cmap=plt.cm.get_cmap('Spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```

On voit bien qu'il y a un partitionnement très visible en 10 groupes !

Classification Faisons appel au classifieur bayésien naïf :

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

Evaluons la précision :

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Pour évaluer pourquoi il y a 20% d'échec on peut faire appel à la méthode "confusion matrix" :

```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(ytest, y_model)
sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```

Cette matrice comptabilise le nombre d'échec

On peut encore mettre en évidence les chiffres mal prédis :

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
subplot_kw={‘xticks’:[], ‘yticks’:[]},
gridspec_kw=dict(hspace=0.1, wspace=0.1))
```

```

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(y_model[i]),
            transform=ax.transAxes,
            color='green' if (ytest[i] == y_model[i]) else 'red')

```

9.7 Fouille sur les réseaux

Soit les individus :

```

users = [
    { "id": 0, "name": "Hero" },
    { "id": 1, "name": "Dunn" },
    { "id": 2, "name": "Sue" },
    { "id": 3, "name": "Chi" },
    { "id": 4, "name": "Thor" },
    { "id": 5, "name": "Clive" },
    { "id": 6, "name": "Hicks" },
    { "id": 7, "name": "Devin" },
    { "id": 8, "name": "Kate" },
    { "id": 9, "name": "Klein" }
]

```

Et la relation de proximité entre eux :

```

friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
               (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]

```

Nous pouvons ainsi établir une table explicite de cette relation :

```

for user in users:
    user["friends"] = []
for i, j in friendships:
    # this works because users[i] is the user whose id is i
    users[i]["friends"].append(users[j]) # add i as a friend of j
    users[j]["friends"].append(users[i]) # add j as a friend of i

```

Une des requêtes de base que l'on se pose sur un réseau d'individus, on peut citer en premier la détection des éléments centraux intermédiaires (betweenness centrality). Pour les détecter il est question de trouver tous les chemins dans le graphe, puis repérer les éléments qui sont intermédiaires.

Ci-dessous le code qui permet de trouver tous les chemins entre toutes les paires de sommets :

```

from collections import deque
def shortest_paths_from(from_user):
    # a dictionary from "user_id" to *all* shortest paths to that user
    shortest_paths_to = { from_user["id"] : [[]] }
    # a queue of (previous user, next user) that we need to check.

```

```

# starts out with all pairs (from_user, friend_of_from_user)
frontier = deque((from_user, friend))
    for friend in from_user["friends"])
# keep going until we empty the queue
while frontier:
    prev_user, user = frontier.popleft() # remove the user who's
    user_id = user["id"] # first in the queue
    # because of the way we're adding to the queue,
    # necessarily we already know some shortest paths to prev_user
    paths_to_prev_user = shortest_paths_to[prev_user["id"]]
    new_paths_to_user = [path + [user_id] for path in paths_to_prev_user]
    # it's possible we already know a shortest path
    old_paths_to_user = shortest_paths_to.get(user_id, [])
    # what's the shortest path to here that we've seen so far?
    if old_paths_to_user:
        min_path_length = len(old_paths_to_user[0])
    else:
        min_path_length = float('inf')
    # only keep paths that aren't too long and are actually new
    new_paths_to_user = [path
        for path in new_paths_to_user
        if len(path) <= min_path_length
        and path not in old_paths_to_user]
    shortest_paths_to[user_id] = old_paths_to_user + new_paths_to_user
    # add never-seen neighbors to the frontier
    frontier.extend((user, friend))
        for friend in user["friends"]
            if friend["id"] not in shortest_paths_to)
return shortest_paths_to

```

Puis,

```

for user in users:
    user["shortest_paths"] = shortest_paths_from(user)

```

Pour chaque paire d'individus i et j , nous connaissons les n chemins les reliant, où pour chacun de leurs sommets on ajoute $1/n$ à sa *betweenness* :

```

for user in users:
    user["betweenness_centrality"] = 0.0
for source in users:
    source_id = source["id"]
    for target_id, paths in source["shortest_paths"].iteritems():
        if source_id < target_id: # don't double count
            num_paths = len(paths) # how many shortest paths?
            contrib = 1 / num_paths # contribution to centrality
            for path in paths:
                for id in path:
                    if id not in [source_id, target_id]:
                        users[id]["betweenness_centrality"] += contrib

```

Une deuxième requête répondue consiste à établir le degré de *fermeture* d'un individu. Plus elle est petite, mieux c'est :

```
def farness(user):
    """the sum of the lengths of the shortest paths to each other user"""
    return sum(len(paths[0])
               for paths in user["shortest_paths"].values()))

for user in users:
    user["closeness_centrality"] = 1 / farness(user)
```

Algorithme PageRank

Dans un réseau social, on peut être intéressé par les personnes qui sont convoitées par d'autres, et non l'inverse. Pour mettre en avant cette dimension, nous revoyons les données :

```
endorsements = [(0, 1), (1, 0), (0, 2), (2, 0), (1, 2),
                (2, 1), (1, 3), (2, 3), (3, 4), (5, 4),
                (5, 6), (7, 5), (6, 8), (8, 7), (8, 9)]
for user in users:
    user["endorses"] = []      # add one list to track outgoing endorsements
    user["endorsed_by"] = []   # and another to track endorsements
for source_id, target_id in endorsements:
    users[source_id]["endorses"].append(users[target_id])
    users[target_id]["endorsed_by"].append(users[source_id])
```

Nous pouvons ainsi repérer les personnes les plus convoitées :

```
endorsements_by_id = [(user["id"], len(user["endorsed_by"]))
                      for user in users]
sorted(endorsements_by_id,
      key=lambda user_id, num_endorsements: num_endorsements,
      reverse=True)
```

Etre populaire n'est pas nécessairement suffisant : il est plus pertinent d'être convoité par une personne qui est elle-même convoitée que par des personnes non convoitées. C'est l'idée même de l'algorithme Pagerank de google.

Le principe de PageRank est le suivant :

1. Il y a un total unitaire 1.0 sur l'ensemble de tout le réseau.
2. Initialement, PageRank est équitablement réparti sur les noeuds du réseau.
3. Les fractions de PageRank d'un noeud est transmis en sortie vers les noeuds suivants.

L'algorithme est donné comme suit :

```
def page_rank(users, damping = 0.85, num_iters = 100):
    # initially distribute PageRank evenly
    num_users = len(users)
    pr = { user["id"] : 1 / num_users for user in users }
```

```

# this is the small fraction of PageRank
# that each node gets each iteration
base_pr = (1 - damping) / num_users
for __ in range(num_iters):
    next_pr = { user["id"] : base_pr for user in users }
    for user in users:
        # distribute PageRank to outgoing links
        links_pr = pr[user["id"]] * damping
        for endorsee in user["endorses"]:
            next_pr[endorsee["id"]] += links_pr / len(user["endorses"])
pr = next_pr
return pr

```

9.8 Systèmes de recommandations

Soit une base des intérêts des utilisateurs :

```

users_interests = [
    ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
    ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
    ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
    ["R", "Python", "statistics", "regression", "probability"],
    ["machine learning", "regression", "decision trees", "libsvm"],
    ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
    ["statistics", "probability", "mathematics", "theory"],
    ["machine learning", "scikit-learn", "Mahout", "neural networks"],
    ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
    ["Hadoop", "Java", "MapReduce", "Big Data"],
    ["statistics", "R", "statsmodels"],
    ["C++", "deep learning", "artificial intelligence", "probability"],
    ["pandas", "R", "Python"],
    ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
    ["libsvm", "regression", "support vector machines"]
]

```

On peut repérer les items les plus populaires :

```

popular_interests = Counter(interest
                            for user_interests in users_interests
                            for interest in user_interests).most_common()

```

On obtient :

```

[('Python', 4),
 ('R', 4),
 ('Java', 3),
 ('regression', 3),
 ('statistics', 3),
 ('probability', 3),
 # ...
]

```

On peut proposer les items les plus populaires que l'utilisateur n'a pas encore acquis :

```
def most_popular_new_interests(user_interests, max_results=5):
    suggestions = [(interest, frequency)
                    for interest, frequency in popular_interests
                    if interest not in user_interests]
    return suggestions[:max_results]
```

Par exemple, l'utilisateur 1, on lui recommande :

```
["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"]
```

```
most_popular_new_interests(users_interests[1], 5)
# [('Python', 4), ('R', 4), ('Java', 3), ('regression', 3), ('statistics', 3)]
```

A l'utilisateur 3, on lui recommande :

```
[('Java', 3),
 ('HBase', 3),
 ('Big Data', 3),
 ('neural networks', 2),
 ('Hadoop', 2)]
```

9.8.1 Recommandations à base de similarité entre les utilisateurs

Etant donné deux vecteurs, on peut calculer leur similarité via la fonction :

```
def cosine_similarity(v, w):
    return dot(v, w) / math.sqrt(dot(v, v) * dot(w, w))
```

Elle mesure l'angle entre v and w . Si v et w pointent dans la même direction, alors le numérateur et le dénominateur sont égaux, et leur cosine est 1. Si v et w pointent dans des directions opposées, alors leurs est -1. Si v est 0 alors le cosine est 0.

Soit les items populaires triés :

```
unique_interests = sorted(list({ interest
                                    for user_interests in users_interests
                                    for interest in user_interests }))
```

On obtient :

```
['Big Data',
 'C++',
 'Cassandra',
 'HBase',
 'Hadoop',
 'Haskell',
 # ...
 ]
```

Soit la fonction qui nous renvoie un vecteur binaire où un livre est à 1 s'il fait partie des items en entrée :

```
def make_user_interest_vector(user_interests):
    """given a list of interests, produce a vector whose ith element is 1
    if unique_interests[i] is in the list, 0 otherwise"""
    return [1 if interest in user_interests else 0
            for interest in unique_interests]
```

On peut construire une matrice globale :

```
user_interest_matrix = map(make_user_interest_vector, users_interests)
```

Ainsi, $\text{user_interest_matrix}[i][j]$ est 1 si l'utilisateur i a pris l'item j .

Vu la petite taille de la base, on peut se permettre une matrice de similarité globale :

```
user_similarities = [[cosine_similarity(interest_vector_i, interest_vector_j)
                      for interest_vector_j in user_interest_matrix]
                      for interest_vector_i in user_interest_matrix]
```

On peut ainsi récupérer les utilisateurs triés suivant leur similarité à un utilisateur donné :

```
def most_similar_users_to(user_id):
    pairs = [(other_user_id, similarity) # find other
              for other_user_id, similarity in
              enumerate(user_similarities[user_id]) # nonzero
              if user_id != other_user_id and similarity > 0] # similarity
    return sorted(pairs, # sort them
                  key=lambda usersim: usersim[1], # most similar
                  reverse=True) # first
```

Pour l'utilisateur 0, on a :

```
[(9, 0.5669467095138409),
 (1, 0.3380617018914066),
 (8, 0.1889822365046136),
 (13, 0.1690308509457033),
 (5, 0.1543033499620919)]
```

On peut ainsi proposer les items suivant leurs poids des autres utilisateurs :

```
def user_based_suggestions(user_id, include_current_interests=False):
    # sum up the similarities
    suggestions = defaultdict(float)
    for other_user_id, similarity in most_similar_users_to(user_id):
        for interest in users_interests[other_user_id]:
            suggestions[interest] += similarity
    # convert them to a sorted list
```

```

suggestions = sorted(suggestions.items(),
                     key=lambda (_, weight): weight,
                     reverse=True)
# and (maybe) exclude already-interests
if include_current_interests:
    return suggestions
else:
    return [(suggestion, weight)
            for suggestion, weight in suggestions
            if suggestion not in users_interests[user_id]]

```

Pour l'utilisateur 0, on a :

```

[('MapReduce', 0.5669467095138409),
 ('MongoDB', 0.50709255283711),
 ('Postgres', 0.50709255283711),
 ('NoSQL', 0.3380617018914066),
 ('neural networks', 0.1889822365046136),
 ('deep learning', 0.1889822365046136),
 ('artificial intelligence', 0.1889822365046136),
 #...
]

```

9.8.2 Recommandations à base de similarité entre items

On transpose l'intérêt calculé précédemment :

```

interest_user_matrix = [[user_interest_vector[j]
                        for user_interest_vector in user_interest_matrix]
                        for j, _ in enumerate(unique_interests)]]

```

On peut calculer les similarités :

```

interest_similarities = [[cosine_similarity(user_vector_i, user_vector_j)
                           for user_vector_j in interest_user_matrix]
                           for user_vector_i in interest_user_matrix]

```

On peut calculer les items populaires similaire à un item :

```

def most_similar_interests_to(interest_id):
    similarities = interest_similarities[interest_id]
    pairs = [(unique_interests[other_interest_id], similarity)
              for other_interest_id, similarity in enumerate(similarities)
              if interest_id != other_interest_id and similarity > 0]
    return sorted(pairs,
                 key=lambda (_, similarity): similarity,
                 reverse=True)

```

Sur 0, on a :

```
[('Hadoop', 0.8164965809277261),
 ('Java', 0.6666666666666666),
 ('MapReduce', 0.5773502691896258),
 ('Spark', 0.5773502691896258),
 ('Storm', 0.5773502691896258),
 ('Cassandra', 0.4082482904638631),
 ('artificial intelligence', 0.4082482904638631),
 ('deep learning', 0.4082482904638631),
 ('neural networks', 0.4082482904638631),
 ('HBase', 0.3333333333333333)]
```

On peut faire les recommandations en sommant les similarités des items :

```
def item_based_suggestions(user_id, include_current_interests=False):
    # add up the similar interests
    suggestions = defaultdict(float)
    user_interest_vector = user_interest_matrix[user_id]
    for interest_id, is_interested in enumerate(user_interest_vector):
        if is_interested == 1:
            similar_interests = most_similar_interests_to(interest_id)
            for interest, similarity in similar_interests:
                suggestions[interest] += similarity
    # sort them by weight
    suggestions = sorted(suggestions.items(),
                          key=lambda _, similarity: similarity,
                          reverse=True)
    if include_current_interests:
        return suggestions
    else:
        return [(suggestion, weight)
                for suggestion, weight in suggestions
                if suggestion not in users_interests[user_id]]
```

Ainsi, l'utilisateur 0 sera recommandé avec :

```
[('MapReduce', 1.861807319565799),
 ('Postgres', 1.3164965809277263),
 ('MongoDB', 1.3164965809277263),
 ('NoSQL', 1.2844570503761732),
 ('programming languages', 0.5773502691896258),
 ('MySQL', 0.5773502691896258),
 ('Haskell', 0.5773502691896258),
 ('databases', 0.5773502691896258),
 ('neural networks', 0.4082482904638631),
 ('deep learning', 0.4082482904638631),
 ('C++', 0.4082482904638631),
 ('artificial intelligence', 0.4082482904638631),
 ('Python', 0.2886751345948129),
 ('R', 0.2886751345948129)]
```

Bibliographie

- [Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann.
- [Agrawal and Srikant, 1995] Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In Yu, P. S. and Chen, A. L. P., editors, *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pages 3–14. IEEE Computer Society.
- [Biernat and Lutz, 2015] Biernat, E. and Lutz, M. (2015). *Data science : fondamentaux et études de cas*. Eyrolles.
- [Bourgelt, 2015] Bourgelt, C. (2015). Frequent pattern mining. Technical report, Support de cours, European Center for Soft Computing, <http://www.borgelt.net/teaching.html>.
- [De Raedt and Zimmermann, 2007] De Raedt, L. and Zimmermann, A. (2007). Constraint-based pattern set mining. In *Proceedings of the Seventh SIAM International Conference on DM*. SIAM.
- [E.Shannon, 1948] E.Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal (juillet et octobre 1948)*, 27 :379–423/623–656.
- [Garofalakis et al., 2002] Garofalakis, M. N., Rastogi, R., and Shim, K. (2002). Mining sequential patterns with regular expression constraints. *IEEE Trans. Knowl. Data Eng.*, 14(3) :530–552.
- [Guns et al., 2011] Guns, T., Nijssen, S., and De Raedt, L. (2011). Itemset mining : A constraint programming perspective. *Artificial Intelligence*, 175(12) :1951–1983.
- [Han, 2005] Han, J. (2005). *Data Mining : Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [Kemmar, 2017] Kemmar, A. (2017). Programmation par contraintes appliquée pour la résolution des problèmes d'appariement d'objets discrets. Technical report, THÈSE, Docteur en Sciences, Université d'Oran 1.
- [Kemmar et al., 2016a] Kemmar, A., Lebbah, Y., Loudni, S., Boizumault, P., and Charnois, T. (2016a). Une contrainte globale pour l'extraction de motifs séquentiels. *Revue d'Intelligence Artificielle*, 30(6) :675–703.
- [Kemmar et al., 2017] Kemmar, A., Lebbah, Y., Loudni, S., Boizumault, P., and Charnois, T. (2017). Prefix-projection global constraint and top-k approach for sequential pattern mining. *Constraints*, 22(2) :265–306.

- [Kemmar et al., 2016b] Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., and Charnois, T. (2016b). A global constraint for mining sequential patterns with GAP constraint. In Quimper, C., editor, *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings*, volume 9676 of *Lecture Notes in Computer Science*, pages 198–215. Springer.
- [Khiari et al., 2010] Khiari, M., Boizumault, P., and Crémilleux, B. (2010). Constraint programming for mining n-ary patterns. In *CP'10*, volume 6308 of *LNCS*, pages 552–567. Springer.
- [Lazaar et al., 2016] Lazaar, N., Lebbah, Y., Loudni, S., Maamar, M., Lemière, V., Bessiere, C., and Boizumault, P. (2016). A global constraint for closed frequent pattern mining. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 333–349. Springer.
- [Maamar et al., 2015] Maamar, M., Lazaar, N., Loudni, S., and Lebbah, Y. (2015). Localisation de fautes à l'aide de la fouille de données sous contraintes. In *COSI'2015*.
- [Maimon and Rokach, 2005] Maimon, O. and Rokach, L. (2005). *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- [Pei et al., 2001] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. (2001). PrefixSpan : Mining sequential patterns by prefix-projected growth. In *ICDE*, pages 215–224. IEEE Computer Society.
- [Pei et al., 2002] Pei, J., Han, J., and Wang, W. (2002). Mining sequential patterns with constraints in large databases. In *CIKM'02*, pages 18–25. ACM.
- [Preux, 2006] Preux, P. (2006). Notes de cours de fouille de données. Technical report, Support de cours, Université de Lille 3.
- [Quinlan, 1987] Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3) :221–234.
- [Read and Corneil, 1977] Read, R. C. and Corneil, D. G. (1977). The graph isomorphism disease. *Journal of Graph Theory*, 1 :339–363.
- [Tan et al., 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [Uno et al., 2004] Uno, T., Asai, T., Uchida, Y., and Arimura, H. (2004). An efficient algorithm for enumerating closed patterns in transaction databases. In *DS 2004*, pages 16–31.
- [Wiering and van Otterlo, 2014] Wiering, M. and van Otterlo, M. (2014). *Reinforcement Learning : State-of-the-Art*. Springer Publishing Company, Incorporated.
- [Zaki, 2000] Zaki, M. J. (2000). Sequence mining in categorical domains : Incorporating constraints. In *Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 6-11, 2000*, pages 422–429.