PROJET :
THÉORIE DE L'INFORMATION

Rédigé par :

Abdou DIENG

David CAKPOSSE

Mamadou Saliou TOURE

Ousmane BARRY

Sous la direction de

Dr Philippe BERTHET

Année Académique : 2024-2025

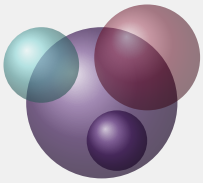


Table des matières

1 Introduction **1**

2 Codage **2**
2.1 Généralités . . . 2
2.2 Inégalité de Kraft . . . 4
2.3 Codage de Huffman . . . 6
2.3.1 Définition . . . 6
2.3.2 Propriétés . . . 6
2.3.3 Applications et domaines d'utilisation . . . 9
2.4 Autres codages . . . 10
2.4.1 Codage Arithmétique . . . 10
2.4.2 Codage de Shannon-Fano . . . 12

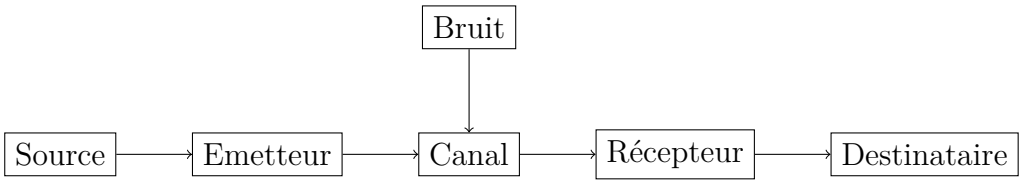
3 Entropie **13**
3.1 Définition . . . 13
3.1.1 Cas discret . . . 13
3.1.2 Cas continu . . . 14
3.2 Interprétation . . . 14
3.3 Lien entre entropie, tirage aléatoire et codage . . . 15
3.3.1 Lien avec la longueur minimale d'un code . . . 15
3.3.2 Issue d'un tirage aléatoire . . . 16
3.4 Maximum d'entropie . . . 17
3.5 Entropie minimum et maximum de vraisemblance . . . 18
3.5.1 Entropie minimum . . . 18
3.5.2 Estimation par maximum de vraisemblance . . . 19
3.5.3 Lien avec la divergence de Kullback-Leibler (entropie relative) . . . 19
3.5.4 Notion de contraste et distances entre distributions . . . 20
3.5.5 Application au codage . . . 20
3.5.6 Schéma récapitulatif . . . 20
3.6 Implementation, Visualisation et Interprétation des résultats . . . 20

4 Conclusion **23**

5 Références Bibliographiques **24**

Introduction

Dans un contexte où la quantité de données générées et traitées ne cesse d'exploser, la maîtrise des techniques de transmission et de traitement de l'information est devenue cruciale. La théorie mathématique de la communication, initiée par Claude Shannon en 1948, fournit un cadre rigoureux pour étudier dans quelles conditions un message peut être transmis de manière rapide et fiable, même en présence de perturbations (le bruit).



Le modèle shannonnien décrit un système de communication par une succession d'étapes : une **source** qui détient un message, un **émetteur** qui le transforme en signal, un **canal** – souvent sujet aux perturbations – par lequel le signal est transmis, et enfin un **récepteur** qui reconstruit l'information destinée au **destinataire** . La mission centrale de la théorie est alors d'étudier comment transmettre un message fidèle malgré les imperfections du canal. Deux questions majeures guident cette réflexion :

1. **Quelle est la capacité maximale d'un canal à transmettre de l'information ?**
2. **Comment assurer une transmission fiable, même en présence de bruit ?**

Une réponse essentielle à ces questions réside dans la notion d'**entropie**, mesure probabiliste de l'incertitude ou de la quantité d'information contenue dans un message. L'entropie permet d'évaluer les possibilités de **compression** sans perte, en exploitant la redondance des symboles dans les messages. Elle est également à la base des méthodes de **codage correcteur d'erreurs**, qui permettent de détecter et corriger les altérations introduites lors de la transmission.
Nous nous intéressons ici principalement aux problématiques de compression et de codage, deux applications majeures de la théorie de l'information, qui seront développées dans les chapitres consacrés à l'entropie et aux algorithmes de codage. À l'aide d'outils tels que les distributions de probabilité et les notions d'entropie, nous verrons comment mesurer l'incertitude d'un message et concevoir des codes efficaces, comme le codage de Huffman, le code de Shannon-Fano ou encore le codage arithmétique.
Ces approches seront approfondies à travers :

— l'étude de l'inégalité de Kraft et son lien avec les codes préfixes,

— la définition, les propriétés et les algorithmes associés aux différents types de codage,

— une comparaison des performances et de l'efficacité de ces méthodes.

Nous inclurons également une implémentation en Python permettant d'illustrer la construction et l'efficacité des codes sur des exemples simples.

Codage

2.1 Généralités

• Notion de codage

Le codage de l'information consiste à associer à chaque symbole d'un alphabet source un mot d'un alphabet cible (souvent binaire). On cherche des codages qui soient :

— non ambigus ;

— efficaces (en longueur moyenne notamment).

• Codage instantané ou sans perte

Soit S un ensemble fini ou dénombrable que l'on appellera *alphabet*. Ses éléments sont appelés *lettres* ou *symboles*, et une suite finie de lettres est appelée un *mot*. Pour tout entier $n \geq 1$, on définit naturellement

$$S^n = \underbrace{S \times S \times \cdots \times S}_{n \text{ fois}},$$

qui est exactement le produit cartésien (ou espace produit) de n copies de l'alphabet S.
Notation. On notera S^* l'ensemble des mots (ou chaînes finies de lettres) formés à partir de l'alphabet S :

$$S^* = \bigcup_{n=1}^{\infty} S^n.$$

On peut bien sûr concaténer deux mots pour former un nouveau mot plus long, et on notera $\cdot$ l'opération de concaténation :

$$(S^* \cup \{\emptyset\})^2 \to S^* \cup \{\emptyset\},$$

où $\emptyset$ désigne le *mot vide*.
Nous noterons $\ell(w)$ la *longueur* d'un mot w : si $n \in \mathbb{N}^*$, alors

$$\ell(w) = n \quad \Longleftrightarrow \quad w \in S^n.$$

Par extension, on pose $\ell(\emptyset) = 0$, et on a la relation suivante :

$$\ell(w \cdot w') = \ell(w) + \ell(w').$$

• Code

Définition : Un **code** est une application injective d'un alphabet source $S = \{s_1, s_2, \ldots, s_n\}$ vers l'ensemble des mots finis sur un alphabet cible (souvent $\{0,1\}^*$) :

$$C : S \to \{0,1\}^*$$

Chaque image $C(s_i)$ est appelée *mot de code*.

• **Code préfixe**

Un code est dit **préfixe** si aucun mot de code n'est préfixe d'un autre :

$$\forall i \neq j,\ C(s_i) \not\prec C(s_j)$$

où $u \prec v$ signifie que u est un *préfixe strict* de v c'est à dire $\exists w \in S^*,\ w \neq \emptyset\ \big(v = u\,w\big)$.

• **Code instantané**

Un **code instantané** est un code préfixe permettant un décodage immédiat, symbole par symbole, sans retour arrière.

Théeorème 2.1.0.1. Un code est instantané si et seulement s'il est préfixe.

Démonstration.
($\Rightarrow$) Un code instantané est préfixe : Supposons que C soit instantané mais non préfixe. Alors il existe $a, b \in S$, $a \neq b$, tels que $C(a)$ est un préfixe strict de $C(b)$. Soit $C(b) = C(a)z$, avec $z \in S^*, z \neq \emptyset$. Alors, lorsqu'on lit $C(b)$, on commence par lire $C(a)$, ce qui rend le décodage ambigu : on ne sait pas si le mot est $C(a)$ seul ou s'il continue. Cela contredit l'instantanéité. Donc, un code instantané est nécessairement préfixe.

($\Leftarrow$) Un code préfixe est instantané :
Supposons maintenant que C est préfixe. Alors, aucun code-mot n'est un préfixe d'un autre. Ainsi, lorsqu'on lit un mot dans l'image de C, dès qu'un code-mot est lu, il correspond à un seul symbole de S. On peut donc le décoder immédiatement, sans regarder la suite. En répétant ce procédé, on peut décoder toute séquence encodée de manière *instantanée*.

□

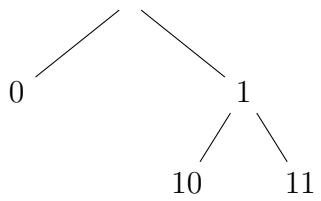
Représentation par arbre binaire
Un code instantané peut être représenté par un **arbre binaire**, où :

- Chaque arête est étiquetée par 0 (gauche) ou 1 (droite),
- Chaque mot de code correspond à un chemin de la racine à une feuille,
- Aucun mot n'est préfixe d'un autre : chaque mot est une **feuille**.

Un code comme suit :

$$\{0, 10, 11\}$$

est représenté par :



Longueur d'un arbre binaire :
On appelle longueur (ou hauteur) d'un arbre binaire T le nombre de nœuds (ou, selon la convention, d'arêtes) sur le plus long chemin partant de la racine et aboutissant à une feuille. On la note souvent $h(T)$ et on la définit récursivement par :

$$h(T) = \begin{cases} 0, & \text{si } T \text{ est l'arbre vide,} \\ 1 + \max\Big(h(T_{\text{gauche}}),\ h(T_{\text{droit}})\Big), & \text{sinon,} \end{cases}$$

où T_{gauche} et T_{droit} sont respectivement les sous-arbres gauche et droit de T.

2.2 Inégalité de Kraft

Théeorème 2.2.0.1 (Inégalité de Kraft)**.**

(i) Pour tout code préfixe $\{c_i\}_{i\geq 1}$, avec longueurs $\{\ell_i\}_{i\geq 1}$, on a :

$$\sum_i 2^{-\ell_i} \leq 1 \tag{1}$$

(ii) Réciproquement, si $\{\ell_i\}$ satisfait (1), alors il existe un code préfixe avec ces longueurs.

Preuve du (i). Soit $\{c_i\}$ un code préfixe, où c_i est un mot de code de longueur $\ell_i = |c_i|$. On définit une fonction $f : c_i \to [0, 1]$ qui calcule la valeur décimale de c_i, par :

$$f(c_i) = \sum_{j=1}^{\ell_i} c_{i,j} \cdot 2^{-j}$$

Pour référence ultérieure, on examine l'intervalle :

$$\Big[f(c_i), f(c_i) + 2^{-\ell_i}\Big]$$

Notons que :

1. $0 \leq f(c_i) \leq 1$.

2. $f(c_i 000\ldots 0) = f(c_i)$, c'est-à-dire que l'ajout de zéros à la fin du mot de code ne change pas la valeur de $f(c_i)$.

3. $f(c_i 111\ldots) = f(c_i) + \sum_{j=1}^{\infty} 2^{-(\ell_i+j)} = f(c_i) + 2^{-\ell_i}$. Cela découle de :

$$q^n - 1^n = \Big(1 + q + q^2 + \ldots + q^{n-1}\Big)\,(q-1),$$

ce qui donne :

$$1 + q + q^2 + q^3 + \ldots + q^n = \frac{q^{n+1} - 1}{q - 1}.$$

Sans perte de généralité, on peut supposer que les c_i sont ordonnés par ordre lexicographique croissant, ce qui signifie que $f(c_i) \leq f(c_k)$ pour tout $i \leq k$.
Comme $\{c_i\}$ est un code préfixe, on a :

$$f(c_{i+1}) \geq f(c_i) + 2^{-\ell_i} \tag{2}$$

Ainsi, les intervalles

$$\left[f(c_i), f(c_i) + 2^{-\ell_i}\right]$$

sont deux à deux disjoints.
Par application récurrente de l'inégalité (2), on obtient :

$$f(c_m) \geq \sum_{i=1}^{m} 2^{-\ell_i}$$

Puisque par définition $f(c_m) \leq 1$, cela prouve la première partie du théorème :

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$$

On a vu qu'une condition nécessaire pour qu'un code $\{c_i\}$ soit préfixe est que les intervalles

$$\left[f(c_i), f(c_i) + 2^{-\ell_i}\right]$$

soient deux à deux disjoints. □

La preuve de la deuxième partie repose sur l'affirmation que cette condition est aussi suffisante :

Lemme 1 : Étant donné un code $\{c_i\}$ tel que les intervalles

$$\left[f(c_i), f(c_i) + 2^{-\ell_i}\right]$$

soient disjoints, alors le code est préfixe.

Remarque : Dans la preuve suivante, nous utilisons le fait que pour prouver $A \Rightarrow B$, on peut montrer que $\neg B \Rightarrow \neg A$.

Démonstration. Supposons par contraposée que le code $\{c_i\}$ n'est pas préfixe. Alors on peut trouver deux mots de code c_m et c_n (sans perte de généralité $m > n$ donc $\ell_m > \ell_n$), pour lesquels les $|c_n|$ premiers bits de c_m sont identiques à ceux de c_n. Dans ce cas :

$$f(c_m) = \sum_{j=1}^{\ell_m} c_{m,j} \cdot 2^{-j} = \sum_{j=1}^{\ell_n} c_{n,j} \cdot 2^{-j} + \sum_{j=\ell_n+1}^{\ell_m} c_{m,j} \cdot 2^{-j} < f(c_n) + 2^{-\ell_n}$$

Ainsi, on a $f(c_m) < f(c_n) + 2^{-\ell_n}$, ce qui contredit le fait que les intervalles

$$\left[f(c_n), f(c_n) + 2^{-\ell_n}\right] \quad \text{et} \quad \left[f(c_m), f(c_m) + 2^{-\ell_m}\right]$$

soient disjoints. Donc le code est préfixe. □

Preuve du (ii). Supposons que les longueurs $\{\ell_i\}$ sont données et satisfont l'inégalité de Kraft (1). Nous prouvons qu'il est possible de construire un code préfixe avec ces longueurs. Sans perte de généralité, supposons que $\ell_1 \leq \ell_2 \leq \ldots$.
On définit le mot c_i comme l'image réciproque, par l'application f, du nombre :

$$\sum_{j=1}^{i-1} 2^{-\ell_j}$$

C'est-à-dire que c_i est le seul mot (à des zéros près à droite) tel que :

$$f(c_i) = \sum_{j=1}^{i-1} 2^{-\ell_j}$$

Pour calculer c_i, on utilise la fonction inverse $f^{-1} : [0,1] \to c_i$. Pour justifier cette utilisation, on montre d'abord que $0 < f(c_i) \leq 1$.
Par la structure de $f(c_i)$, on voit facilement que $f(c_i) > 0$ pour tout i. De plus, en utilisant l'hypothèse du théorème (i.e. l'inégalité (1)), on a :

$$f(c_i) = \sum_{j=1}^{i-1} 2^{-\ell_j} \leq 1$$

Donc $0 < f(c_i) \leq 1$.
On montre ensuite que la longueur de chaque mot de code c_i ainsi construit ne dépasse pas l_i.
Encore une fois, selon la structure de $f(c_i)$, le nombre maximal de bits nécessaires est l_{i-1} bits. Comme on suppose que les longueurs sont ordonnées par ordre croissant ($l_{i-1} \leq l_i$ pour tout i), la longueur de chaque mot de code c_i ne peut excéder $|c_i| = l_i$. Si elle est plus courte, on ajoute des zéros à droite jusqu'à atteindre la longueur désirée.
Pour conclure la preuve, il suffit de montrer que les intervalles :

$$I_i = \left[f(c_i), f(c_i) + 2^{-\ell_i}\right] = \left[\sum_{j=1}^{i-1} 2^{-\ell_j}, \sum_{j=1}^{i} 2^{-\ell_j}\right]$$

sont deux à deux disjoints, puis d'utiliser le Lemme 1.
Comme par définition $f(c_i)$ augmente avec i et que la borne droite de l'intervalle I_i est la borne gauche de l'intervalle I_{i+1}, les intervalles $\{I_i\}$ sont deux à deux disjoints, ce qui conclut la preuve. □

2.3 Codage de Huffman

2.3.1 Définition

Le codage de Huffman est un algorithme de compression de données sans perte. Le codage de Huffman utilise un code à longueur variable pour représenter un symbole de la source (par exemple un caractère dans un fichier). Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents.

Un code de Huffman est optimal au sens de la plus courte longueur pour un codage par symbole, et une distribution de probabilité connue. Des méthodes plus complexes réalisant une modélisation probabiliste de la source permettent d'obtenir de meilleurs ratios de compression.

Il a été inventé par David Albert Huffman, et publié en 1952.

2.3.2 Propriétés

Principe du codage de Huffman

Le principe du codage de Huffman repose sur la construction d'une structure d'arbre avec des nœuds.

Dans cet algorithme, on se base sur l'idée selon laquelle la meilleure manière de coder une information est de trouver la longueur optimale pour représenter chaque caractère de la séquence initiale. La longueur optimale du code correspondant à un caractère est donnée par le nombre de branches qu'il faudra parcourir partant du sommet de l'arbre de Huffman ou Huffman Tree en anglais pour arriver à ce caractère qui sera toujours une feuille de la Huffman Tree.
L'idée derrière l'arbre de Huffman est de minimiser la longueur des symboles qu'on utilise pour coder un message et ceci revient en quelque sorte à minimiser la taille de cet arbre. On pourrait cependant se demander comment s'effectue la construction de l'arbre de Huffman.

Construction de l'arbre de Huffman
Pour ce qui est de la construction de l'arbre, on associe chaque fois les deux nœuds de plus faibles poids, pour donner un nouveau nœud « père » dont le poids est équivalent à la somme des poids de ses fils. On réitère ce processus jusqu'à n'en avoir plus qu'un seul nœud : la racine de l'arbre de Huffman qui aura pour poids la somme des poids de tous les nœuds feuilles initiaux. On associe ensuite par exemple le code 0 à chaque embranchement partant vers la gauche et le code 1 vers la droite. Pour obtenir le code binaire de chaque caractère, on effectue un parcours de l'arbre allant du sommet jusqu'aux feuilles en prenant à chaque fois le caractère 0 ou 1 des branches que l'on traverse selon qu'on passe par la droite ou par la gauche.

Les différentes étapes de la construction de la Huffman Tree

1. Construire des nœuds feuilles à partir de chaque caractère du message à coder, chaque caractère étant pris en compte une seule fois avec les différentes fréquences d'apparition des caractères à ranger par ordre de priorité, les prioritaires étant ceux de plus faibles fréquences.
2. Extraire les deux noeuds de plus faibles poids par ordre croissant de fréquence c'est-à-dire le noeud ayant la fréquence la plus faible doit être extrait en premier.
3. Créer un nouveau nœud interne avec une fréquence égale à la somme des fréquences de ces deux nœuds. Définir le premier nœud extrait comme le fils gauche de ce nœud et le second nœud comme le fils droit. Ajouter le nouveau noeud créé à la liste des autres noeuds restants.
4. Répéter les étapes 2 et 3 jusqu'à être passé(e) par tous les nœuds obtenus à l'étape 1. Le nœud restant est la racine de l'arbre et la construction est terminée.

Un exemple pour illustrer le codage
Pour illustrer l'idée du codage de Huffman, nous allons considérer le message ci-dessous que nous allons représenter de manière optimale sous forme binaire :

$$aabbccddbbeaebdddfffdbfffddabbbbbcdefaabbccccccaabbddfffdcecc$$

Pour un message à coder comme celui-ci, commençons d'abord par comptabiliser la fréquence des caractères distincts dans le message. Après comptabilisation, on obtient le résultat suivant : a : 8, b : 15, c : 11, d : 12, e : 4, f : 9
Nous allons implémenter l'algorithme du codage de Huffman sur cet exemple.
La première étape consiste à utiliser les caractères les moins fréquents dans le message à coder. Il faudra donc commencer avec les lettres 'e' et 'a' qui ont les fréquences les plus faibles, puis sommer leurs poids c'est-à-dire sommer leurs fréquences d'apparition afin de remplacer les deux caractères par un nouveau caractère unique qui aura pour fréquence la somme des fréquences de ces deux caractères

dans la liste. Au terme de cette première étape, la liste devient :
new node : 12, b : 15, c : 11, d : 12, f : 9
où le nœud new node remplace les nœuds a et e avec un poids égal à la somme de leurs poids respectifs.
[new node : 12 = (a :8, e :4)]
Ensuite, il faudra choisir dans la nouvelle liste obtenue les deux éléments ayant les plus petits poids, puis itérer le processus jusqu'à obtenir le nœud de poids maximal c'est-à-dire ayant pour poids la somme des fréquences de tous les caractères. On remarque qu'à la fin de la construction de l'arbre de Huffman, tous les caractères distincts du message initial à coder sont des feuilles de l'arbre de Huffman, encore appelés Leaf Nodes en anglais. On obtient les résultats ci-dessous au terme de l'exécution :

Symbole	**Code binaire**
d	00
e	010
a	011
b	10
f	110
c	111

La complexité temporelle de cet algorithme est quasi-linéaire c'est-à-dire en O(nlogn) où n est le nombre de caractères uniques de la liste à compresser et la complexité spatiale est linéaire donc en O(n)

Propriétés d'optimalité

Considérons une source de symboles S, qui est susceptible de produire n symboles distincts $s_1 \ldots, s_n$; vous pouvez imaginer pour simplifier que cette source de symboles est un fichier, ou un canal de communication. Chaque symbole s_i $(1 \leq i \leq n)$ a une probabilité p_i d'occurrence sur la source S.
Un code C associe un mot binaire à chaque symbole s_i que peut produire la source : notons $C(s_i)$ le mot binaire associé au symbole s_i, et $\ell(C(s_i))$ sa longueur. L'espérance de la longueur du code C est donnée par :

$$L(C) = \sum_{i=1}^{n} p_i \cdot \ell(C(s_i))$$

Notons C^* le code de Huffman que l'on peut déduire à partir des probabilités d'occurrences $p_1, \ldots, p_n$.
Le code de Huffman est un code optimal au sens où, pour tout autre code C, on a :

$$L(C^*) \leq L(C)$$

Preuve de la propiété d'optimalité

Nous supposons dans cette preuve que le code est binaire $(q = 2)$ pour alléger les notations, mais la démonstration se généralise naturellement à un alphabet de taille $q \geq 2$.
La preuve repose sur un raisonnement par récurrence sur le nombre de symboles n.

Cas de base : $n = 1$

Si la source ne produit qu'un seul symbole s_1, alors $p_1 = 1$ et le seul mot que l'on peut lui associer est le mot vide. Ainsi :

$$L(C^*) = 0 = L(C).$$

Le résultat est donc trivialement vrai.

Étape de récurrence : $n \geq 2$

Soit C un code instantané optimal pour les symboles $s_1, \ldots, s_n$, représenté par un arbre binaire A. Soit C^* le code de Huffman correspondant, représenté par un arbre H.

— Dans tout arbre optimal A, il n'existe pas de nœud avec un seul fils contenant des feuilles, car on pourrait supprimer ce nœud pour réduire la longueur moyenne.

— Si $p_i < p_j$, alors on a nécessairement $\ell(C(s_i)) \geq \ell(C(s_j))$, car échanger ces deux symboles réduirait la longueur moyenne, ce qui contredirait l'optimalité.

— On peut supposer que les deux symboles de plus faibles probabilités s_1 et s_2 sont codés par deux feuilles sœurs (ayant un parent commun) dans A.

On procède alors comme dans la construction du code de Huffman :

1. On fusionne s_1 et s_2 en un nouveau symbole s' de probabilité $p' = p_1 + p_2$.
2. On considère la nouvelle source réduite $S' = \{s', s_3, \ldots, s_n\}$.
3. Le code C induit un code C' sur S', où les longueurs des mots associés à $s_3, \ldots, s_n$ restent inchangées, et s' hérite d'une longueur réduite d'un cran par rapport à s_1 et s_2.
4. Soit C'^* le code de Huffman sur S'. Par hypothèse de récurrence, on a :
$$L(C'^*) \leq L(C').$$
5. En « décompressant » les deux feuilles sœurs s_1 et s_2, on a :
$$L(C) = L(C') + p_1 + p_2, \quad L(C^*) = L(C'^*) + p_1 + p_2.$$
6. Ainsi :
$$L(C^*) = L(C'^*) + p_1 + p_2 \leq L(C') + p_1 + p_2 = L(C).$$

Conclusion
Par récurrence, on a montré que pour tout code instantané C, le code de Huffman C^* vérifie :

$$\boxed{L(C^*) \leq L(C)}.$$

Ce qui conclut la preuve de l'optimalité du code de Huffman.

2.3.3 Applications et domaines d'utilisation

Le codage de Huffman, au-delà de sa rigueur mathématique et de son efficacité théorique, trouve de nombreuses applications dans des domaines variés. Sa simplicité de mise en œuvre et son adaptabilité à différentes distributions en font un outil central de la compression et de la transmission de données.

Compression de fichiers et multimédia : Le codage de Huffman est utilisé dans plusieurs standards de compression, tels que les formats `ZIP`, `PNG`, `MP3` ou encore `JPEG`. Il permet de représenter les données de manière plus compacte en attribuant des codes courts aux symboles fréquents. Cette approche est idéale pour le stockage ou la transmission efficace de données numériques.

Télécommunications : Dans les systèmes de communication, notamment ceux soumis à des contraintes de bande passante, Huffman permet de réduire la taille des messages transmis. L'économie réalisée en bits transmis a un impact direct sur la vitesse et la fiabilité de la transmission, tout en assurant une restitution exacte du message.

Bioinformatique : Les séquences génétiques, telles que l'ADN, présentent souvent des redondances dans la succession de bases nucléotidiques. Le codage de Huffman est utilisé pour compresser ces séquences, en tirant parti des fréquences inégales des motifs dans les données biologiques. Cela permet de stocker ou d'analyser de grandes bases de données génomiques plus efficacement.

Traitement du langage naturel (NLP) : Dans les modèles de traitement du langage tels que `Word2Vec`, une version hiérarchique du *softmax* fait appel à un arbre de Huffman pour accélérer les calculs de probabilité. Cela permet une réduction significative du temps de traitement, tout en maintenant une précision élevée dans les tâches de modélisation sémantique.

Apprentissage statistique et théorie de l'information : Le codage de Huffman s'intègre naturellement dans le cadre du principe de longueur minimale de description (MDL), qui vise à représenter un modèle ou des données avec le moins d'information possible. Il est aussi lié à la notion d'entropie, en fournissant un code dont la longueur moyenne approche la borne théorique.

Systèmes embarqués et Internet des Objets (IoT) : Les capteurs de l'Internet des Objets produisent souvent des flux de données redondants. Huffman permet de les compresser localement, avant transmission, réduisant ainsi l'énergie consommée et la charge réseau. Son implémentation légère le rend adapté aux microcontrôleurs et systèmes à ressources limitées.

Jeux vidéo et animation : Dans les jeux ou les environnements interactifs, le codage de Huffman est utilisé pour compresser les textures, sprites ou données sonores. Cela permet de gagner en mémoire et en vitesse de chargement, tout en maintenant la qualité perçue.

Ainsi, le codage de Huffman s'étend bien au-delà de la simple théorie, trouvant des applications concrètes dans des domaines technologiques, scientifiques et industriels. Sa pertinence repose sur sa capacité à traduire efficacement la fréquence d'occurrence des symboles en une représentation binaire compacte.

2.4 Autres codages

2.4.1 Codage Arithmétique

Définition

Le **codage arithmétique** est une méthode de compression sans perte qui encode une séquence entière de symboles en un seul nombre réel compris dans l'intervalle $[0, 1[$.
À la différence du codage de Huffman, le codage arithmétique code des séquences de valeurs et non des valeurs prises individuellement, ce qui représente une amélioration.
Par exemple, au lieu de coder chaque symbole (un octet, un nombre à virgule flottante, etc.), le codage

arithmétique remplace l'ensemble du message par un seul mot représentant un nombre compris entre 0 et 1.

Description de l'Algorithme

Supposons que nous ayons une source de symboles S avec un alphabet $\{s_1, s_2, \ldots, s_D\}$ et une fonction de probabilité $p(s_i)$ associée à chaque symbole.

On définit la fonction de répartition cumulative C par :

$$C(s_i) = \sum_{j=1}^{i-1} p(s_j), \quad \text{avec } C(s_1) = 0.$$

Ainsi, le symbole s_i correspond à l'intervalle :

$$\Big[\, C(s_i),\, C(s_i) + p(s_i) \,\Big[.$$

Désignons par :

- BI_k : Borne Inférieure de l'intervalle à l'étape k.
- BS_k : Borne Supérieure de l'intervalle à l'étape k.
- t : Taille de l'intervalle courant, définie par

$$t = BS_{k-1} - BI_{k-1}.$$

- x_k : Le symbole à la position k dans la séquence à coder.

Étape 1 : Initialisation
On définit les bornes initiales de l'intervalle comme suit :

$$BI_0 = 0, \quad BS_0 = 1.$$

Étape 2 : Traitement de chaque symbole
Pour chaque symbole x_k dans la séquence, l'algorithme procède comme suit :

1. Calcul de la taille de l'intervalle courant :

$$t = BS_{k-1} - BI_{k-1}.$$

2. Si $x_k = s_i$, alors l'intervalle est raffiné selon les formules :

$$BI_k = BI_{k-1} + t \times C(s_i),$$

$$BS_k = BI_{k-1} + t \times \Big(C(s_i) + p(s_i)\Big).$$

Étape 3 : Intervalle de compression
Après traitement de tous les symboles de la séquence (soit n symboles), l'intervalle final $[BI_n, BS_n[$ représente le code arithmétique associé à la séquence initiale. Un nombre choisi dans cet intervalle sert de représentation compressée de l'ensemble de la séquence.

2.4.2 Codage de Shannon-Fano

Définition

Le **codage de Shannon-Fano** est une méthode de *compression sans perte* qui attribue des codes binaires aux symboles d'un message en fonction de leur fréquence d'apparition. Plus un symbole est fréquent, plus son code sera court.

Principe de l'algorithme

1. Énumérer les symboles et leurs probabilités.
2. Trier les symboles par probabilité décroissante.
3. Diviser l'ensemble en deux sous-groupes ayant des sommes de probabilités aussi équilibrées que possible.
4. Attribuer `0` à un groupe, `1` à l'autre.
5. Répéter récursivement le processus pour chaque sous-groupe.

Exemple

Soient les symboles avec leurs probabilités :

Symbole	Probabilité
A	0.4
B	0.2
C	0.2
D	0.1
E	0.1

On peut obtenir les codes suivants :

Symbole	Code Shannon-Fano
A	00
B	01
C	10
D	110
E	111

Remarques

— Le code obtenu est un *code préfixe*, c'est-à-dire qu'aucun code n'est préfixe d'un autre.

— Le codage de Shannon-Fano est parfois moins optimal que le codage de Huffman. (voir la section 3.6)

— C'est une méthode intuitive et simple à implémenter.

Entropie

Dans le cadre de la théorie de l'information, l'entropie constitue la mesure fondamentale de l'incertitude associée à la production d'un message par une source. En quantifiant le degré d'imprévisibilité des symboles émis, l'entropie permet d'évaluer la quantité moyenne d'information contenue dans chaque message. Ce concept est crucial pour définir les limites théoriques de la compression sans perte et pour élaborer des stratégies de codage qui optimisent la transmission de données, même en présence de bruit et d'autres perturbations. En d'autres termes, l'entropie offre une vision précise de la diversité des messages et oriente le développement de techniques robustes pour assurer la fiabilité des communications.

3.1 Définition

Dans le contexte de la théorie de l'information, l'entropie $H(S)$ d'une source qui émet des symboles s_i avec des probabilités $p(s_i)$ est définie par :

$$H(S) = -\sum_i p(s_i) \log_b p(s_i)$$

où :

- $p(s_i)$ représente la probabilité d'apparition du symbole s_i,
- Le logarithme est habituellement en base 2 (ce qui donne une entropie exprimée en bits), mais d'autres bases (comme e pour les nats) peuvent aussi être utilisées.

par définition :

$$\log_b x = \frac{\ln x}{\ln b}.$$

En base e :

$$\log_e x = \ln x.$$

Mathématiquement, cela ne change pas la nature de l'entropie, seulement l'échelle (un facteur constant). Cette formule mesure l'incertitude moyenne associée à l'émission d'un symbole par la source.

3.1.1 Cas discret

Pour une source d'information discrète, qui émet un nombre fini ou dénombrable de symboles, l'entropie est calculée en sommant sur toutes les valeurs possibles. Pour une source avec D symboles, la formule s'écrit :

$$H(S) = -\sum_{i=1}^{D} p(s_i) \log_2 p(s_i)$$

Propriétés importantes dans le cas discret

— **Entropie maximale :**
Lorsque tous les symboles sont équiprobables, c'est-à-dire $p(_i) = \frac{1}{D}$ pour chaque i, l'entropie atteint son maximum :

$$H_{\max} = \log_2 D$$

Cette situation correspond à un maximum d'incertitude.

— **Additivité pour des sources indépendantes :**
Pour deux sources indépendantes S_1 et S_2, l'entropie du système conjoint est la somme des entropies individuelles :

$$H(S_1, S_2) = H(S_1) + H(S_2)$$

Ceci reflète le fait que l'incertitude totale résulte de l'addition des incertitudes propres à chaque source.

Ce cadre discret est particulièrement utilisé dans le domaine de la compression de données, où l'objectif est de réduire la redondance en exploitant précisément la distribution de probabilité des symboles.

3.1.2 Cas continu

Dans le cas d'une variable aléatoire continue, l'information est décrite par une fonction de densité de probabilité $f(s)$. L'équivalent de l'entropie dans ce cadre est l'entropie *différentielle*, définie par :

$$h(S) = -\int_{-\infty}^{+\infty} f(s) \ln f(s)\, ds$$

Points à noter pour le cas continu

— **Sensibilité aux échelles :**
Contrairement à l'entropie discrète, l'entropie différentielle peut prendre des valeurs négatives. En effet, elle ne représente pas directement une quantité de bits mais plutôt une densité d'information qui dépend de l'unité de mesure utilisée pour la variable S.

— **Transformation de variables :**
Sous une transformation affine, l'entropie différentielle change d'une quantité additive. Par exemple, pour une transformation $Y = aX + b$, on a :

$$h(Y) = h(X) + \ln |a|$$

Ceci traduit l'invariance relative de la notion d'incertitude sous les transformations linéaires.

L'entropie différentielle est utilisée dans des domaines comme le traitement du signal et l'estimation statistique, où les variables continues modélisent des phénomènes réels.

3.2 Interprétation

L'entropie dans la théorie de l'information peut être interprétée de plusieurs façons complémentaires :

Mesure de l'incertitude

L'entropie quantifie le degré d'incertitude associé à une source d'information. Plus l'entropie est élevée, moins il est possible de prédire quel symbole sera émis. Cette incertitude se traduit par la quantité d'information moyenne contenue dans chaque symbole émis.

Limite de compression

Dans le domaine de la compression, l'entropie fournit un seuil théorique : aucun algorithme de compression sans perte ne peut en moyenne coder les symboles d'une source en utilisant moins de $H(S)$ bits par symbole. C'est ce qu'on appelle la limite de Shannon.

Évaluation de la diversité

L'entropie permet d'évaluer la diversité ou l'hétérogénéité des messages produits par une source. Une entropie élevée signifie que la distribution des symboles est plus dispersée, tandis qu'une entropie faible suggère une prédominance de certains symboles.

Exemple en cas discret
Supposons qu'une source d'information émette quatre symboles $\{A, B, C, D\}$ avec les probabilités suivantes :

$$p(A) = 0.1, \quad p(B) = 0.2, \quad p(C) = 0.3, \quad p(D) = 0.4$$

L'entropie $H(X)$ de cette source (en utilisant la base 2) est alors :

$$\begin{aligned} H(X) &= -\Big[0.1\log_2(0.1) + 0.2\log_2(0.2) + 0.3\log_2(0.3) + 0.4\log_2(0.4)\Big] \\ &\approx -\Big[0.1 \times (-3.32) + 0.2 \times (-2.32) + 0.3 \times (-1.74) + 0.4 \times (-1.32)\Big] \\ &\approx 0.332 + 0.464 + 0.522 + 0.528 \\ &\approx 1.846 \text{ bits} \end{aligned}$$

Cet exemple montre que, même si le symbole D est le plus probable, la diversité des probabilités des symboles engendre une incertitude moyenne d'environ 1.85 bits par symbole.

Exemple en cas continu
Considérons une variable aléatoire S suivant une loi normale $\mathcal{N}(\mu, \sigma^2)$. L'entropie différentielle de X est donnée par :

$$h(S) = \frac{1}{2}\ln(2\pi e\sigma^2)$$

Cette expression met en évidence le rôle de l'écart-type σ dans la mesure de l'incertitude. Une distribution plus étalée (avec un σ plus grand) correspond à une entropie plus élevée.

3.3 Lien entre entropie, tirage aléatoire et codage

3.3.1 Lien avec la longueur minimale d'un code

On peut montrer que pour une source S d'entropie de Shannon $H(S)$, la longueur moyenne ℓ d'un mot de code obtenu par un codage de Huffman vérifie l'inégalité suivante :

$$H(S) \leq \ell < H(S) + 1$$

Cela signifie que le codage de Huffman permet d'approcher efficacement la longueur minimale théorique d'un code, laquelle est donnée par l'entropie de la source. Plus précisément, cette inégalité montre que la longueur moyenne d'un code de Huffman est toujours comprise entre l'entropie et l'entropie augmentée d'une unité. En ce sens, Huffman fournit un code proche de l'optimum.
Cependant, cette efficacité peut être limitée dans certains cas. En particulier, si l'entropie de la source est élevée, l'écart maximal d'un bit peut représenter un surcoût non négligeable en termes de compression. De plus, le codage de Huffman produit des longueurs de code entières pour chaque symbole, ce qui limite la possibilité d'optimisation fine par rapport à des méthodes comme le codage arithmétique, qui permettent des longueurs non entières moyennes.

Exemple

Soit une source S qui émet les symboles a, b, c avec les probabilités suivantes :

$$P(a) = 0{,}5, \quad P(b) = 0{,}3, \quad P(c) = 0{,}2$$

L'entropie est donnée par :

$$H(S) = -\left[0{,}5\log_2(0{,}5) + 0{,}3\log_2(0{,}3) + 0{,}2\log_2(0{,}2)\right] \approx 1{,}485$$

Un codage de Huffman peut donner :

$$a \to 0, \quad b \to 10, \quad c \to 11$$

La longueur moyenne du code est :

$$L = 0{,}5 \times 1 + 0{,}3 \times 2 + 0{,}2 \times 2 = 1{,}5$$

Ce qui respecte bien l'inégalité :

$$H(S) \leq \ell < H(S) + 1$$

3.3.2 Issue d'un tirage aléatoire

En théorie de l'information, chaque tirage d'une variable aléatoire correspond à une *issue* parmi plusieurs possibles. La quantité d'information transmise par ce tirage dépend directement de son *incertitude*, c'est-à-dire de l'**entropie** de la source. Plus le résultat du tirage est imprévisible, plus l'information qu'il contient est élevée.
Considérons une urne contenant quatre boules de couleurs différentes : rouge, bleue, jaune et verte. Si on tire une boule au hasard et que chaque couleur est *équiprobable*, alors aucune couleur n'est privilégiée. L'entropie est dans ce cas maximale, égale à :

$$H = \log_2(4) = 2 \text{ bits}$$

Cela signifie qu'il faut en moyenne 2 bits pour coder l'information correspondant à la couleur tirée. Par exemple, on peut attribuer les codes suivants : rouge $\to$ 00, bleue $\to$ 01, jaune $\to$ 10, verte $\to$ 11. En revanche, si la probabilité des différentes issues varie, alors l'entropie diminue. Supposons maintenant que l'urne contienne 4 boules rouges, 2 bleues, 1 jaune et 1 verte. Les probabilités sont alors :

— rouge : $p = \dfrac{4}{8} = 0{,}5$

— bleue : $p = \frac{2}{8} = 0{,}25$

— jaune : $p = \frac{1}{8} = 0{,}125$

— verte : $p = \frac{1}{8} = 0{,}125$

L'entropie de la source devient :

$$H = -\left(0{,}5\log_2(0{,}5) + 0{,}25\log_2(0{,}25) + 0{,}125\log_2(0{,}125) + 0{,}125\log_2(0{,}125)\right) \approx 1{,}75 \text{ bits}$$

Un codage optimal comme le codage de Huffman permet de refléter cette distribution. Par exemple :

— rouge → `0`

— bleue → `10`

— jaune → `110`

— verte → `111`

Avec ce code, la **longueur moyenne** d'un message (c'est-à-dire du code utilisé pour exprimer la couleur tirée) est :

$$L = 0{,}5 \times 1 + 0{,}25 \times 2 + 0{,}125 \times 3 + 0{,}125 \times 3 = \frac{7}{4} = 1{,}75 \text{ bits}$$

Cette valeur coïncide avec l'entropie calculée, ce qui montre que la **quantité d'information moyenne nécessaire pour déterminer l'issue d'un tirage aléatoire dépend directement de l'entropie** de la source. Plus la source est déséquilibrée (avec une issue dominante), moins il faut d'information en moyenne pour identifier l'issue.

Lien entre les notions d'entropie et de codage optimal

3.4 Maximum d'entropie

Le principe du maximum d'entropie, proposé par E.T. Jaynes (1957), stipule que lorsqu'on ne connaît qu'une information partielle sur une source (par exemple des contraintes sur des espérances), alors la meilleure estimation de la distribution sous-jacente est celle qui maximise l'entropie de Shannon tout en respectant ces contraintes. Cela permet de ne pas introduire d'information injustifiée.

Formulation

Soit $\{x_1, \ldots, x_n\}$ un ensemble fini. On cherche une loi de probabilité $p = (p_1, \ldots, p_n)$ telle que :

$$S(p) = -\sum_{j=1}^{n} p_j \log p_j$$

soit maximale, sous les contraintes suivantes :

— $\sum_{j=1}^{n} p_j = 1$ (normalisation)

— $\sum_{j=1}^{n} p_j f_i(x_j) = \mu_i, \quad \text{pour } i = 1, \ldots, k$ (contraintes sur les espérances)

On introduit alors la fonction de Lagrange :

$$\mathcal{L}(p, \lambda) = -\sum_{j=1}^{n} p_j \log p_j + \lambda_0 \left(1 - \sum_{j=1}^{n} p_j\right) + \sum_{i=1}^{k} \lambda_i \left(\mu_i - \sum_{j=1}^{n} p_j f_i(x_j)\right)$$

En annulant le gradient de $\mathcal{L}$ par rapport aux p_j, on obtient la distribution de maximum d'entropie :

$$p_j = \frac{1}{Z(\lambda)} \exp\left(-\sum_{i=1}^{k} \lambda_i f_i(x_j)\right)$$

où $Z(\lambda)$ est la fonction de partition :

$$Z(\lambda) = \sum_{j=1}^{n} \exp\left(-\sum_{i=1}^{k} \lambda_i f_i(x_j)\right)$$

Remarque : Cette forme est celle des **modèles exponentiels**, très utilisés en statistique et en apprentissage automatique.

Lemme de Gibbs

Lemme 1 (Gibbs). Soit $p = (p_1, \ldots, p_n)$ une distribution quelconque sur $\{x_1, \ldots, x_n\}$, et q une distribution exponentielle de la forme précédente. Alors :

$$S(p) \leq -\sum_{j=1}^{n} p_j \log q_j,$$

avec égalité si et seulement si $p = q$.
Ce lemme exprime que parmi toutes les distributions satisfaisant les contraintes, la distribution exponentielle est celle qui maximise l'entropie.

3.5 Entropie minimum et maximum de vraisemblance

Dans la théorie de l'information, l'entropie mesure l'incertitude ou la quantité d'information moyenne d'une source aléatoire. Lorsqu'on cherche à modéliser une source à partir de données, deux approches fondamentales émergent : **l'entropie minimale** et **le maximum de vraisemblance**. Ces deux concepts sont étroitement liés à la manière dont on estime les probabilités des symboles à partir d'observations, et ils jouent un rôle central dans la compression et le codage optimal.

3.5.1 Entropie minimum

L'entropie d'une source discrète de probabilité $P = (p_1, ..., p_n)$ est définie par :

$$H(P) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

L'entropie est minimale (égale à zéro) lorsque la source est totalement déterministe, c'est-à-dire lorsqu'un seul symbole a une probabilité 1 et tous les autres 0. Dans ce cas, il n'y a aucune incertitude : le message est parfaitement prévisible, et il n'y a donc aucune information à transmettre.
Cependant, dans la pratique, on cherche rarement à minimiser l'entropie, car cela reviendrait à ignorer la variabilité réelle de la source. Au contraire, on cherche à **modéliser au mieux la distribution réelle** des symboles, ce qui nous amène à la méthode du maximum de vraisemblance.

3.5.2 Estimation par maximum de vraisemblance

Supposons que l'on observe une séquence de symboles $x_1, ..., x_n$ provenant d'une source inconnue, et que l'on souhaite estimer la distribution de probabilité $P = (p_1, ..., p_m)$ qui a généré ces données.
La **vraisemblance** d'un modèle P vis-à-vis des données observées est donnée par :

$$L(P) = \prod_{i=1}^{n} P(x_i)$$

L'**estimation par maximum de vraisemblance** (EMV ou MLE en anglais) consiste à choisir la distribution $\hat{P}$ qui maximise cette vraisemblance, c'est-à-dire qui rend les données observées les plus probables sous le modèle choisi.
Dans le cas discret, l'estimateur du maximum de vraisemblance pour la probabilité d'un symbole a est simplement la fréquence observée de ce symbole dans l'échantillon :

$$\hat{p}_a = \frac{\text{nombre d'occurrences de } a}{n}$$

Exemple :
Si l'on observe la séquence suivante de 10 lettres : A, B, A, C, A, B, A, A, C, B, alors :

- $n_A = 5$, $n_B = 3$, $n_C = 2$,
- $\hat{p}_A = 0.5$, $\hat{p}_B = 0.3$, $\hat{p}_C = 0.2$.

Ces probabilités maximisent la vraisemblance des observations.

3.5.3 Lien avec la divergence de Kullback-Leibler (entropie relative)

La **divergence de Kullback-Leibler** (KL) est une mesure fondamentale de la différence entre deux distributions de probabilité P (la vraie distribution) et Q (un modèle) :

$$D_{KL}(P\|Q) = \sum_i P(i) \log_2 \frac{P(i)}{Q(i)}$$

Dans le contexte de l'estimation, P est la distribution empirique (celle observée dans les données), et Q est la distribution proposée par le modèle. **Maximiser la vraisemblance revient à minimiser la divergence de Kullback-Leibler entre la distribution empirique et le modèle.**
Autrement dit, l'estimateur du maximum de vraisemblance est celui qui rend le modèle Q aussi proche que possible des observations, au sens de la divergence KL.
Ce lien est fondamental en théorie de l'information :

- L'**entropie empirique** $H(P)$ mesure l'incertitude moyenne observée.
- La divergence KL $D_{KL}(P\|Q)$ mesure le surcoût d'information (en bits) si l'on code les données issues de P en utilisant le modèle Q au lieu de P.
- Le maximum de vraisemblance fournit le modèle Q qui minimise ce surcoût.

Exemple (surcoût en bits lors du codage). Supposons que l'on souhaite coder une source composée de deux symboles, A et B. La distribution empirique observée (la vraie) est $P = (p_A, p_B) = (0.8, 0.2)$, mais on utilise par erreur un modèle $Q = (q_A, q_B) = (0.5, 0.5)$ pour construire le code.
Calculons la divergence de Kullback-Leibler :

$$D_{KL}(P\|Q) = 0.8 \log_2 \left(\frac{0.8}{0.5}\right) + 0.2 \log_2 \left(\frac{0.2}{0.5}\right)$$

$$= 0.8 \log_2(1.6) + 0.2 \log_2(0.4)$$
$$\approx 0.8 \times 0.6781 + 0.2 \times (-1.3219)$$
$$\approx 0.5425 - 0.2644 = 0.2781 \text{ bits}$$

Cela signifie que, si l'on code les messages générés par P en utilisant le code optimal pour Q, la longueur moyenne du code sera supérieure de 0.2781 bits par symbole à la longueur minimale (celle donnée par l'entropie de P). Ce surcoût correspond exactement à la divergence $D_{KL}(P\|Q)$.

3.5.4 Notion de contraste et distances entre distributions

La divergence de Kullback-Leibler est un exemple de **contraste** : une fonction qui mesure l'écart entre deux distributions de probabilité. Dans le cas discret, la divergence KL s'identifie à l'**entropie relative de Shannon**. Il existe d'autres contrastes (distance de Hellinger, chi-carré, etc.), mais la KL est la plus utilisée en théorie de l'information et en statistique.

3.5.5 Application au codage

Dans la pratique, l'estimation par maximum de vraisemblance des probabilités permet de construire des codes optimaux (comme le code de Huffman) adaptés à la source observée.

— Si on utilise les fréquences observées pour estimer les probabilités, on obtient la longueur moyenne de code la plus proche possible de l'entropie empirique.

— Si on utilise un modèle inadapté, la longueur moyenne sera plus grande, ce qui correspond exactement à la divergence KL entre la distribution réelle et le modèle utilisé.

3.5.6 Schéma récapitulatif

$$\begin{array}{ccc} \text{Observations} \longrightarrow \text{Distribution empirique } P & & \\ & \downarrow & \\ \text{Maximum de vraisemblance} & \longrightarrow & \text{Modèle } Q \text{ (à ajuster)} \\ & & \uparrow \\ & & \text{Minimisation de } D_{KL}(P||Q) \end{array}$$

En résumé, le maximum de vraisemblance est la méthode naturelle pour estimer les probabilités d'une source à partir de données. Il permet de construire des codes adaptés, minimisant la longueur moyenne des messages, et il est intimement lié à la notion d'entropie relative (divergence de Kullback-Leibler), qui mesure la qualité de l'adaptation du modèle aux données.

3.6 Implementation, Visualisation et Interprétation des résultats

Alogrithme universel (Lempel-Ziv)

• Principe général

L'algorithme de Lempel-Ziv (LZ) est un algorithme universel de compression sans perte qui ne nécessite aucune connaissance préalable sur les probabilités d'apparition des symboles. Il existe deux

variantes principales :

— **LZ77 (Sliding Window)** : encode le texte en triplets (offset, longueur, caractère).

— **LZ78 (Dictionary-based)** : encode le texte en paires (index, caractère) à l'aide d'un dictionnaire construit dynamiquement.

• LZ77 : Fenêtre glissante

Principe : Pour chaque position du texte, on recherche la plus longue séquence déjà vue dans la fenêtre glissante précédente. On encode cette répétition avec un triplet :

— `offset` : la distance à rebours où commence la séquence similaire.

— `length` : la longueur de cette séquence.

— `next_char` : le caractère suivant.

Exemple : Soit le texte `abcabcabcabc`. L'encodage LZ77 peut produire :

(0,0,a), (0,0,b), (0,0,c), (3,3,a), ...

• LZ78 : Dictionnaire dynamique

Principe : On construit dynamiquement un dictionnaire de motifs déjà rencontrés. À chaque étape, on encode la plus longue séquence vue sous forme de paire :

— `index` : indice de la séquence connue la plus longue.

— `caractère` : caractère suivant.

Exemple : Pour le texte `ababc`, on obtient :

(0,a), (0,b), (1,b), (0,c)

• Avantages et comparaison

— **Universel** : pas besoin de connaître les probabilités.

— **Adaptatif** : s'adapte dynamiquement à la structure du message.

— **Utilisation** : PNG, ZIP, GIF, PDF.

Méthode	**Proba connue ?**	**Préfixe**	**Complexité**	**Avantages**
Huffman	Oui	Oui	$O(n \log n)$	Optimal par symbole
Arithmétique	Oui	Non	$O(n)$	Très proche de l'entropie
Lempel-Ziv	Non	Oui	$O(n)$	Universel, adaptatif

L'algorithme de Lempel-Ziv est un pilier de la compression moderne. Il est simple, rapide, efficace et s'adapte automatiquement aux données. Il est la base de nombreux formats industriels (ZIP, PNG, etc.) et représente une avancée majeure dans les techniques de codage universel.

Comparaisons des méthodes de codage

Cette étude vise à comparer quatre méthodes classiques de **compression sans perte** et le codage **universel** :

- Codage de Huffman
- Codage de Shannon-Fano
- Codage arithmétique
- Codage de Lempel-Ziv (LZ78)

Nous évaluons leurs performances selon quatre critères essentiels :

- Entropie du message ;
- Longueur moyenne du code ;
- Complexité algorithmique ;
- Sensibilité à la distribution des symboles ;

Nous travaillerons sur l'alphabet français étendu (juste un choix), composé des 26 lettres minuscules (`a` à `z`) et de l'espace, soit un total de 27 symboles. Ce choix reflète une base réaliste pour simuler des messages textuels.
Pour mettre en lumière les forces et limites de chaque méthode, nous considérons deux de types de situations réalistes :

Longueur des messages

- Message courts : entre 5 et 10 caractères.
- Messages longs : entre 500 et 1000 caractères.

Distribution des symboles

- Distribution uniforme : chaque caractère a une probabilité identique d'apparaître.
- Distribution naturelle : les caractères suivent les fréquences moyennes d'occurrence dans la langue française (ex. : le `e` est bien plus fréquent que le `z`). Cette configuration reflète un contexte réaliste, proche de celui des textes en français courant.

Le notebook complet contenant l'implémentation des codes, la comparaison, les graphiques et l'interprétation est disponible en ligne ici : implémentation

Conclusion

Ce projet nous a permis d'explorer les fondements de la théorie de l'information, en mettant l'accent sur les concepts clés que sont l'entropie et les techniques de codage. À travers l'étude de méthodes comme le codage de Huffman, Shannon-Fano, arithmétique et Lempel-Ziv, nous avons illustré comment l'optimisation de la longueur des codes permet une compression efficace des données, en lien direct avec la mesure d'incertitude fournie par l'entropie.
Les résultats théoriques, tels que l'inégalité de Kraft ou l'optimalité du codage de Huffman, ont été confrontés à des implémentations pratiques en Python, renforçant notre compréhension des mécanismes sous-jacents. Nous avons également observé que chaque méthode présente des avantages spécifiques : Huffman pour les codes préfixes optimaux, Lempel-Ziv pour son universalité, ou le codage arithmétique pour son approche probabiliste fine.
Enfin, ce travail a souligné l'importance de ces concepts dans des applications modernes, comme la compression de fichiers. Ces notions, bien qu'introduites dès les années 1950, restent incontournables dans la conception de systèmes efficaces de transmission et de stockage de l'information.
Ce projet constitue ainsi une introduction solide aux principes de la théorie de l'information, préparant à des études plus avancées en compression, cryptographie ou traitement du signal, tout en offrant des outils concrets pour aborder des problématiques actuelles en science des données.
En conclusion, ce projet nous a permis de comprendre les bases de la théorie de l'information et de voir comment ces concepts sont appliqués dans des méthodes de codage et de compression de données. Les algorithmes étudiés, comme le codage de Huffman et le codage arithmétique, montrent comment on peut optimiser la transmission et le stockage de l'information en utilisant des techniques de codage efficaces. Les implémentations en Python ont permis de voir comment ces algorithmes fonctionnent en pratique et comment ils peuvent être utilisés pour compresser des données de manière efficace.
De plus, l'étude de l'entropie nous a montré comment mesurer l'incertitude et la quantité d'information dans un message, ce qui est crucial pour comprendre les limites théoriques de la compression de données. Les applications pratiques de ces concepts, comme dans la compression de fichiers et la transmission de données, montrent leur pertinence dans le monde moderne.
En somme, ce projet a été une excellente introduction à la théorie de l'information et a fourni une base solide pour des études plus approfondies dans ce domaine. Les compétences acquises et les outils développés seront utiles pour aborder des problématiques plus complexes en science des données et en traitement de l'information.

Bibliographie

[1] https://fr.wikipedia.org/wiki/Codage_de_Huffman

[2] https://fr.wikipedia.org/wiki/Entropie_de_Shannon

[3] https://fr.wikipedia.org/wiki/Codage_de_Shannon-Fano

[4] https://fr.wikipedia.org/wiki/Codage_arithm%C3%A9tique

[5] https://fr.wikipedia.org/wiki/LZ77_et_LZ78

[6] https://fr.wikipedia.org/wiki/In%C3%A9galit%C3%A9_de_Kraft

[7] https://fr.wikipedia.org/wiki/Th%C3%A9orie_de_l%27information

[8] Philippe BERTHET. Extrait du document *Modèle statistique paramètrique*, Niveau Master, Université de Toulouse
Vraisemblance et exhaustivité - ici
Contraste de Küllback-Leibler - ici

[9] Dominique Bontemps. *Introduction à la théorie des probabilités*. Cours Math2-Prob1 (FSI - KMAXIP01), Licence 2 Mathématiques, Université Paul Sabatier Toulouse III, 2023–2024. Disponible sur : ici.

[10] Jean-Guillaume Dumas, Jean-Louis Roch, Éric Tannier, et Sébastien Varrette. *Théorie des codes : compression, cryptage, correction. Cours et exercices avec solutions*. Dunod, 2009. Niveau Master et Écoles d'ingénieurs.