M1Info

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Université Paris 8 - Vincennes à Saint-Denis

M1 MIASHS : Big Data et fouille de données

A Clonal Selection Algorithm with Levenshtein Distance based Image Similarity in Multidimensional Subjective Tourist Information and Discovery of Cryptic Spots by Interactive GHSOM

PANCHALINGAMOORTHY Gajenthran

Organisme d'accueil :	Université Paris 8
Course :	Technique d'apprentissage

Table des matières

Chapitre 1

Introduction

Les avancées technologiques permettent de transmettre les informations de manière quasi-instantanée. Cette circulation d'informations offrent de nombreux avantages notamment pour le tourisme : cela aide grandement à l'expérience des touristes. L'article "A Clonal Selection Algorithm with Levenshtein Distance based Image Similarity in Multidimensional Subjective Tourist Information and Discovery of Cryptic Spots by Interactive GHSOM" écrit par Takumi Ichimura et Shin Kamada [TI18a], décrit un ensemble de méthodes facilant les recherches des touristes sur un lieu donné, à l'aide de MPPS (Mobile Phone based Participatory Sensing). En effet, le MPPS est un système s'appuyant sur la récolte d'informations d'un lieu provenant des utilisateurs. Ces informations qui viennent essentiellement des téléphones peuvent avoir plusieurs formes : nous pouvons avoir des informations audios, des informations visuelles ou encore des informations textuelles.
Plus de 500 données ont été récoltées par l'application Androïd [Map11] de Takumi Ichimura et Shin Kamada pour réaliser des classifications et offrir des résultats convenables aux visiteurs, à savoir des photos des régions les plus touristiques ou les plus pertinentes du lieu en question. Ces classifications se basent notamment sur l'Interactive GHSOM [TI18d] pour les informations textuelles d'une part et d'autre part, sur la distance de Levenshtein [MG] et le modèle CSAIM [TI18b] (Clonal Selection Algorithm with Immunological Memory) pour les informations visuelles (les informations audios ne seront pas traitées).
Dans ce papier, nous étudierons dans un premier temps l'ensemble des données pris en compte pour l'apprentissage, puis nous nous concentrerons sur la phase d'apprentissage qui se découpe en 3 parties avec l'Interactive GHSOM, la distance de Levenshtein et le modèle CSAIM, et enfin nous testerons l'efficacité de ces modèles à travers des expérimentations.

Chapitre 2

Datasets

Les données mises en évidence pour étudier ces modèles concerneront des données visuelles et textuelles. Ainsi, les données récupérées contiendront des fichiers sous format *.jpeg*. Quant aux informations textuelles, nous aurons un champs pour le nom du lieu, un autre qui contiendra un commentaire sur le lieu et enfin un dernier champs qui se chargera de l'évaluation de la donnée. Les commentaires sont écrits en langage naturel, repris du fichier HTML du site, et doivent être convertis en une valeur numérique afin d'être exploités. Par ailleurs, l'évaluation de l'information correspond à une valeur entre 0 et 4 (une note sur 4) pour connaître la pertinence des informations.

Chapitre 3

Apprentissage

Une fois que les étapes de pré-processing ont été réalisées et que les données ont été générées, il faudra se focaliser sur la phase d'apprentissage avec les trois modèles que nous allons étudier ci-dessous à savoir l'Interactive GHSOM, la distance de Levenshtein et le modèle CSAIM. L'interactive GHSOM se contentera de classifier les données textuelles c'est-à-dire qu'il s'occupera des champs "commentaire", "évaluation" et "lieu". Les données visuelles ne seront pas traitées par GHSOM car les informations sont beaucoup trop importantes pour qu'on puisse extraire les caractéristiques spécifiques de l'image. C'est pourquoi, la distance de Levenshtein et CSAIM vont être utilisés pour les images.

3.1 Interactive GHSOM

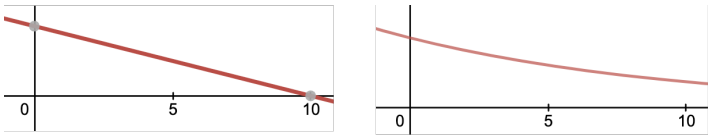
Avant de traiter directement du modèle Interactive GHSOM, nous allons décomposer l'architecture en traitant les différentes étapes ayant possiblement amener les auteurs à choisir l'Interactive GHSOM.

3.1.1 SOM

Le Self-Organizing Map [Koh01], également connu sous le nom de SOM, ou encore les cartes de Kohonen, est un réseau de neurones qui s'appuie sur des méthodes d'apprentissage non supervisées. Les cartes de Kohonen se basent sur des cartes qui permettent de réaliser le clustering des données et donc d'étudier la répartition des données selon leurs caractéristiques : c'est ainsi que nous réaliserons la classification des données textuelles entrées par les utilisateurs.
L'architecture de SOM se fonde sur les étapes suivantes :

- **Phase d'initialisation**
  - **Données** : les données sont déjà initialisées (voir chapitre précédent). Il faudra également penser à normaliser les valeurs pour SOM.
  - **Réseau** : les neurones seront représentées sous la forme de carte (d'où le nom de cartes de Kohonen). Chaque neurone contiendra un vecteur de poids, qui va être initialisé par rapport aux données, et une étiquette leur permettant de les classifier. La carte de neurones peut être initialisée avec des valeurs par défaut, à l'aide de la formule suivante : $round(5*n^{0.5})$ où *round* correspond à la fonction permettant d'arrondir le résultat à l'entier supérieur et n le nombre d'enregistrements dans la base de données. Par ailleurs, il est également nécessaire d'initialiser les paramètres du réseau qui vont servir à influencer l'apprentissage : deux paramètres seront ajoutés au réseau qui sont le coefficient d'apprentissage et le rayon de voisinage. Le coefficient d'apprentissage que nous nommerons α aura un impact lors de l'apprentissage d'un neurone tandis que le rayon de voisinage *nhd* permettra de savoir le nombre de voisins du BMU (dont on précisera la définition dans la partie suivante) capable d'apprendre avec le BMU. Ce dernier offre cette aspect coopératif au modèle, en aidant les voisins du BMU, à également participer à l'apprentissage des données.
- **Phase d'apprentissage** La phase d'apprentissage sera itérée *it* fois où *it* correspond à un nombre arbitraire choisi par l'utilisateur. La phase d'apprentissage peut être découpée en 2 parties : une partie d'apprentissage qui correspondra à $\frac{1}{4}$ du nombre d'itérations total et une autre partie d'affinage qui correspondra à $\frac{3}{4}$ du nombre d'itérations. La partie d'affinage initialisera certains paramètres avec des valeurs beaucoup moins importantes que lors de la première phase.
  - **Choix des données** : Pour commencer la phase d'apprentissage, nous allons tout d'abord choisir de façon aléatoire une donnée d de la base de données D.
  - **BMU** : Pour la donnée d choisie, nous trouverons le BMU (best match unit) qui correspond au neurone se rapprochant le plus de d. Le degré de similarité est calculé à l'aide de la distance euclidienne entre le vecteur de poids des neurones et la donnée d. Le BMU se traduira par le neurone ayant la plus petite distance euclidienne par rapport à d.

- **Apprentissage** : Une fois le BMU trouvé, afin que le BMU se rapproche le plus possible de d, nous lui appliquerons la règle d'apprentissage suivante : $w_v(t+1) = w_v(t) + \theta(u,v,t) * \alpha(t) * (x - w_v)$ où w représente le vecteur de poids du neurone, v est la position du neurone choisi dans la map, t est l'itération actuelle, θ est la fonction de voisinage, u est l'indice du BMU afin de savoir la qualité de l'apprentissage et x est le vecteur de donnée de d. θ sera équivalent à $\theta(u,v,t) = exp(\frac{-dst(u,v)^2}{2*nhd^2(t)})$ où dst correspond à la distance entre le BMU et le voisin en question.
- **Coopération** : Après le BMU modifié, appliquer la même règle d'apprentissage pour les voisins du BMU que nous avons expliquer ci-dessus, cependant cette fois-ci la fonction θ prendra son importance. La fonction θ aura un impact sur l'apprentissage des voisins selon la distance à laquelle sont les voisins du BMU : plus les voisins sont éloignés du BMU moins ils apprendront.
- **Mis à jour des paramètres** : À la fin de chaque itération, les paramètres $alpha$ et nhd seront mis à jour afin de limiter l'apprentissage. α décroitra au fur et à mesure des itérations afin de réduire l'apprentissage des BMU ($alpha$ evoluera de manière linéaire : $\alpha(t) = \alpha_{init} * (1.0 - \frac{it}{IT})$ où IT fait référence au nombre d'itérations total). De même pour nhd qui décroitra également, nous pouvons nous remettre à la fonction linéaire utilisée pour α, ou des fonctions qui décroit moins rapidement comme la fonction qui suit : $nhd(t) = nhd_{init} * exp(-\frac{it}{IT})$ (fig.3.1).



(a) (b)

FIGURE 3.1 – Réprésentation de la fonction (a) linéaire et (b) exponentielle

- **Phase d'étiquetage** : La phase d'étiquetage sert à visualiser la classification des données à l'aide de la carte de Kohonen que nous venons d'étudier lors de la phase d'apprentissage.

Cependant, d'autres alternatives de la carte de Kohonen existent et c'est ce dont nous allons voir ci-dessous afin d'améliorer les performances de ce modèle.

3.1.2 GSOM

GSOM (Growing Self-Organizing Map) fait parti des alternatives de SOM qui viendra pallier certains incovénients que nous pouvons rencontrer sur les cartes de Kohonen. En effet, la structure de réseau représente une carte fixe et les variantes qui proposent des cartes dynamiques ont tendance à réaliser des cartes beaucoup trop grandes et difficile à gérer. C'est là qu'intervient la variante GSOM [MCK13]. GSOM reprend les principes de SOM en ce qui concerne la phase d'apprentissage, il se différencie notamment par sa taille et par le fait qu'il soit dynamique. C'est pour cela que nous allons éviter, dans la suite de cette section, d'expliquer les détails que nous avons vu dans les sections précédentes : nous allons nous contenter de préciser que les ajouts par rapport au modèle précédent.

- **Initialisation** : Au lieu d'avoir une taille déjà prédéfinie comme avec SOM, GSOM commence avec une carte de 4 noeuds initiaux qui au fur et à mesure de l'apprentissage s'agrandira (fig.3.2). Pour que la carte s'agrandise et se développe, le modèle se penchera sur deux nouveaux paramètres qui s'ajoutent au réseau : le seuil d'agrandissement GT et l'erreur accumulative AE. A partir de ces paramètres, nous commençons l'apprentissage avec 4 noeuds, et en initialisant GT avec une valeur arbitraire ou en se fiant sur l'expression suivante : $GT = -n * ln(SF)$ où n correspond à la taille de la base de données D et SF qui correspond à une valeur arbitraire, à choisir judicieusement : plus SF est grand, plus la valeur de GT sera grande. Par conséquent, la croissance de la structure de la carte dépendra à la fois de la taille de la base de données et de SF.
  AE, contenue dans chaque neurone, sera initialisée à 0 et se rapporte à la différence, accumulée à travers les itérations, entre le vecteur de poids d'un neurone et une donnée choisie.

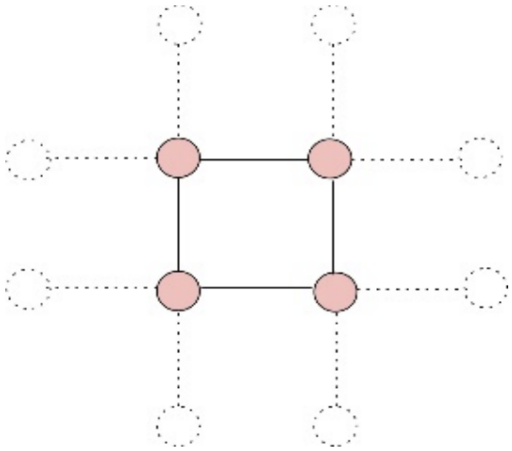


FIGURE 3.2 – Réprésentation des noeuds initiaux de GSOM qui peuvent se répendre sur les traits en pointillé

- **Apprentissage** : La phase se déroulera de la même manière qu'avec SOM, à savoir la recherche du BMU par rapport à une donnée d pris aléatoirement de la base de données, l'application de la règle d'apprentissage au BMU et à ses voisins. Néanmoins, il se distingue sur un seul point : le BMU choisi verra son AE s'accumulée ($AE_{BMU} = AE_{BMU} - (w_{BMU} - v_d)$). Si $AE_{BMU} > GT$, créer des noeuds autour du BMU si ce dernier est un noeud situant à l'extrémité de la grille. Si ce n'est pas le cas, donner les vecteurs de poids à ces voisins.

Dans la quête de trouver une taille adaptée selon la base de données, GSOM se trouve à être une très bonne solution pour pallier ce problème de taille. De plus il est également possible de contrôler la croissance de la carte à l'aide du SF lors de l'initialisation du seuil d'agrandissement GT.

3.1.3 GHSOM

Avec la réalisation de la variante de SOM à savoir GSOM, un des problèmes importants des cartes de Kohonen a été remédié. Au-delà de la structure statique de SOM, il existe un autre problème aux cartes de Kohonen : la difficulté à représenter les relations hiérarchiques entre les données. C'est pourquoi, Andreas Rauber a décidé de mettre en place GHSOM (Growing Hierarchical Self-Organizing Map) [Rau]. Ce dernier a la capacité de croître de la même manière que le modèle GSOM, mais en offrant des relations hiérarchiques entre les données (fig.3.3).

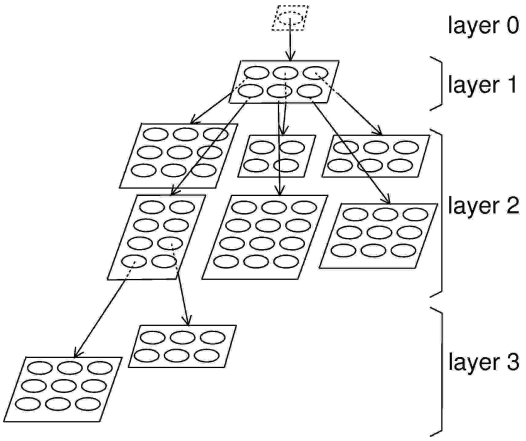


FIGURE 3.3 – Réprésentation de l'architecture GHSOM

- **Initialisation** : Comme pour GSOM, la carte sera initialisée avec 1 noeud pour la couche que nous nommerons la couche 0, car en effet, GHSOM introduira la notion de couche pour véhiculer cette idée de hiérarchie entre les données. Pour cela, nous mettrons également en place deux nouvelles valeurs τ_u et τ_m. D'un côté, τ_u s'occupera de la qualité de la représentation des données en entrée à la fin de la phase d'apprentissage, tandis que τ_m jouera sur l'architecture du modèle, avec le choix entre un modèle ayant des couches profondes ou des cartes plus importantes.
  Avant de commencer le processus de GHSOM sur la couche 0, il faut tout d'abord calculer l'erreur de quantification moyenne *mqe* pour la seule unité existante. *mqe* permettra d'avoir un impact sur l'évolution de l'architecture de GHSOM et se présente de la manière suivante : $mqe_{u_0} = \frac{||w_0 - x||}{n}$ où w_0 représente la moyenne de toutes les données en entrée, et d le nombre de données x. Une fois le calcul de mqe_{u_0} fait, nous allons pouvoir passer à la couche suivante et commencer notre processus d'apprentissage GHSOM avec la première couche ayant une grille de 2 * 2 unités. Chaque vecteur de poids des unités sera initialisé avec des valeurs aléatoires qui appartiennent aux données en entrée.

- **Apprentissage** : La phase se déroulera de la même manière qu'avec SOM, à savoir la recherche du BMU par rapport à une donnée d pris aléatoirement de la base de données, l'application de la règle d'apprentissage au BMU et à ses voisins. Cependant, ce modèle aura la possibilité d'évoluer comme GSOM mais pas forcément de la même manière. En effet, à des intervalles d'itérations donnés, nous allons calculer cette fois-ci l'erreur de quantification moyenne pour la carte qui prendra la

forme suivante : $mqe_{m_j} = \frac{\sum_{i=0}^{n} mqe_{u_i}}{u}$ où u représente le nombre de neurones dans la carte m. Nous distinguerons mqe_u et mqe_m comme étant respectivement le mqe d'une unité et d'une carte. mqe_{u_i} est calculée comme étant la distance moyenne entre le vecteur de poids w_i et les données en entrée pour l'unité i (comme nous l'avons vu pour mqe_{u_0}) Donc, deux sortes d'évolution sont possibles :

- **Si** $mqe_{m_j} \geqslant \tau_m * mqe_{u_0}$, alors une nouvelle colonne ou une nouvelle ligne est ajoutée à la carte m. Pour le choix de l'insertion, dans un premier temps, nous choisirons l'unité ayant la plus grande mqe dans la carte m que nous nommerons u_e, puis dans un second temps, l'ajout d'une colonne ou d'une ligne se décidera selon la position du voisin de u_e étant le plus différent par rapport à celui-ci que nous nommerons u_d. Dans la figure 3.4, u_e est placé au dessus de u_d, nous rajouterons donc une ligne de neurones dans la carte entre u_e et u_d. Si u_d était était place à la droite ou la gauche de u_e alors, nous aurions rajouté une colonne verticale entre les deux. Le poids des nouveaux neurones sera initialisé comme étant la moyenne des neurones u_e et u_d.

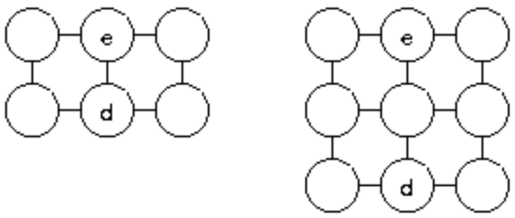


FIGURE 3.4 – Insertion de neurones dans une carte, dans GHSOM

- **Si** $mqe_{m_j} < \tau_m * mqe_{u_0}$, alors le modèle s'étendra avec la présence d'une nouvelle carte sur une nouvelle couche. Par conséquent, cela passe par l'analyse des unités de la carte afin de savoir la manière dont le modèle va se répendre. En effet les neurones pour lesquels $mqe_{u_i} > \tau_u * mqe_{u_0}$ seront candidats à la création d'un nouveau SOM sur une autre couche.

Ainsi, les paramètres τ_m et τ_u jouent des rôles cruciaux dans la phase d'apprentissage du modèle GHSOM et de sa croissance. Plus τ_m est grand, plus nous aurons de couches et plus le modèle sera profond. Par ailleurs, τ_u influencera sur l'étalement des cartes sur d'autres couches et impactera la représentation des données.

3.1.4 Interactive GHSOM

Takumi Ichimura et Takashi Yamaguchi proposent un GHSOM interactive [TI18d], se focalisant sur le contrôle de l'architecture de GHSOM en gérant de façon judicieuse les différentes branches constituant notre modèle. En effet, comme nous l'avons observé précédemment, le modèle de Andreas Rauber se fie aux paramètres τ_u et τ_m pour gérer la disposition des couches et des cartes. Par conséquent, GHSOM reste dans l'incapacité de transformer sa structure en données de façon adaptative lors de phase d'apprentissage et de la création des cartes. Nous nous retrouvons donc avec un nombre faible de données sur les cartes des couches les plus profondes.
Le modèle Interactive GHSOM consiste à maîtriser la forme de croissance en évitant les branches de cartes SOM inutiles car jugées redondantes, afin de permettre, quelque soit la profondeur de la couche, la formation d'un GHSOM qui soit plus interactive et organisé de manière à obtenir le moins de branche possible par rapport au modèle précédent. Cela passe alors par une reconstruction des méthodes de hiérarchie du GHSOM traditionnel : les méthodes entraînant la hiérarchisation des cartes seront affectées par ce processus désormais. En effet, deux conditions ont été mises en place dans le but de modérer la création de branches :

- **Si** $n_{BMU} \leqslant \alpha * n_I$ **et** $mqe_{m_j} < \tau_m * mqe_{u_0}$ où n_{BMU}, n_I et α représentent respectivement le nombre de données en entrée pour le BMU, le nombre de données en entrée total et une constante, alors nous ignorerons la hiérarchisation du modèle comme le voudrait le modèle principal mais nous allons plutôt insérer de nouveaux neurones en suivant les mêmes étapes que le GHSOM traditionnel.

- **Si** $mqe_{u_i} > \beta * \tau_m * \sum mqe_{u_j}, j \in S_k$ **et** $mqe_{u_i} > \tau_u * mqe_{u_0}$ où S_k est l'ensemble des BMU, alors nous allons créer une nouvelle unité au lieu de créer une nouvelle couche.

Ainsi nous remarquons que le plan de ce modèle joue sur la limitation de la stractification des couches dans le modèle GHSOM. Néanmoins, nous pouvons également pousser ce processus encore plus loin, en éliminant des unités pour alléger l'espace de nos cartes. Certains neurones peuvent être perçus comme étant des éléments inutiles à ce réseau, ou encore des éléments redondants en se basant sur certains critères.

3.2 Distance de Levenshtein

La distance de Levenshtein (LD) [DM10] permet de calculer la distance entre deux chaînes. La distance correspond au nombre d'insertion, de suppression, ou de substitutions nécessaire pour passer de la chaîne source s à la chaîne destination t. Plus LD est importante, plus les séquences sont différentes. Elle est notamment utilisée en cryptologie, pour la reconnaissance de pattern, la reconnaissance de la parole... LD se repose sur les étapes suivantes (fig.3.5) :

$$lev_{s,t}(i,j) = \begin{cases} \max(i,j) & \text{, if } \min(i,j) = 0 \\ \min \begin{cases} lev_{s,t}(i-1,j) + 1, \\ lev_{s,t}(i,j-1) + 1, \\ lev_{s,t}(i-1,j-1) + [s_i \neq t_j]) \end{cases} & \text{,} \\ & \text{, otherwise} \end{cases}$$

FIGURE 3.5 – Algorithme de la distance de Levenshtein

La figure 3.6 décrit le déroulement de LD en prenant comme example les chaînes *HYUNDAI* et *HONDA*. Comme nous le voyons la dernière case du tableau (celle qui se situe tout en bas à droite), retourne la valeur 3 qui correspond au nombre de changement à réaliser pour passer de la chaîne *HONDA* à *HYUNDAI* (ou vice-versa). En effet, il est nécessaire de remplacer le O par Y, puis ajouter les lettres U et I : ce qui équivaut donc à 3 changements, en partant de la chaîne *HONDA*.
La suite de ce paragraphe va décrire le résultat obtenu sur la première case bleue dans la figure 3.6 afin d'expliquer l'algorithme. La case obtenue pour H (de *HYUNDAI*) et H (de *HONDA*), nous prendrons la valeur minimale par rapport aux choix suivants : la valeur en haut de la case + 1, la valeur en bas de la case + 1 et la valeur de la case en diagonal (en haut à droite) + z où z sera équivalent à 0 si les deux caractères sont identiques et 1 si ce n'est pas le cas (en l'occurence, ici, nous avons deux caractères similaires donc $z = 0$). Ainsi nous avons le choix entre 2, 2 et 0, alors nous choisirons le troisième choix à savoir 0.

		H	Y	U	N	D	A	I
	0	1	2	3	4	5	6	7
H	1	0	1	2	3	4	5	6
O	2	1	1	2	3	4	5	6
N	3	2	2	2	2	3	4	5
D	4	3	3	3	3	2	3	4
A	5	4	4	4	4	3	2	3

FIGURE 3.6 – Exemple de l'utilisation de LD entre deux chaînes *HYUNDAI* et *HONDA*

En l'occurence, dans cet article, cette méthode est utilisée pour la reconnaissance et la similarité des images. De plus le parcours de la séquence de pixels de l'image sera quelque peu différente car elle s'appuyera sur des anneaux concentriques. Une fois, les similarités trouvées entre les images, ces dernières seront classifiées par le modèle CSAIM que nous verrons dans la section suivante.

3.3 CSAIM

Le modèle Clonal Selection Algorithm with Immunological Memory (CSAIM) est un système traitant à propos des interactions entre les anticorps *Abs* et les antigènes *Ags*, qui s'insèrera sur le modèle *RESCA* [TI18c]. A l'aide de la distance de Levenshtein, ce modèle résolvera le problème de la classification des images. C'est pourquoi, l'article décide d'introduire la méthode de l'hypermutation somatique, de modification du recepteur permettant de calculer le degré d'affinité entre les anticorps et les antigènes. En effet, en immunologie [JTH04], lors d'une interaction entre un antigène et les lymphocytes, nous constatons qu'un antigène réagit différemment selon les lymphocytes. Donc, les lymphocytes ayant les plus grandes affinités avec l'antigène seront alors répliquer à l'identique, le but étant de créer une population d'anticorps pouvant lutter contre l'antigène en question : c'est ce qu'on appelle la sélection clonale. Par ailleurs, pour lutter plus efficacement contre les antigènes, un phénonème d'hypermutation somatique est mis en place afin d'aider à ce que tous les anticorps puissent s'adapter aux antigènes.
Le modèle CSAIM nous aidera à traiter le problème de classification des anticorps selon les antigènes présents. La figure 3.7 représente la structure des anticorps pour la classification pour des données concernant les maladies du coeur se basant sur la base de données de Coronary Heart Disease : *CHD*

représente la base de données, θ représente le seuil qui est une valeur arbitraire choisie par l'utilisateur, w_i représente le poids de chaque anticorps, et ORIGIN, EDUCATE, TOBACCO, ALCOHOL, SBP, DBP, TC, LVH, qui représentent les causes des maladies (à noter qu'elles sont elles-mêmes catégorisées en groupe). Le poids des anticorps sera notamment analysé et adapté lors de la phase d'hypermutation somatique et de la modification des recepteurs, qui sera traité plus tard dans cette section.

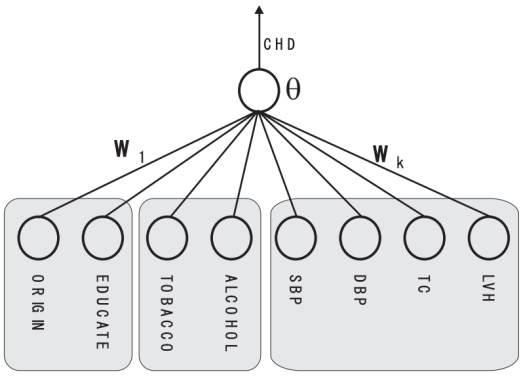


FIGURE 3.7 – Structure d'un anticorps

Ainsi, le modèle CSAIM se basant sur le modèle *RESCA* de Gao [SG18], se répartit en 8 étapes (fig.3.9) :

- Création d'un ensemble de *nab* anticorps comme étant candidat pour l'ensemble des antigènes *Ags*
- Calculer les affinités entre les antigènes et les anticorps à l'aide des formules suivantes. Nous noterons *D()* la fonction permettant de calculer l'affinité.
- Séléctionner les n anticorps de l'ensemble *nab*, ayant les meilleurs affinités. Nous les placerons dans un nouvelle ensemble qui est nommé par l'auteur comme étant l'*élite*
- Une fois l'ensemble d'*élite* recruté, nous nous chargerons de les trier selon les affinités et de les séparer en n ensembles d'*élite*.
- Cloner chaque ensemble d'*élite* selon son affinité avec l'antigène. Le nombre de clone généré suit la formule : $P_i = round(\frac{n-i}{n} * Q)$, avec i étant $i$ème ensemble d'*élite*, *round* étant la fonction permettant d'arrondir le résultat à l'entier le plus proche et Q étant la valeur représentant l'affinité du clone par rapport à l'antigène.

- Appliquer le phénonème d'hypermutation somatique ou l'édition des recepteurs. Afin de savoir lesquels d'entre eux vont être choisir, une note leurs sera attribuée, qui sera bien évidemment inversement proportionnelle à leur affinité avec l'antigène, vu que le principe de ces phénomènes consistent à aider à faire participer tous les anticorps, à travers les expressions suivantes : $P_hm = a/D()$ et $P_re = (D() - a)/D()$ avec P_hm et P_re respectivement pour hypermutation somatique et receptor editingn et a étant une valeur arbitraire.
  - Pour un poids aléatoire d'un anticorps choisi, nous lui appliquons les formules suivantes pour l'hypermutation somatique : $w_i = w_i + \Delta_w$ et $\theta = \theta + \Delta_\theta$ où Δw et θ sont respectivement $-\lambda_w < \Delta_w < \lambda_w$ et $-1 < \Delta_\theta < \lambda_\theta$ avec λ_{thetha} et λ_w des nombres petits.
  - La figure 3.8 montre le croisement réalisé lors de la phase de l'édition des récepteurs, qui se charge de croiser deux ensemble de w_i. En l'occurence dans l'exemple suivant, il s'agit de w_2 et w_{k-1}.

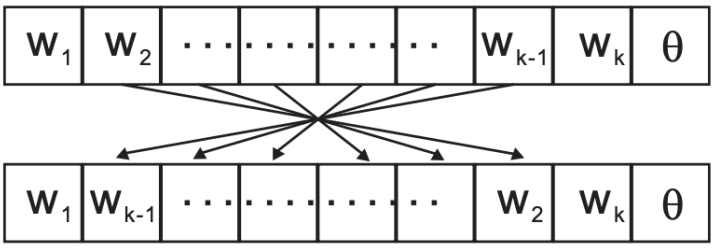


FIGURE 3.8 – Edition des recepteurs pour w_2 et w_{k-1}

- $D(Bi) = max(D(Ab_i, 1), ..., D(Ab_i, p_i)), i = 1, 2, ..., n$ permettra de choisir les anticorps B_i les plus appropriés pour l'antigène en question dans chaque ensemble d'*élite*.
- Pour chaque ensemble d'*élite*, nous mettons à jour les anticorps resortis grâce à l'étape précédente à travers le résultat retourné par la fonction Pb qui suit certaines conditions : if $D(Ab_i) < D(B_i)$ then $Pb = 1$, if $D(Ab_1) \geqslant D(B_1)$ then $Pb = 0, if D(Ab_i) \geqslant D(B_i), i \neq 1$ then $Pb = exp(\frac{D(B_i) - D(Ab_i)}{a})$.
- Remplacer l'ensemble *d'élite* ayant le moins d'affinité avec un nouvel ensemble d'anticorps à chaque itération t (t étant une valeur fixée arbitrairement) pour éviter les soucis de local optima [1]

1. La question du local optima se pose lorsque les anticorps se basant trop sur les

- Déterminer si le nombre maximal de génération G_{max} a été atteint. Si c'est le cas, nous retournons le meilleur anticorps ou sinon on continue la quatrième étape.

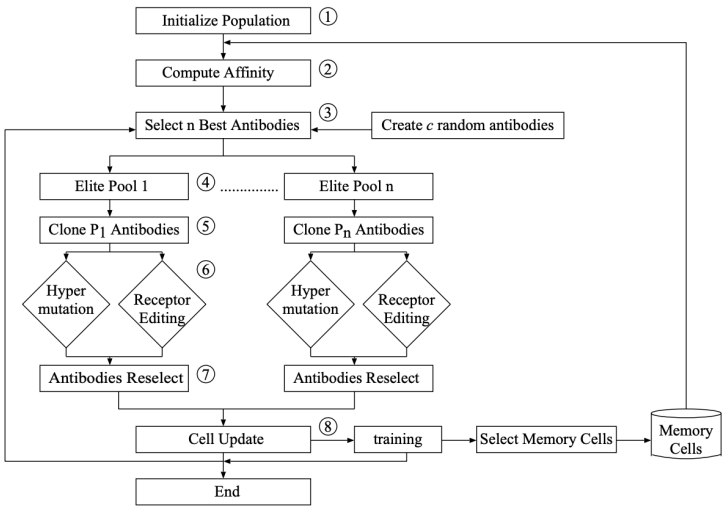


FIGURE 3.9 – Architecture du modèle CSAIM

De plus, pour clusteriser les anticorps obtenus par le modèle et sélectionner les cellules mémoires comme nous le voyons sur la figure 3.9, les anticorps partageant les mêmes caractéristiques sont rassemblés dans des cases et parmi eux, un seul est choisi pour devenir la cellule mémoire. Cependant, il est parfois possible que les anticorps se retrouvent à ne reconnaître aucun élément présent dans les données ; dans ce cas, une nouvelle classe est créée pour les mêmes anticorps formant un groupe.
De plus, le calcul de la distance euclidienne entre les données d'apprentissage normalisées et l'anticorps correspondant permet de regrouper les données en plusieurs catégories ; si la distance est inférieure à μ (μ étant la somme de 12 éléments donnés en entrée). Par conséquent, il faut transformer les données d'apprentissage en anticorps de la manière suivante : $d_i^{'} = d_i * \frac{h_i}{d_m in}(d_i \neq 0$ et $h_i \neq 0)$, où $d_m in$ représente la valeur minimale des éléments dans les données, d_i représente les données en entrée, h_i représente les anticorps. Ainsi, si la distance euclidienne entre d' et h est inférieure à μ, l'anticorps répond alors à la donnée (fig.3.10).

solutions proposées par leurs voisins offrent une solution que nous nommons local optima c'est-à-dire une solution par rapport à un ensemble partiel d'informations là où se distingue le global optima qui est la solution optimale parmi toutes les données.

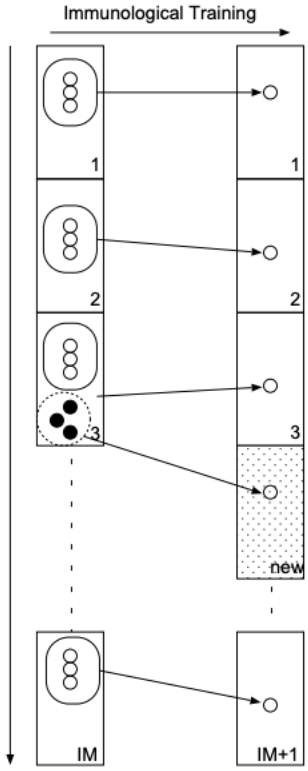


FIGURE 3.10 – Cluserting des cellules mémoires

Chapitre 4

Expérimentations

Pour tester l'efficacité des différents modèles que nous venons de décrire à savoir le modèle CSAIM, la distance de Levenshtein et l'architecture GHSOM. Comme expliqué précédemment, d'un côté, le modèle CSAIM et la distance de Levenshtein serviront à classifier les images ; et d'un autre côté, GHSOM servira à classifier les données textuelles car les images comportent des informations trop larges pour être gérer par le modèle. A noter que la fréquence des mots, concernant les commentaires, est calculée par la méthode *TF-IDF* (term frequency inverse document frequency) [WL18]. Plus le *TF-IDF* est élevé, plus les termes ont des chances d'être pertinents et offrir de meilleurs résultats. Ainsi, les phases de classification sont scindées en deux : les informations visuelles et les informations textuelles. Pour comparer nos résultats et connaître la pertinence des réponses obtenues par les modèles, nous allons les comparer avec les réponses provenant de DuckduckGo (fig.4.1).

FIGURE 4.1 – Résultats obtenus pour "Miyajima" sur Duckduckgo

4.1 Sans photo

Sur la figure 4.2, avec un *TF-IDF* élevé, le GHSOM semble très bien fonctionné malgré le manque d'informations visuelles. Il y a parfois un manque de pertinence sur le choix des images mais cela peut s'expliquer par le nombre de données proposées pour réaliser les expérimentations. Tandis qu'avec un *TF-*

IDF faible, les résultats sont moins intéressants car ils ne sont pas forcément représentatifs ou ne font pas assez référence au mot clef "Miyajima".

FIGURE 4.2 – Classification de Miyajima sans photo

4.2 Avec des photos

À l'aide des images (fig.4.3), les résultats devraient être de meilleures qualités. Avec un *TF-IDF* élevé, prenant comme mots clés les lieux les plus visités de Miyajima, nous tombons sur les mêmes résultats que lors de l'application du GHSOM sans les photos, à un détail près : la qualité des images obtenues est meilleure, avec des images plus précises comme nous le voyons avec le Torii qui est plus visible dans le second exemple. Quant à la classification avec photo et avec faible *TF-IDF* (sans mot clef), les résultats sont identiques à ceux obtenus par le GHSOM.

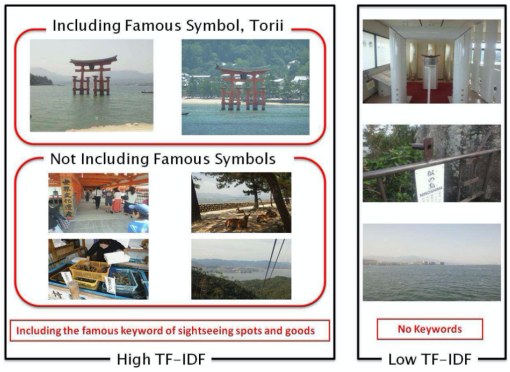


FIGURE 4.3 – Classification de Miyajima avec photos

Par conséquent, nous distinguons trois sortes de catégories de classifications : une se reposant sur la forte similarité entre les images et un *TF-IDF* très élevé qui renvoie ainsi aux lieux les plus touristiques et aux monuments les plus visités, une autre avec une faible similarité entre les images mais un *TF-IDF* très élevé qui renvoie plus ou moins au même résultat que GHSOM, et enfin la dernière, avec un *TF-IDF* faible et une faible similarité, est identique à celle de GHSOM.

Chapitre 5

Conclusion

Pour conclure, GHSOM, CSAIM semble être un ensemble de modèles efficace. Il parait être un modèle polyvalent capable de gérer de nombreux types de données (visuelles et textuelles, en considérant les données audios comme inutiles). Ce dernier se détache également par sa capacité à fournir des résultats pertinents en prenant en compte très peu de données. Le principal défaut du modèle vient notamment de MPPS et des données en entrée : les résultats obtenus sont parfois en mauvaise qualité, et pour renforcer la pertinence des images, il faudrait plus de données de la part des utilisateurs.

Bibliographie

[DM10] Manolis I.A. Lourakis Damien Michel, Antonis A. Argyros. *Horizon matching for localizing unordered panoramic images*. Thèse en informatique, a Institute of Computer Science, Foundation for Research and Technology – Hellas, N. Plastira 100, Vassilika Vouton, 700 13 Heraklion, Crete, Greece, 2010.

[JTH04] L.N. de Castro J. Timmis, T. Knight and E. Hart. *An Overview of Artificial Immune Systems*. Thèse en informatique, Computing Laboratory, University of Kent, Canterbury, UK, 2004.

[Koh01] Teuvo Kohonen. *Self-Organizing Maps*. Springer Science & Business Media, 2001.

[Map11] Hiroshima Tourist Map. Itproducts. Technical report, [online], 2011. `https://play.google.com/store/apps/details?id=com.navitime.local.navitime`.

[MCK13] Qiang Fang Mengxue Cao, Aijun Li and Bernd J. Kroger. *Growing Self-Organizing Map Approach for Semantic Acquisition Modeling*. Thèse en informatique, Laboratory of Phonetics and Speech Sciences, Institute of Linguistics Chinese Academy of Social Sciences, Beijing, China, 2013.

[MG] Merriam Park Software Michael Gilleland. Levenshtein distance. Technical report. `https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm`.

[Rau] Andreas Rauber. The ghsom architecture and training process. Technical report. `http://www.ifs.tuwien.ac.at/~andi/ghsom/description.html`.

[SG18] Gang Yang Shangce Gao, Hongwei Dai. *A Novel Clonal Selection Algorithm and Its Application to Traveling Salesman Problem*.

Thèse en informatique, Faculty of Management and Information Systems, Prefectural University of Hiroshima, 2018.

[TI18a] Shin Kamada Takumi Ichimura. *A Clonal Selection Algorithm with Levenshtein Distance based Image Similarity in Multidimensional Subjective Tourist Information and Discovery of Cryptic Spots by Interactive GHSOM.* Thèse en informatique, Faculty of Management and Information Systems, Prefectural University of Hiroshima, 2018.

[TI18b] Shin Kamada Takumi Ichimura. *Clustering and Retrieval Method of Immunological Memory Cell in Clonal Selection Algorithm.* Thèse en informatique, Faculty of Management and Information Systems, Prefectural University of Hiroshima, 2018.

[TI18c] Shin Kamada Takumi Ichimura. *Clustering and Retrieval Method of Immunological Memory Cell in Clonal Selection Algorithm.* Thèse en informatique, Faculty of Management and Information Systems, Prefectural University of Hiroshima, 2018.

[TI18d] Takashi Yamaguchi Takumi Ichimura. *A Proposal of Interactive Growing Hierarchical SOM.* Thèse en informatique, Faculty of Management and Information Systems, Prefectural University of Hiroshima, 2018.

[WL18] Ho Chung Wu and Robert Wing Pong Luk. *Interpreting TF-IDF Term Weights as Making Relevance Decisions.* Thèse en informatique, The Hong Kong Polytechnic University, Kam Fai Wong, the Chinese University of Hong Kong, 2018.