Projets informatiques
L3 EURIA

10 octobre 2024

Consignes :

— Les soutenances dureront 20 minutes (10 minutes de présentation et 10 minutes de questions). Elles auront lieu le **9 décembre**.

— Envoyer un premier compte-rendu avant le **9 novembre**. Ce comptes-rendu décrira les travaux effectués ainsi qu'un planning prévisionnel pour la suite du travail.

— N'attendez pas le dernier moment pour travailler sur les projets et n'hésitez pas à poser des questions.

— **N'oubliez pas de respecter la charte anti-plagiat de l'UBO** disponible via le lien ci-dessous
`http://ubodoc.univ-brest.fr/wp-content/uploads/2014/12/charte-antiplagiat-ubo.pdf`

— Le rapport (environ 4 pages au format .pdf) et les programmes commentés devront être déposés au plus tard le **3 décembre** sur la plateforme Moodle de l'EURIA :
`https://moodlesciences.univ-brest.fr/moodle/course/view.php?id=1083`

— l'équipe pédagogique ayant bien conscience de la possibilité de recourrir à des générateurs de code automatique (via des agents conversationnels), il est impératif que tout membre du groupe s'approprie le code rendu. En particulier, lors de la soutenance tout membre devra être capable d'expliquer précisément toute partie du code rendu questionné par le jury.

1. Classification et représentation de données par cartes de Kohonen :

— *Langage :* R, R-Shiny

— *Sujets abordés :* analyse de données, algorithme stochastique

1.a) Introduction

L'analyse de données multidimensionnelles porte typiquement sur l'étude de n individus déterminés par p variables (quantitatives ou qualitatives) : un individu est un élément d'un espace de dimension p.

Les méthodes factorielles (A.C.P. et dérivées), qui sont en fait des méthodes de projection de l'Algèbre linéaire permettent des *représentations* graphiques des données en dimension 1,2 ou 3. Il est cependant difficile de construire des classes de proximité à partir des projections si les individus ne sont pas représentables sans perte d'information dans un espace de dimension

≤ 3. Deux individus dont les projections sont proches ne sont pas toujours proches dans l'espace initial de dimension p.

D'un autre côté, nous pouvons distinguer deux familles de méthodes de *classification* :

- Classification hiérarchique : processus itératif d'agrégation des classes les plus voisines. Initialement, chaque individu est une classe. On considère les n classes et on regroupe les 2 plus proches (pour une distance choisie) pour former une nouvelle classe (de 2 individus). Et on itère sur les $n-1$ classes obtenues...jusqu'à l'obtention d'une seule classe. Ceci donne lieu à une représentation par un arbre. On peut ensuite choisir le nombre de classes qui paraît le mieux adapté en sélectionnant un niveau de regroupement.

- Classification non hiérarchique : le nombre de classes est fixé à priori et on attribue sa classe à chaque individu par un algorithme convergeant vers une répartition minimisant l'inertie intra-classe (i.e. les sommes des distances de chaque individu d'une classe au centre de la classe). Exemple : algorithme de moyennes mobiles, ou k-means.

Pour ces méthodes de classification, deux points d'une même classe sont proches dans l'espace initial, mais comment représenter globalement les classes en conservant la topologie initiale des données ? Il n'y a pas de notion de voisinage des classes.

L'algorithme de Kohonen tente de jumeler *Représentation* et *Classification*. Teuvo Kohonen a proposé dès 1982 un algorithme dont la fonction principale est de faire correspondre les éléments de l'espace des entrées avec des unités ordonnées sur une carte – une représentation graphique (de dimension 1, 2 ou 3) où chaque unité est entourée de ses voisines (pour une distance prédéfinie).

Le résultat est une fonction de l'espace des entrées vers l'ensemble des unités, telle que les images de deux éléments voisins au sens d'une certaine distance dans l'espace des entrées sont la même unité ou des unités voisines sur la carte.

A noter que T.Kohonen a proposé cet algorithme dans le cadre de ses travaux sur la modélisation mathématique du fonctionnement des neurones biologiques. L'algorithme a été conçu en 1982 comme une modélisation de la formation automatique de cartes dans les zones sensorielles du cortex même si depuis, l'étude mathématique de l'algorithme et ses applications – notamment en Analyse des données – l'ont éloigné de son cadre biologique originel. Par exemple, considérons les connexions nerveuses des cellules rétiniennes vers le cortex cérébral traitant les informations visuelles (le tectum). Si on imagine la rétine comme une grille de dimension 2, un fait important est que la topologie de la rétine est préservée par l'ensemble des connexions dans le sens que deux cellules proches sur la rétine sont connectées à deux cellules proches dans le cortex. Il n'y a aucune raison que ceci se réalise spontanément à la naissance, et on peut penser que ce phénomène est dû à un processus d'auto-organisation et de sélection gouverné par les activités spontanées des neurones cérébraux.

C'est précisément cette idée que T.Kohonen a voulu adapté lorsqu'il a défini son algorithme d'auto-organisation, ajoutant un concept important aux modèles connexionnistes antérieurs qu'il connaissait très bien (modèles de mémoire associative de D.Gabor, J.Hopfield, etc...). D'un point de vue algorithmique, le processus d'auto-organisation se déroule par une mise à jour locale des connections selon des règles complémentaires de compétition et de coopération, à chaque présentation d'un prototype.

1.b) Notations et définitions du modèle

L'**espace des données** est un sous-ensemble borné convexe $X \subset \mathbb{R}^p$. Nous considérons que

$\mathbb{R}^p$ est muni d'une norme $\|.\|$ (par exemple la norme associée à la distance Euclidienne).

$\Rightarrow$ On considére un échantillon $(x(1), \ldots, x(t), \ldots)$, issu d'observations successives dans X (approche statistique), ou une suite de réalisations de variables aléatoires indépendantes de même loi de probabilité μ à valeurs dans X (approche probabiliste).

Le **réseau** est formé de n unités (ou neurones) disposés selon une topologie déterminée :

- à $d = 1, 2$ ou 3 dimensions : une ligne, un carré ou un cube,

- selon un maillage dont la structure de voisinage est déterminée par une fonction de voisinage.

On représente les unités par un sous-ensemble I de $\mathbb{Z}^d$, et la fonction de voisinage est une fonction Λ définie sur $I \times I$,

- symétrique (i.e. $\Lambda(i, j) = \Lambda(j, i)$),

- dépendant seulement d'une distance D sur I, de norme associée $|\ .\ |$.

- décroissant avec la distance : $\Lambda(i, j)$ tend vers 0 quand $D(i, j) = |\ i - j\ | \longrightarrow +\infty$.

On convient souvent que $\Lambda(i, i) = 1$. Quand $\Lambda(i, j) = 1$, on dit que i et j sont fortement connectés, et quand $\Lambda(i, j) = 0$, i et j sont totalement disconnectés et n'ont pas d'interaction.

Chaque unité i est dotée d'un vecteur d'état $W_i(t) \in \mathbb{R}^p$ pointant dans l'espace des donnéees, et susceptible d'être modifié.

L'état du réseau à l'instant t est donné par $(W_i(t), i \in I)$.

$\Rightarrow$ L'objectif est de trouver des vecteurs W_i ayant des propriétés de :

- **Quantification** : le nombre de vecteurs W_i dans une région A de X est approximativement proportionnel à $\mu(A)$,

- **Organisation** : deux unités i et j proches (i.e. $\Lambda(i, j) \sim 1$) ont des vecteurs W_i et W_j proches.

Une fois de tels vecteurs W_i déterminés, ils sont utilisés pour définir la classe d'un élément $x \in X$ quelconque : on attribue à x la classe i^* telle que

$$i^*(x, W) = \text{argmin}\{\|x - W_i\|, i \in I\}.$$

Etudions maintenant l'algorithme de Kohonen qui détermine des vecteurs W_i par une méthode d'apprentissage.

1.c) L'Algorithme de Kohonen

- Les vecteurs $W_i(t = 0)$ sont initialisés aléatoirement.

- A l'instant t, l'état du réseau est donné par $W(t) = (W_i(t), i \in I)$. Un vecteur de l'espace des données $x(t + 1)$ est choisi aléatoirement.

- La **phase de compétition** désigne l'unité gagnante :

$$i^*(x(t+1), W(t)) = \text{argmin}\{\|x(t+1) - W_i(t)\|, i \in I\} \qquad (\mathbf{1})$$

(Dans le cas où plusieurs unités minimisent cette distance, on convient d'une règle, par exemple le premier indice i pour l'ordre lexicographique sur $\mathbb{Z}^d$).

• La **phase de coopération** modifie les vecteurs W_i :

$$\forall j \in I, W_j(t+1) = W_j(t) - \varepsilon_t \, \Lambda_t(i^*, j) \, (W_j(t) - x(t+1)) \qquad (\mathbf{2})$$

On poursuit l'algorithme tant que t est inférieur à une valeur seuil M fixée à priori, ou on peut imposer une condition d'arrêt alternative ou supplémentaire s'il n'y a plus d'amélioration notable. Seules les unités proches du neurone gagnant ont leur vecteur W_i modifié : notion de coopération.

Les paramètres essentiels sont :

- La dimension p de l'espace des données.

- La topologie du réseau.

- La fonction de voisinage Λ_t, constante ou dépendant du temps.

Exemples : $\Lambda_t(i,j) = \mathbf{1}_{\{|i-j| \leq k\}}, (k = 1, \text{ ou } 2)$.

$\Lambda_t(i,j) = g(\mid i - j \mid)$, où g est une fonction en cloche.

$\Lambda_t(i,j) = g(\frac{\mid i - j \mid}{\lambda(t)})$, où $\lambda(t) \to 0$, quand $t \to +\infty$.

Un exemple courant est : $\Lambda_t(i,j) = \exp(-\frac{\mid i - j \mid^2}{2\sigma(t)^2})$, où $\sigma(t) = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{t/M}$, avec $\sigma_i > \sigma_f > 0$ (par exemple $\sigma_i = 5$ et $\sigma_f = 0,2$).

- Le paramètre d'adaptation ε_t, à valeurs dans $]0,1[$, qui peut être constant ou décroissant en t. Par exemple, choisir ε_t tels que $\sum_{t \geq 0} \varepsilon_t = +\infty, \sum_{t \geq 0} \varepsilon_t^2 < +\infty$, comme $\varepsilon_t \sim 1/t$ (conditions de Robbins-Monro issues des Algorithmes stochastiques).

Un autre exemple courant est : $\epsilon(t) = \epsilon_i \left(\frac{\epsilon_f}{\epsilon_i}\right)^{t/M}$, avec $\epsilon_i > \epsilon_f > 0$ (par exemple $\epsilon_i = 0,1$ et $\epsilon_f = 0,005$).

- La loi de probabilité μ.

1.d) Applications

Cet algorithme et ses très nombreuses variantes connaissent une multitude d'applications, par exemple : analyse des données (classification), traitement du signal, reconnaissance d'images et acoustiques (paroles,...), prévision de séries chronologiques, robotique, optimisation (le problème célèbre du voyageur de commerce).

On pourra consulter notamment l'article de M. Cottrell *et al* (2003) où sont présentés quelques exemples : analyse et comparaison de pays repérés par des variables socio-économiques, courbes de consommation électrique, démographie dans des communes de la vallée du Rhône, profils de consommateurs au Canada, segmentation du marché du travail en France, etc. L'avantage de cette méthode de classification est qu'elle permet de réprésenter graphiquement les observations en respectant la topologie du réseau. On associe ainsi à chaque unité les observations appartenant à cette classe. Lorsque les données et les classes sont très nombreuses, il est possible de réaliser une deuxième classification sur les vecteurs représentant les classes, afin d'obtenir un niveau de classification plus grossier, qui sera plus facile à interpréter. De nombreuses variantes de cet algorithme ont été proposés. En particulier, il est possible de l'appliquer dans le cas de variables qualitatives, en l'associant à une analyse factorielle des correspondances (Cottrell *et al*, 2003]).

Dans le cadre de ce projet, il est demandé d'implémenter cet algorithme et de réaliser une interface R-Shiny pour rendre l'application interactive, avec des animations proposées suite à des choix de paramètres. Pour l'application R-Shiny, on considérera des données en dimension 2, et une grille en dimension 1 ou 2. L'avantage est qu'on pourra superposer graphiquement l'espace des données et la grille et représenter les vecteurs W_i dans l'espace des données. L'objet de l'interface est d'être un outil pédagogique permettant de visualiser facilement le mécanisme d'auto-organisation des vecteurs W_i.

1.e) Bibliographie

M.Cottrell, S. Ibbou, P. Letremy,, P. Rousset, "*Cartes organisées pour l'analyse exploratoire des données et la visualisation*", Journal de la Société Française de Statistique, n° 144, p. 67-106, (2003). (`https://arxiv.org/abs/math/0611422`)

2. Analyse statistique

— *Langage :* R
— *Sujets abordés :* Visualisation de données et statistiques exploratoires

L'objectif général de ce projet est de réaliser une analyse statistique et graphique d'un jeu de données, en utilisant le langage R et ses packages spécialisés dans la visualisation de données.

1. Choix et préparation des données Vous devez tout d'abord sélectionner un jeu de données riche et pertinent. Vous pouvez explorer des sources de données publiques ou collaboratives telles que :

— `https://www.data.gouv.fr/fr/`
— `https://opendata.brest-metropole.fr/pages/accueil/`
— `https://www.kaggle.com/datasets`

Les données choisies doivent être suffisamment variées pour permettre des analyses intéressantes et des visualisations sophistiquées (variables catégorielles, numériques, géographiques)
Tâches associées : Téléchargement des données avec R ; Nettoyage des données (gérer les valeurs manquantes, erreurs de saisie,...) et transformations (création de variables dérivées, regroupement de catégories, ...)

2. Analyse statistique Une fois les données prêtes, une **analyse statistique exploratoire** sera réalisée pour comprendre la distribution des données et mettre en avant des liens statistiques entre les données.
Tâches associées : Calculs statistiques de base (moyenne, médiane, écart-type, fréquences,...) ; Analyse des relations entre variables (corrélations, tests statistiques,...)

3. Visualisation des données L'accent principal du projet sera mis sur la **visualisation des données** en utilisant par exemple le package `ggplot2`. Vous devrez produire une variété de graphiques afin d'illustrer les caractéristiques des données de manière claire et informative.
Tâches associées : Graphiques de base (histogrammes, boxplots, scatter-plots, barplots) ; Graphiques avancés (par exemple superposition de plusieurs couches d'information sur un même graphique) ; Tracer des cartes (packages `ggplot2, sf, leaflet`) ; Visualisation interactive [facultatif] (par exemple package `plotly)`.

4. Critères d'évaluation Le projet sera évalué sur plusieurs critères :

1. Pertinence des analyses statistiques : Choix approprié des méthodes d'analyse en fonction du jeu de données.
2. Qualité et variété des visualisations : Clarté, originalité et complexité des graphiques produits avec `ggplot2` ou d'autres packages.
3. Programmation et organisation du code : Usage judicieux des fonctions, modularité du code et bonnes pratiques de programmation.
4. Rapport final : Structure, qualité des explications, intégration harmonieuse du code et des résultats avec R Markdown (`http://larmarange.github.io/analyse-R/rmarkdown-les-rapports-automatises.html`)

3. Déplacement d'un robot et Optimisation

— *Langage :* Python ou R
— *Sujets abordés :* Simulation, représentation graphique, système discret et contrôle.

Contexte. On souhaite simuler le déplacement d'un robot dans un domaine borné $\mathcal{D} \subset \mathbb{R}^2$ d'intérieur non vide, connexe et défini par $\mathcal{D} = \{(x, y) \in \mathbb{R} \mid d(x, y) \leq 0\}$ où $d : \mathbb{R}^2 \ni (x, y) \mapsto d(x, y) \in \mathbb{R}$ est une fonction différentiable (convenablement choisie). Par exemple pour $d(x, y) = x^2 + y^2 - 1$, $\mathcal{D}$ correspond au disque unité.

Soit $z_0 \in \partial\mathcal{D}$ correspondant à la position initiale du robot. La position du robot à l'instant t est une fonction $\mathbb{N} \ni t \mapsto z(t) \in \mathcal{D}$ et le robot au temps t est modélisé par le disque centré en $z(t)$ et de rayon r et noté $R(t) := \{w \in \mathbb{R}^2 \mid |w - z(t)| \leq r\}$ où $|.|$ est la norme euclidienne de $\mathbb{R}^2$. La position z est déterminée pour tout temps entier naturel par le système discret :

$$(S) \qquad \begin{cases} z(t+1) & = z(t) + u(t), \qquad \forall t \in \mathbb{N} \\ z(0) & = z_0 \in \partial\mathcal{D} \\ z(t) & \in \mathcal{D}, \quad \forall t \in \mathbb{N} \\ u(t) & \in U := \{u \in \mathbb{R}^2 \mid |u(t)| \leq 1\} \end{cases}$$

où la fonction $u : t \mapsto u(t) \in U$, appelée contrôle, est à déterminer pour que la contrainte $z(t) \in \mathcal{D}$ soit respectée pour tout entier naturel t. Il est tout à fait possible d'utiliser des heuristiques relativement simple pour le calcul de $u(t)$ (du moment que la contrainte $z(t) \in \mathcal{D}$ est satisfaite), mais cela impactera la performance de votre algorithme (cf. le cahier des charges).

Travail à faire. Pour que votre code soit évalué sur les fichiers tests de l'équipe pédagogique, il est important de respecter rigoureusement la signature des fonctions suivantes.

1. Écrire une fonction `init()` qui initialise les valeurs de d, r et $z0$ avec la structure suivante (donnée ici en Python) :

```
def init():
    def d(x,y):
        return(x**2+y**2-1)
        #autre exemple non convexe:
        #return(((x^2*(x+1)*((x-2)/4)+((y^4-2*y^2+y+2)/3)-2*x*y)/2)-0.5)
        #...
    def rayonR(f):
        #calcule du rayon r dans la definition du disque R(.)
        pass
    r = rayonR(d)
    def genererz0(f):
        #genere la position initiale sur le bord de D:{f<=0}
        pass
    return([d,r,z0])
```

   La fonction `d` donnée en exemple est relativement simple : le domaine $\mathcal{D}$ associé est le disque unité. C'est un domaine intéressant pour le développement du programme mais

assez pauvre pour évaluer ses performances (cf. cahier des charges). De manière générale, le domaine $\mathcal{D} = \{(x, y) \in \mathbb{R}^2 \mid d(x, y) \leq 0\}$ doit être non vide, connexe, borné (et non forcément convexe). La diversité des différents domaines considérés sera prise en compte dans l'évaluation (voir le point "tests" dans le cahier des charges).

Le rayon r calculé par la fonction `rayonR` est égal au rapport $\tilde{r}/10$ où $\tilde{r}$ est une estimation du rayon du plus grand disque inclus dans $\mathcal{D}$.

La valeur de retour la liste $[d, r, z0]$ est ensuite passée à la fonction `simuler` suivante.

2. Écrire une fonction `simuler(d,r,z0)` qui prend une fonction d définissant le domaine $\mathcal{D}$ et simule le mouvement du robot en suivant la dynamique donnée par le système (S). La fonction s'arrêtera au premier temps $T \in \mathbb{N}$ tel que

$$\mathbb{Z}^2 \cap \bigcup_{t=0}^{T} R(t) \subset \mathbb{Z}^2 \cap \mathcal{D}. \tag{1}$$

Pour chaque point $z \in \mathbb{Z}^2 \cap \mathcal{D}$, on assigne le nombre $f(z) = \frac{1}{T} \sum_{t=0}^{T} \mathbf{1}_{R(t)}(z)$ où $\mathbf{1}_Z$ est la fonction indicatrice de l'ensemble Z. On note $N = \text{card}(\mathbb{Z}^2 \cap \mathcal{D})$, on calculera lors de l'exécution de `simuler` la quantité

$$V = \frac{1}{N} \sum_{z \in \mathbb{Z}^2 \cap \mathcal{D}} (f(z) - m)^2 \tag{2}$$

où $m = \frac{1}{N} \sum_{z \in \mathbb{Z}^2 \cap \mathcal{D}} f(z)$.

A la fin de son exécution, la fonction `simuler(d,r,z0)` devra retourner une liste dont les éléments sont dans l'ordre : la liste $[u(0), u(1), \ldots, u(T-1)]$, l'entier T définis par (1) et la valeur de V définie par (2). Si votre fonction `simuler` renvoie $T \geq 1e6$ ou ne converge pas alors retourner $+\infty$ pour les valeurs de T

Cahier des charges. Votre code sera évalué à partir des éléments suivants :

- le respect rigoureux du nom des fonctions, des arguments et des valeurs de retours.
- la grande part d'initiative personnelle du projet se situe sur le calcul du contrôle u dans la fonction `simuler` et du choix de la position initiale $z0$. La performance de votre algorithme sera évaluée à partir des valeurs de T et V pour différents fichiers tests. Plus ces valeurs sont petites et plus votre algorithme sera considéré comme performant, la difficulté étant que vous ne connaissez pas la fonction d sur laquelle votre fonction `simuler` va être testée.
- **tests :** vous exécuterez successivement `simuler` pour plusieurs fonctions d convenablement choisies dans le but de montrer les performances/limites de votre programme. Vous noterez alors les valeurs de T et V dans un tableau que vous pourrez mettre en annexe de votre rapport. Si votre fonction `simuler` a un comportement aléatoire, calculez les valeurs moyennes de T et V pour plusieurs exécutions de `simuler` sur le même domaine d.
- la complexité de votre algorithme sera prise en compte, relativement par rapport à la poltique de calcul de u. Néanmoins assurez-vous que le temps d'exécution de `simuler` n'excède pas la minute sur un ordinateur standard.
- toute initiative personnelle pertinente sera valorisée. Par exemple l'illustration graphique du domaine $\mathcal{D}$, ou un graphique représentant les fréquences $f(z)$, $z \in \mathbb{Z}^2 \cap \mathcal{D}$, ou l'analyse graphique de l'évolution de T par rapport à l'aire de $\mathcal{D}$,...

Bonus.

- prendre en compte des domaines plus généraux, c'est à dire définis comme l'intérieur d'une courbe brisée et fermée (par exemple la liste `[[1,1],[-1,1],[-1,-1],[1,-1]]` définirait $\mathcal{D}$ comme le carré de côté 2 et centré en 0).
- créer une interface graphique avec par exemple : R Shiny, Shiny for Python, Pygame, Python Dash ... et une fenêtre dédiée à l'affichage de la trajectoire $t \mapsto z(t)$ du robot pour $t \in \{0, \dots, T\}$.

4. Algorithme des k Plus Proches Voisins sur des images

— *Langage :* Python (ou R)
— *Sujets abordés :* Manipulation et traitement d'images, Algorithme des plus proches voisins.

Partie 1 : Introduction sur la méthode des k-plus proches voisins

— Faire l'état de l'art sur l'algorithme des plus proches voisins : principe, complexité, applications, avantages/inconvénients/limites...
— À partir d'une collection d'images de votre choix, implémenter cet algorithme pour résoudre un problème académique que vous aurez choisi.

Partie 2 : Application au décodage de Captcha

Contexte. DefendTheWeb (`www.defendtheweb.net`) est une plateforme proposant des contenus pour découvrir le monde du hacking éthique à travers différents articles, challenges et une communauté. On s'intéresse ici au challenge Captcha4 `https://defendtheweb.net/playground/captcha4` (créer un compte sur la plateforme pour accéder au contenu). On dispose d'une table de correspondance (Figure 1 (droite)) qui, à un émoticône donné, fait correspondre une chaîne de deux caractères ASCII. La question génère aléatoirement une image comme celle représentée en Figure 1 (gauche) sur laquelle est disposée plusieurs émoticônes, chacun appartenant (à une couleur/rotation près) à un émoticône de la table de correspondance. Votre but consiste à entrer dans le champ "Answer" la chaîne de caractères correspondant aux émoticônes affichés en moins de cinq secondes. Par exemple, la réponse à l'exemple donné est :

:);)B) : D : O : (;) : @ : / : (

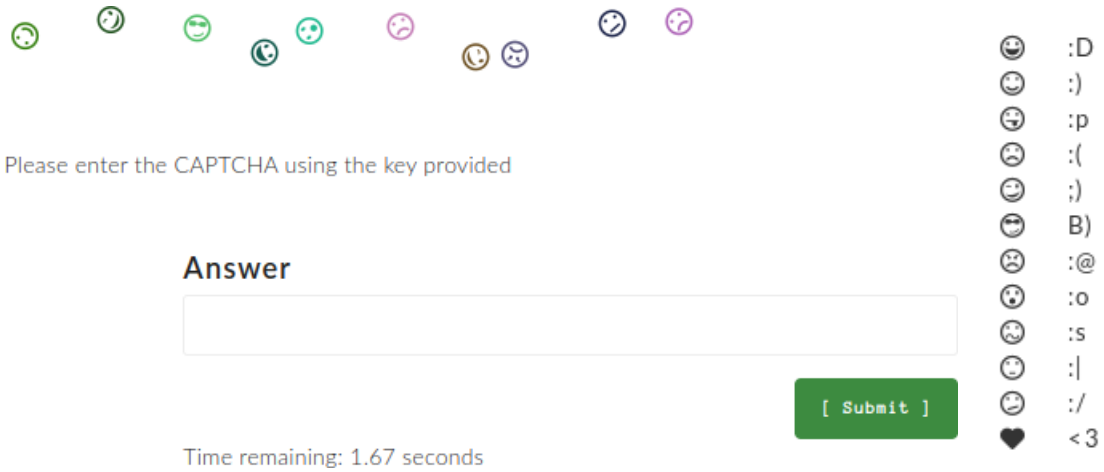

FIGURE 1 – (gauche) Décoder le code dans le champ "Answer" en moins de cinq secondes. (droite) Règles de codage.

Travail demandé

Pour résoudre le challenge, votre méthode devra être basée sur un algorithme de type "plus proche voisin" que vous avez étudié en partie 1. Voici quelques points dont vous pourrez vous inspirer.

1. une première étape serait de construire votre base de données d'entraînement. Pour cela, à partir de plusieurs images correspondant à la Figure 1 (gauche), vous pouvez extraire la liste de dix images correspondant aux 10 émoticônes. Ces images constituent les données d'entraînement pour l'agorithme des plus proches voisins. Un exemple d'une telle base est donnée en Figure 2.

FIGURE 2 – Exemple de dnnées d'entrâinement à constuire.

   Pour cette étape, on pourra utiliser la librairie `opencv` ou `opencv2` ou `PIL` disponible sur Python et R.
2. Implémenter l'algorithme des plus proches voisins pour résoudre le challenge. (quel est l'espace des données ? qui sont les voisins ? quelle distance sur l'espace des données choisir ? (la distance euclidienne entre deux images n'est peut être pas appropriée ...).
3. La qualité des données d'entraînement est cruciale. En particulier, assurez vous que les émoticônes extraits soient centrés et tous de même taille. Le choix de la distance est aussi important : n'hésitez pas à parcourir la documentation d'opencv pour trouver une fonction adéquate au sens où si la distance entre deux émoticônes est petite, alors les deux émoticônes sont les mêmes (à une couleur et rotation près)
4. Voici un code Python pour vous connectez sur la plateforme, téléchargez l'image, la sauvegarder et envoyer la solution automatiquement.

```
import requests
from requests import Session
from io import BytesIO
from PIL import Image

# ouvrir une session sur la plateforme a partir de votre compte
session     = requests.Session()
```

```
session.post("https://defendtheweb.net/?login", ...
   {"username" : "votre-username", "password" : "votre-mot-de-passe"})

# telecharger le captcha
session.get("https://defendtheweb.net/extras/playground/captcha/captcha4.php")

# sauvegarde de l'image en local
image       = Image.open(BytesIO(rsp.content))
image.save("captcha.png")

#####################################################
#Implémentez votre méthode ici
#.
#.
#.
#####################################################
# poster votre solution (stockee dans la variable sol de type 'str'
session.post("https://defendtheweb.net/playground/captcha4", {"answer" : sol})
```