DS 3

Informatique de tronc commun, classe de PC

Julien Reichert

Ce devoir consiste en trois parties totalement indépendantes.

Partie 1 : Bases de données MNIST (mais ce n'est pas du SQL !)

Mentionnée en cours, la base de données MNIST recense des milliers d'images de chiffres manuscrits (60000 pour l'apprentissage et 10000 pour le test) utilisées pour tester les algorithmes de reconnaissance. Nous allons écrire un tel algorithme qui s'appuie sur l'apprentissage supervisé.

Les fichiers disponibles sont des images carrées en noir et blanc de 28 pixels de côté. On les représentera comme des listes de listes de booléens (couleur noire = vrai, couleur blanche = faux), ces derniers pouvant être remplacés par les entiers correspondants.

Pour comparer deux images dans l'optique d'utiliser l'algorithme kNN, la fonction de distance utilisée ne sera pas la distance euclidienne mais le nombre de pixels de différence, c'est-à-dire le nombre d'éléments différents dans la liste de listes.

Exercice 1.1 : Pour se rapprocher du cadre délimité par le programme officiel, on pourrait assimiler les listes de listes à des points dans un espace $\mathbb{R}^d$. Que vaudrait d dans ce cas ?

Exercice 1.2 : Écrire une fonction prenant en argument deux listes de listes dont on supposera que le nombre de listes et les tailles correspondent (si on veut que tout vaille `28`, c'est autorisé...) et renvoyant le nombre d'éléments différents entre les deux.

Exercice 1.3 : Écrire, éventuellement en reprenant exactement l'algorithme kNN du cours et du TP 5, une fonction prenant en argument (i) une liste de données d'entraînement, qui sont des couples formés par un chiffre manuscrit représenté en tant que liste de listes de booléens comme indiqué en introduction et par un entier correspondant au chiffre écrit, (ii) une donnée supplémentaire sous forme de liste de listes de booléens selon le même principe et (iii) tous les paramètres supplémentaires nécessaires pour le bon fonctionnement de l'algorithme, et renvoyant le chiffre représenté par la liste de listes en deuxième argument.

Exercice 1.4 : On considère les variables (globales si on veut) `ENTRAINEMENT` et `TEST` contenant chacune des listes de données d'entraînement de la même forme que dans l'exercice précédent. En se servant de la fonction précédente (si l'exercice n'est pas fait, on peut supposer l'existence de la fonction en donnant sa spécification), écrire une fonction (avec des arguments pertinents) qui construit la matrice de confusion des données de test et qui renvoie à partir de cette matrice de confusion le taux de réussite de l'algorithme exprimé en tant que pourcentage en maintenant les deux chiffres après la virgule vu la taille attendue pour la variable `TEST`.

Exercice 1.5 : En considérant que chaque comparaison entre deux images nécessite une microseconde, que chaque comparaison ou opération arithmétique entre deux nombres nécessite dix nanosecondes, que chaque accès en écriture à un élément d'une liste ou d'un dictionnaire nécessite dix nanosecondes également et en extrapolant la durée de toutes les autres opérations supplémentaires des deux exercices précédents, donner un ordre de grandeur du temps mis pour calculer le taux de réussite avec la base de données MNIST.

Pour l'exercice 1.5, la réponse sera à exprimer en tant qu'intervalle entre deux puissances de dix qui sont deux multiples de trois successifs, d'unité la seconde. Autrement dit on répondra « entre 1 ms et 1 s » ou « entre 1 s et 1000 s », par exemple.

Partie 2 : Intelligence artificielle pour le jeu des bâtonnets

Cette partie s'inspire d'un travail trouvé sur internet à l'adresse `https://members.loria.fr/MDuflot/files/med/IAnim.html`, citant des sources supplémentaires.

Nous allons construire par apprentissage une stratégie gagnante pour le jeu des bâtonnets, dont on rappelle brièvement les règles : vingt-et-un bâtonnets sont disposés en ligne, chaque joueur en retire entre un et trois à tour de rôle, et si un joueur ne peut pas jouer, il a perdu, de sorte que le joueur qui retire le dernier bâtonnet gagne.

Exercice 2.1 : Rappeler la liste des configurations gagnantes pour Ève, parmi les configurations détaillant l'état de la partie en tant que couples (joueur dont c'est le tour, nombre de bâtonnets restants). On pourra la décrire par une phrase ou une expression mathématique ou, si on n'a rien de mieux à faire, explicitement en tant qu'ensemble...

Exercice 2.2 : Dessiner l'arène de jeu en nommant les sommets selon le même principe que pour les configurations que dans l'exercice précédent. On restreindra l'arène aux configurations où il reste entre zéro et huit bâtonnets.

Exercice 2.3 : Dans le dessin précédent, entourer de manière précise les ensembles $\mathcal{A}_0$, $\mathcal{A}_1 \setminus \mathcal{A}_0$, $\mathcal{A}_2 \setminus \mathcal{A}_1$, $\mathcal{A}_3 \setminus \mathcal{A}_2$ et $\mathcal{A}_4 \setminus \mathcal{A}_3$ où chaque $\mathcal{A}_k$ est l'ensemble gagnant pour Ève lors de l'étape k du calcul de l'attracteur. Les différences ensemblistes permettent de ne pas surcharger le dessin vu les inclusions successives...

En ce qui concerne la partie apprentissage, nous allons construire l'ensemble des stratégies positionnelles possibles en construisant pour chaque joueur une liste de listes : à l'indice `i`, pour `i` entre `3` et `21`, on met la liste `[1, 2, 3]`, et aux indices entre `0` et `2` on retire les éléments strictement supérieurs à l'indice. Il s'agira de l'ensemble initial des coups possibles.

Exercice 2.4 : Écrire une fonction qui renvoie l'ensemble initial des coups possibles d'un joueur selon la représentation ci-avant.

L'apprentissage se fait selon le principe suivant : au cours d'une partie, le joueur dont c'est le tour consulte la liste à l'indice correspondant au nombre de bâtonnets restant. Il pioche un élément au hasard dans cette liste (on pourra utiliser la fonction `choice` du module `random` pour ce faire) et retire le nombre de bâtonnets correspondant, mais si la liste est vide, le joueur abandonne. Quand la partie est terminée, le joueur qui abandonne (par exemple parce qu'il ne reste aucun bâtonnet) revient à la dernière étape où la liste n'était pas vide et supprime le coup qu'il a fait (ceci nécessite en particulier de stocker l'information nécessaire dans une variable, voire plusieurs). Intuitivement, le dernier coup effectué n'était pas gagnant et donc ne fait pas partie d'une stratégie gagnante, mais si la liste se retrouve vide, c'est que la position elle-même n'est pas gagnante et l'abandon se justifie parce qu'après la phase d'apprentissage chaque joueur est censé jouer optimalement.

Exercice 2.5 : Écrire une fonction qui lance une partie et fait la mise à jour correspondante, les deux ensembles de coups possibles étant en argument. On considère que c'est toujours Ève qui commence.

Exercice 2.6 : Écrire une fonction sans argument qui lance l'expérience et répète la fonction précédente jusqu'à ce que la liste à l'indice `20` de la liste d'Adam soit vide. La fonction renverra les deux listes de coups restants.

Exercice 2.7 : Une fois la fonction précédente terminée, quel est l'élément qui reste à l'indice `21` de la liste d'Ève s'il n'en reste qu'un ? Et par ailleurs, en reste-t-il forcément un seul ?

En raison de l'intervention du hasard, le nombre de parties nécessaires pour arrêter la fonction de l'exercice 2.6 n'est pas forcément fixe, et dépend si les coups joués au hasard sont optimaux ou non.

Exercice 2.8 (difficile) : Quel est le nombre maximal de parties nécessaires à l'arrêt de l'expérience ? Justifier rigoureusement.

L'intérêt de cette modélisation est qu'on peut aussi faire jouer un humain contre l'IA ainsi programmée. Si l'IA commence, elle ne tardera pas à gagner systématiquement une fois la stratégie gagnante construite.

Partie 3 : Faciliter le travail de P. et S.

Toute ressemblance avec une compétition organisée dans la classe de PC n'est pas fortuite...

Dans le cadre du challenge « 0 à 1000 », dix équipes numérotées de `0` (le professeur) `9` répondent à des questions de culture générale. Le score à chaque question est marqué selon le principe suivant :

- L'équipe la plus proche de la bonne réponse marque trois points.
- La deuxième équipe la plus proche de la bonne réponse en marque deux.
- La troisième équipe la plus proche de la bonne réponse en marque un.
- Les autres n'en marquent pas.
- En cas d'égalité, le meilleur score est marqué par toutes les équipes à égalité, quitte à ce que plus de trois équipes marquent des points. Par exemple, si trois équipes sont à égalité pour la troisième place, chacune marque un point. Par ailleurs, si deux équipes sont à égalité pour la deuxième place, elles marquent certes deux points chacune, mais aucune équipe ne marquera alors un point. De même, si deux équipes sont à égalité pour la première place, l'équipe suivante marque un point et sans égalité pour la troisième place c'est tout pour les attributions.

Une fois toutes les questions posées, un classement général est établi.

Il n'est pas nécessaire de se servir de dictionnaires pour cette partie, mais cela reste autorisé. Dans ce cas, toute mention de « liste » dans les exercices à suivre pourra être remplacée par « dictionnaire » si on le souhaite.

Exercice 3.1 : Écrire une fonction prenant en argument la liste des résultats à chaque question et renvoyant la liste des scores de chaque équipe à l'issue du challenge. Un résultat à une question est un couple formé par la bonne réponse d'une part et la liste des dix réponses fournies par les équipes d'autre part.

Exercice 3.2 : Écrire une fonction prenant en argument la liste des scores de chaque équipe et renvoyant une liste de couples (équipe, classement) en respectant les égalités. Par exemple, dans une version restreinte à cinq équipes, si la liste des scores est `[20, 10, 13, 12, 12]`, la fonction doit renvoyer `[(0, 1), (2, 2), (3, 3), (4, 3), (1, 5)]` ou la même liste en échangeant `(3, 3)` et `(4, 3)`. On écrira soi-même un algorithme de tri au choix.

Pour une version en SQL de la gestion de la compétition, on considère la table `SOUMISSIONS` d'attributs `question`, `equipe` et `reponse`. La valeur `-1` pour l'attribut `equipe` permet de mettre la bonne réponse sans utiliser une table supplémentaire pour cela.

On signale l'existence de la fonction `ABS` pour calculer la valeur absolue d'un nombre en SQL.

Exercice 3.3 : Écrire une requête affichant la bonne réponse à la question numéro `8`.

Exercice 3.4 : Écrire une requête affichant la distance entre la proposition de l'équipe `5` et la bonne réponse, ceci à la question numéro `1`.

Exercice 3.5 : Écrire une requête affichant toutes les distances entre la proposition de l'équipe `0` et la bonne réponse sur l'ensemble des questions, dont on affichera aussi le numéro.

Exercice 3.6 : Écrire une requête affichant la plus petite distance entre une proposition et la bonne réponse, ceci à la question numéro `2`.

Exercice 3.7 : Écrire une requête affichant toutes les équipes ayant eu exactement la bonne réponse à au moins une question et le nombre de fois où cela s'est produit par équipe.

La somme des distances à la bonne réponse n'est un critère de précision méritant un classement que si chaque équipe participe à chaque question, ce qui n'est pas forcément le cas, donc dans le doute, la question suivante fait intervenir la moyenne...

Exercice 3.8 : Écrire une requête affichant toutes les équipes et la moyenne des distances à la bonne réponse sur l'ensemble des questions, dans l'ordre croissant de cette moyenne.

En raison de la gestion difficile des points attribués en cas d'égalité, on ne simulera pas la compétition elle-même en SQL...