Deep Checker

Projet de statistiques ENS Rennes 2021

Arthur Correnson Igor Martayan Manon Sourisseau

Introduction

Dans ce projet, nous nous intéressons au jeu de dame. L'objectif est d'utiliser des méthodes d'apprentissage statistique pour construire une intelligence artificielle capable de jouer au jeu. L'idée maîtresse étant d'apprendre une heuristique évaluant la qualité des coups. Pour se faire, nous utiliserons des données récoltées lors de la simulation d'un grand nombre de parties de jeu de dame. Notons que ce projet est un projet de bout en bout, c'est à dire qu'il couvre (**1**) la génération des données, (**2**) le choix des modèles et enfin (**3**) l'entraînement des modèles et leur mise en pratique.

Règles du jeu de dames

Afin de simplifier la simulation et la modélisation des parties du jeu de dames, nous avons choisi d'utiliser une version affaiblie des règles officielles : - le plateau compte 64 cases (et non 100 comme dans la variante française) avec 12 pions noirs et 12 pions blancs - les pions se déplacent uniquement vers l'avant - la prise des pions adverses est obligatoire - les prises de pions peuvent être effectuées les unes à la suite des autres - la prise majoritaire est obligatoire : si plusieurs prises sont possibles, le joueur doit obligatoirement choisir celle qui prend le plus de pions adverses - on ne tient pas compte de la promotion des pions lorsqu'ils arrivent au bout du plateau - un joueur qui ne peut plus jouer de coups perd la partie

Pourquoi faire de l'apprentissage statistique ?

On pourrait se demander l'intérêt d'utiliser une approche statistique pour résoudre le jeu de dame plutôt qu'un algorithme d'exploration exhaustif. La raison est en fait assez simple : malgré la simplification des règles que nous avons adoptées, le nombre de coups possibles est beaucoup trop important. Il est très difficile de dénombrer le nombre exact de coups possibles dans cette variante du jeu de dames, mais un rapide estimation du nombre de plateaux possibles avec

12 pions noirs et 12 pions blancs nous donne

$$\binom{32}{12} \times \binom{32-12}{12} \approx 3 \times 10^{13}$$

ce qui est déjà trop pour une simulation exhaustive. Un article paru en 2007 estime le nombre de situations possibles à 5×10^{20} dans un jeu de dames classique (ce qui est probablement une surestimation dans notre variante affaiblie des règles). On voit donc tout l'intérêt d'avoir une approche statistique de ce problème.

1 - Génération des données et création d'un simulateur de jeu de dames

Les techniques d'apprentissage statistique utilisent, comme leur nom l'indique, des données comme matière première. Pour avoir les données nécessaires à l'apprentissage d'une bonne heuristique, nous avons commencé par créer notre propre simulateur de jeu de dames. Pour un bon apprentissage, il est primordial de disposer d'un grand nombre de données, sous peine d'obtenir une heuristique qui serait sur-spécialisée pour les quelques scénarios présentés par le jeu de données mais qui se généraliserait très mal à une partie quelconque. Afin de générer un volume important de données, il était donc nécessaire d'avoir un simulateur qui offre des grandes performances, mais également une représentation très compacte des données pour éviter de faire exploser la mémoire nécessaire à la simulation des parties et aux stockages des informations collectées. Nous avons donc fait le choix d'écrire le simulateur en langage C et de définir par la même occasion une représentation binaire astucieuse des données de jeu.

Sans rentrer dans les détails précis de l'implémentation du simulateur, en voici une présentation générale. Nous commençons par constater qu'un damier possède 64 cases dont seules 32 peuvent-être occupées par des pions. Par ailleurs, une case est vide, occupée par un pion blanc ou occupée par un pions noir. On peut donc modéliser un état du damier comme deux entiers 32 bits (que l'on combinera en un unique entier 64 bits) indiquant lesquelles des 32 cases sont occupées par les pions noirs (respectivement blancs). La simulation d'une partie complète est ensuite réduite au calcul d'opérations bit à bit sur l'état du damier.

Le simulateur garde une trace complète de chaque partie sous la forme d'une liste des états pris par le plateau. La séquence de chaque partie simulée est ensuite écrite dans un fichier texte permettant ainsi le traitement des données à posteriori.

La version finale du simulateur offre des performances remarquables. Plusieurs milliers de parties peuvent-être simulées en moins d'une seconde sur un ordinateur classique. Notons que l'essentiel du temps de simulation est consacré à l'écriture des données dans un fichier texte. Une amélioration possible serait d'écrire les données binaires directement dans un fichier afin d'éviter les temps d'entrées

sorties trop longs et les surcoûts liés à la conversion des données binaires vers des données textuelles. Nous avons tout de même fait le choix de conserver une représentation finale textuelle afin de faciliter la lecture des données par des outils externes (tableurs, scripts python, etc).

2 - Modèles et heuristiques

Rappelons que notre objectif est de construire une fonction (heuristique) permettant d'attribuer à un coup quelconque un score représentant intuitivement sa "qualité". Par souci de simplicité dans les modélisations, on s'intéressera uniquement aux coups de l'un des deux joueurs que l'on appellera joueur 1 (l'autre sera appellé joueur 2). A symétrie du damier près, les résultats obtenus pour noter les coups du joueurs 1 sont toujours généralisables aux coups du joueur 2. Nous présentons plusieurs approches pour déterminer une *bonne* heuristique.

Un premier système de notation des coups

Soit $\mathcal{DB}$ une collection de parties simulées et soit P_i un élément de $\mathcal{DB}$. P_i est une suite de damiers de laquelle on peut extraire l'ensemble C_i des coups c joués au cours de la partie. Un coup c pourra simplement être vu comme une paire de damiers représentant l'état du jeu avant et après le coup. Notons qu'au jeu de dames, un même coup ne peut se produire qu'une seule fois par partie au plus. Cette singularité est due au système de règles qui interdit de jouer en arrière.

Modélisation du problème

Si Ω est l'univers de tous les coups possibles du joueur 1 au jeu de dame, nous cherchons maintenant à trouver une fonction $h : \Omega \to [-1, 1]$ où $h(x) = 1$ si x est un excellent coup, -1 si c'est un très mauvais coup. Un premier squelette partiel de la fonction h peut-être construit à partir de nos données de simulation $\mathcal{DB}$ comme suit :

- Pour chaque partie $P_i \in \mathcal{DB}$ on calcul C_i l'ensemble des coups joués par le joueur 1 au cours de P_i
- On équipe chaque P_i d'une *fonction d'évaluation* notée $\|.\|_i : C_i \to [-1, 1]$ qui caractérise la distance à la victoire d'un coup dans P_i et définie comme : $\|c\|_i = \frac{1}{\sqrt{d(c)}}.v(c)$ où $d(c) \in \mathbb{N}$ est le nombre de coups qui séparent c de la fin de partie et $v(c) = 1$ (resp. -1) si le joueur 1 a gagné (resp. perdu). Intuitivement, plus c est proche de la victoire (resp. de la défaite), plus $\|c\|_i$ est grand (resp. petit).
- On considère à présent une fonction $w_i(c)$ telle que $w_i(c) = \|c\|_i$ si $c \in C_i$ et $w_i(c) = 0$ sinon.
- Pour chaque coup c apparaissant dans l'ensemble des parties de $\mathcal{DB}$, $h(c) = \frac{1}{N}\Sigma w_i(c)$ avec N le nombre de P_i tels que $w_i(c) \neq 0$ (c est un coup joué au cours de P_i).

Pseudo-apprentissage par régression aux k plus proches voisins

Une fois le squelette de h élaboré, on souhaite le généraliser afin de pouvoir noter un coup c quelconque n'apparaissant pas forcément dans $\mathcal{DB}$. Pour se faire, on utilise une méthode dite de régression aux k plus proches voisins (KNN) [**2**]. Cette méthode consiste essentiellement à généraliser le squelette de h de sorte que pour tout coup c hors du domaine de h, on attribue à c le score moyen des k coups les *plus proches* de c dans le domaine. La notion de proximité entre coups s'obtient naturellement à partir de la définition d'une distance. Nous choisissons ici d'introduire la distance entre coups $\langle ., . \rangle_{KNN} : \Omega \to \mathbb{N}$ définie comme la distance d'édition [**4**] entre deux damiers. Intuitivement, il s'agit de compter le nombre de pions qui diffèrent entre les deux coups.

Remarque Dans notre implémentation, les coups sont représentés sous forme de séquences de 128 bits. Le calcul de la distance entre deux coups se réduit alors à un calcul de XOR bit à bit suivi du comptage du nombre de bits à 1 :

$$\langle c_1, c_2 \rangle_{KNN} = \| c_1 \oplus c_2 \|_1$$

On notera $KNN(k, c)$ le score obtenu pour le coup c par régression aux k plus proche voisins.

Mise en pratique de KNN

Dans une situation de plateau donnée, notre heuristique permet calculer le prochain coup à jouer de la manière suivante :

On commence par déterminer l'ensemble des coups c_i jouables. Le prochain coup à jouer est alors $\arg\min_{c_i} \ KNN(k, c_i)$

Empiriquement, nous avons choisi une valeur de $k = 4$. Ce choix peut être sujet à discussion.

Résultats de l'approche KNN

Une fois la méthode KNN mise en place, nous avons observé les résultats sur des vraies parties. Pour se faire, nous simulons 50 parties dans lesquelles le joueur 1 choisit les meilleurs coups selon l'heuristique généralisée par KNN et le joueur 2 joue au hasard. Les résultats obtenus sont très peu convaincants :

Victoires du joueur 1 (KNN)	Victoires du joueur 2 (Aléatoire)
18	32

Ces résultats peuvent s'expliquer de plusieurs manières.

1. L'heuristique choisie est basée sur la distance à la victoire, or il n'est pas rare qu'un coup soit éloigné de la victoire en terme de *temps* mais qu'il

ait des conséquences très bénéfiques sur le long terme (par exemple, un placement anticipé permettant une grande séquence de captures plus tard).
2. Comme nous l'avons vu, la méthode KNN repose sur la *distance* entre deux coups. La distance que nous avons choisi rapproche naturellement les suites de plateaux très proches dans leur globalité. Or, certains coups très éloignés en terme d'état global du jeu correspondent en fait à des schémas très similaires d'un point de vu plus local. De même, deux coups très proches pour KNN peuvent en réalité correspondre à des situations très opposées.

Au regard des points énoncés, il est donc nécessaire d'affiner notre modélisation du problème.

Une meilleure modélisation

Afin de corriger les défauts de l'approche précédente, nous proposons un nouveau squelette d'heuristique ainsi qu'une nouvelle méthode d'apprentissage pour le généraliser.

Perceptron multicouche, apprentissage et évaluation par plateau

Un premier constat est qu'en évaluant un coup (donc une paire d'états de jeu), on met l'accent sur le **changement local** induit par le coup. Pourtant la notion de **distance** entre coups que nous proposions pour KNN rapproche uniquement les coups qui se ressemblent très fortement en terme d'état **global du jeu**. La contradiction de ces deux notions contribue à expliquer les mauvais résultats obtenus.

Afin de rendre plus homogène la modélisation, nous changeons complètement la définition de notre heuristique pour se concentrer cette fois sur la notation d'un état du jeu. La fonction de score est alors $h : \mathcal{D} \to [-1, 1]$ où $\mathcal{D}$ est l'ensemble des états possibles d'un damier.

Le score d'un damier d dans $\mathcal{DB}$ est alors calculé comme suit :

- Pour chaque partie $P_i \in \mathcal{DB}$ on on note D_i l'ensemble des états du damier vu par le joueur 1 au cours de P_i
- On équipe à nouveau P_i d'une fonction d'évaluation notée $\|.\|_i : D_i \to \mathbb{N}$ définie comme : $\|d\|_i = p(d).v(d)$ où $p(c) \in \mathbb{N}$ est le nombre de prises faites par le joueur 1 à partir de l'état courant d et $v(d) = 1$ si le joueur 1 a gagné et $v(c) = 0$ sinon
- On construit maintenant une fonction $w_i(c)$ telle que $w_i(d) = \|c\|_i$ si $d \in D_i$ et $w_i(d) = 0$ sinon
- Pour chaque état de damier d apparaissant dans l'ensemble des parties de $\mathcal{DB}$, $h(d) = \frac{1}{N}\Sigma w_i(d)$ avec N le nombre de parties P_i tels que $w_i(d) \neq 0$ (d est l'un des états pris par le damier dans P_i)

La généralisation de h à l'ensemble des états possibles $\mathcal{D}$ (n'apparaissant pas nécessairement dans les parties de $\mathcal{DB}$) est alors faites grâce à un réseau de

neurones de type perceptron multicouche [**1**] dont la structure est la suivante :

- Les entrées sont des vecteurs de 64 bits représentant un unique damier
- Le coeur du réseau consiste en 4 couches intermédiaires composées respectivement de 64, 64, 32, et 16 noeuds
- Tous les noeuds du réseau sont doté d'une fonction d'activation *relu*
- Nous souhaitons faire de la régression et non de la classification donc la sortie du réseau est de dimension 1 et résulte d'une combinaison linéaire des 16 sorties de la dernière couche puis d'une application de la fonction *relu*

Nous construisons le réseau grâce aux outils fournis par le module python Scikit-learn [**3**]. L'apprentissage est lancé pour une durée de 500 cycles sur l'ensemble des données $\mathcal{DB}$ étiquetées grâce à la fonction partielle h.

Afin d'éviter des problèmes d'*overfitting* (le réseau devient sur-spécialisé pour caractériser exactement les données de $\mathcal{DB}$), nous avons séparé le jeu de données en un jeu d'entraînement et un jeu de test.

Résultats

Les résultats obtenus grâce au nouveau modèle sont bien plus intéressants. Encore une fois, nous avons simulé 50 parties où le joueur 1 choisit les meilleurs coups selon l'heuristique apprise par le réseau et le joueur 2 est aléatoire. Les résultats sont clairs :

Victoires du joueur 1 (MLP)	Victoires du joueur 2 (Aléatoire)
50	0

Coefficients de détermination

Pour mesurer la qualité des prédictions de nos réseaux de neurones, on peut calculer le coefficient de détermination R^2 [**5**] sur un échantillon de test pour chacune des heuristiques. Ce coefficient est défini par

$$R^2 = 1 - \sum_{i=1}^{n} \frac{y_i - \hat{y}_i}{y_i - \bar{y}}$$

où n désigne le nombre de mesures, y_i désigne la mesure numéro i, $\hat{y}_i$ désigne la prédiction numéro i et $\bar{y}$ désigne la valeur moyenne. Autrement dit, cela revient à mesurer le rapport entre l'erreur quadratique moyenne et la variance des mesures.

Heuristique	Coefficient de détermination
Distance à la victoire	0.46
Nombre de prises	0.68

Résultats finaux

Pour avoir une vue globale des différentes méthodes employées lors de ce projet, nous avons croisé les differentes heuristiques et les différents modèles d'apprentissage pour comparer leur efficacité. En tout, deux heuristiques sont proposées : une heuristique de coups qui évalue la *distance à la victoire* et une heuristique de plateau qui évalue le *nombre de prises jusqu'à la victoire*. Nous compararons l'efficacité de ces deux heuristiques après généralisation par KNN et par MLP. L'apprentissage est fait à partir du même jeu de 50 000 parties de jeu de dame dans tous les cas. L'évaluation est faite sur 50 parties contre un joueur purement aléatoire.

Heuristiques + Models	Victoires	Défaites
Distance à la victoire + KNN	18	32
Nombre de prises + KNN	36	14
Distance à la victoire + MLP	39	11
Nombre de prises + MLP	50	0

Nous remarquons que pour les deux heuristiques, la méthode de généralisation par MLP donne des meilleurs résultats. Notons également que le choix de l'heuristique de départ impacte beaucoup l'efficacité et l'heuristique de *nombre de prises* donne des résultats significativement meilleurs quelque soit la méthode de généralisation employée.

Conclusion

Au cours de ce projet, nous avons pu mettre en oeuvre différentes techniques d'apprentissage statistique de bout en bout. Les résultats obtenus nous ont permis de raisonner sur l'efficacité des différents modèles types rencontrés en apprentissage statistique. Nous avons également constaté que dans le cadre des jeux, apprendre une bonne heuristique par apprentissage supervisé nécessite de construire une première fonction de score qui conditionne fortement la qualité des résultats. Plusieurs pistes sont possibles pour améliorer nos résultats et réduire l'importance du choix de l'heuristique initiale. Une première possibilité serait de travailler sur des heuristiques initiales plus fines et de comparer leur efficacité sur un grand nombre de parties. Une seconde direction d'amélioration serait d'utiliser des techniques par renforcement : au lieu de proposer une heuristique de départ puis de la généraliser, on pourrait ainsi générer de toute part une heuristique qui en pratique se comporte bien. Notons que les approches supervisées et par renforcement ne sont pas incompatibles et les résultats obtenus par apprentissage supervisées pourraient servir de base à un apprentissage par renforcement. Une dernière possibilité serait d'utiliser des techniques d'apprentissage plus spécialisées pour ce type de problème telles que des réseaux de neurones convolutifs ou des arbres de Monte-Carlo.

Références

[**1**] *Multi Layer Perceptron*, Wikipédia

[**2**] *K-nearest neighbors algorithm*, Wikipédia

[**3**] *Scikit-learn: Machine Learning in Python* Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., **Journal of Machine Learning Research**, 2012

[**4**] *Levenshtein distance*, Wikipédia

[**5**] *Coefficient of determination*, Wikipédia