

RAPPORT DE PROJET

Jeu d'Awalé

RESUME

L'Awélé est un jeu traditionnel africain. Le jeu connaît de nombreuses appellations et de nombreuses variantes. C'est un jeu de stratégie combinatoire abstrait qui voit s'affronter deux joueurs jouant à tour de rôle, sans informations cachées et sans hasard. Ce projet considère la variante la plus utilisée du jeu.

Delaby Pierre & Nivoix Ludovic

I.F.T.D.

Table des matières

I – Rappel du sujet	2
II – Réflexion	2
II.1 Adaptation des bots	2
II.2 Recherche d'informations	2
II.3 Championnat entre tous nos bots.....	4
III Création	6
III. 1 Ajout de données	6
III.1.1 Réflexion.....	6
III.1.2 DataSupp	6
III.1.3 Adaptation.....	7
III.2 LDA	7
III.3 Multibot.....	8
III.3.1 Combinaison des prédictions	8
III.3.2 Création du bot.....	9
III.4 Programmation	9
ANNEXE	10
Capture du championnat.xlsx.....	10
Affichage de l'évaluation.....	11
ACP	11
ACP	11

I – Rappel du sujet

L'Awalé est un jeu traditionnel africain. Le jeu connaît de nombreuses appellations et de nombreuses variantes. C'est un jeu de stratégie combinatoire abstrait qui voit s'affronter deux joueurs jouant à tour de rôle, sans informations cachées et sans hasard. Ce projet considère la variante la plus utilisée du jeu. Le but de ce projet est de réaliser un modèle de données utilisant un ou plusieurs algorithmes de fouilles de données afin de pouvoir choisir, le plus efficacement possible, le meilleur coup à jouer en fonction d'une situation données. Pour cela, le modèle disposera d'une base de données de coups joués (aweale.data).

II – Réflexion

Pour débiter le projet, nous avons suivi plusieurs pistes de réflexions, qui sont les suivantes :

- Création de bots utilisant d'autres algorithmes de fouilles de données (en s'inspirant de nb)
- Rajouter des données sur le jeu d'essai aweale.data
- Recherche d'informations sur d'autres algorithmes de fouilles de données
- Création d'un bot utilisant plusieurs algorithmes de fouilles de données

II.1 Adaptation des bots

À ce jour, nous avons adapté plusieurs bots utilisant divers algorithmes de fouille de données. Nous avons pris principalement, comme source d'inspiration, le bot utilisant l'algorithme Naïve Bayes. Bot que nous appellerons par la suite nb. Voici la liste des bots obtenues :

- Bot utilisant l'algorithme ADL
- Bot utilisant l'algorithme RL
- Bot utilisant l'algorithme KNN
- Bot utilisant l'algorithme CART
- Bot utilisant l'algorithme MLP
- Bot utilisant l'algorithme SVM

II.2 Recherche d'informations

Afin de pouvoir proposer le meilleur bot possible répondant à la thématique du sujet, nous avons effectué quelques recherches sur l'Awalé. Nous avons tout d'abord recherché des informations sur le jeu et ses règles. Ensuite, dans un souci de compréhension total du sujet, nous avons fabriqué un plateau permettant de jouer à l'Awalé à l'aide de pâtes pour les billes et un calendrier de l'avent pour le plateau.



Nous avons également été à la bibliothèque pour rechercher diverses informations concernant le jeu de l'Awalé et des algorithmes pour obtenir le meilleur score possible. Nous avons donc lu le livre, *L'Awalé*, de Serge Mbarga Owona. Toutefois, les recherches dans ce livre n'ont pas été concluantes. En effet, ce livre ne détaille aucune solution avec un jeu de données d'apprentissage. Il détaille uniquement des algorithmes de résolution. C'est-à-dire, qu'il évoque toutes les combinaisons possibles pour gagner en fonction de la première case jouée.

Les recherches à la bibliothèque étant infructueuses, nous avons donc fait quelques recherches sur internet et nous avons trouvé un algorithme intéressant. Il s'agit de l'algorithme suivant :

- L'algorithme **Négamax**

Cet algorithme s'applique à la théorie des jeux, pour les jeux à deux joueurs à somme nulle. Cela consiste à minimiser la perte maximum. Ce qui n'est pas le cas dans le jeu de l'Awalé, car nous essayons de maximiser le gain. (En effet, nous pouvons minimiser le risque mais cela ne veut pas forcément dire que nous maximisons le gain également. Il peut exister plusieurs coups ayant une perte nulle mais le gain, si il existe, n'est pas forcément le même.) Dans notre cas, cet algorithme n'est pas applicable de façon direct, malheureusement.

Nous avons également tenté différentes fonctions d'évaluations vues en TP mais sans succès car le nombre de dimensions est bien trop important pour pouvoir les interpréter. Les captures relatives à ces « évaluations » sont présentes en annexe.

II.3 Championnat entre tous nos bots

Après avoir conçu un certain nombre de bot nous avons décidé de les faire se battre tous ensemble afin de pouvoir déterminer le ou les meilleurs bots. De cette manière, nous avons pu voir lesquels étaient intéressants de garder pour la création du bot multibot.

Algo/Alog	NB	NB2	pedro	pedrobis	botADL1	adlsum	adl12	afd1	RL	knn1	cart1	cartbis	mlpludo	svm	svm2	mplbis	adlKiller	multibot	GAGNE(rouge)	PERDU (vert)
NB																			0	0 NB
NB2	nb																		0	1 NB2
pedro	pedro	nb2																	1	1 pedro
pedrobis	nb	nb2	égalité																0	2 pedrobis
botADL1	botADL1	nb2	botADL1	pedrobis															2	2 botADL1
adlsum	nb	nb2	adlsum	pedrobis	égalité														1	3 adlsum
adl12	adl12	adl12	adl12	adl12	adl12	adl12													6	0 adl12
afd1	égalité	nb2	pedro	afd1	botADL1	adlsum	adl12												1	5 afd1
RL	nb	RL	pedro	pedrobis	botADL1	adlsum	RL	RL											3	5 RL
knn1	nb	nb2	pedro	knn1	knn1	knn1	adl12	afd1	RL										4	5 knn1
cart1	nb	cart1	pedro	pedrobis	cart1	cart1	adl12	cart1	cart1	cart1									6	4 cart1
cartbis	cartbis	cartbis	pedro	pedrobis	botADL1	adlsum	adl12	cartBis	cartBis	cartBis	cartBis								6	5 cartbis
mlpludo	mlpludo	nb2	mlpludo	pedrobis	botADL1	adlsum	mlpludo	RL	RL	mlpludo	cart1	mlpludo							6	6 mlpludo
svm	bug	bug	bug	bug	bug	bug	bug	bug	bug	bug	bug	bug	bug						0	0 svm
svm2	svm2	svm2	svm2	svm2	botADL1	adlsum	adl12	svm2	svm2	svm2	svm2	cartBis	mlpludo	bug					8	5 svm2
mplbis	mplbis	nb2	mplbis	pedrobis	botADL1	adlsum	mplbis	mplbis	RL	mplbis	cart1	mplbis	égalité	bug	mplbis				8	5 mplbis
adlKILLER	adlkiller	adlkiller	adlkiller	adlkiller	adlkiller	adlkiller	adlkiller	adlkiller	adlkiller	adlkiller	cart1	adlkiller	adlkiller	bug	adlkiller	adlkiller			15	1 adlkiller
multibot	multibot	multibot	multibot	multibot	multibot	multibot	multibot	multibot	multibot	multibot	cart2	multibot	multibot	bug	multibot	multibot	adlkiller			
GAGNE (vert)	6	7	5	7	6	6	5	1	2	0	3	1	1	0	0	0	0			
PERDUS (rouge)	8	7	7	5	4	4	4	7	5	6	2	3	1	0	2	1	0			
colonne gagne																				
colonne perds																				
égalité																				

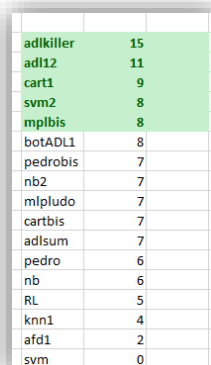
Capture du fichier Excel de championnat

Nous avons obtenu le classement suivant :

	GAGNE	PERDU	SOMME
NB	6	8	14
NB2	7	8	15
pedro	6	8	14
pedrobis	7	7	14
botADL1	8	6	14
adl12	11	4	15
afd1	2	12	14
RL	5	10	15
knn1	4	11	15
cart1	9	6	15
cartbis	7	8	15
mlpludo	7	7	14
svm	0	0	0
svm2	8	7	15
mplbis	8	7	15
adlsum	7	7	14
adlkiller	15	1	16

Capture des résultats du championnat

De la même manière, nous avons pu sélectionner nos différents bots pour le multibot que nous avons créé :



adkiller	15	
adl12	11	
cart1	9	
svm2	8	
mplbis	8	
botADL1	8	
pedrobis	7	
nb2	7	
mplludo	7	
cartbis	7	
adlsum	7	
pedro	6	
nb	6	
RL	5	
knn1	4	
afd1	2	
svm	0	

Capture du classement des bots selon les scores calculés

III - Création

Pour la partie concernant la création, nous avons combiné les différentes techniques mises au point durant la phase de réflexions

III. 1 Ajout de données

III.1.1 Réflexion

Afin d'ajouter de préciser l'analyse, nous avons décidé d'ajouter des données. Nous avons commencé avec des données simples comme les sommes des graines par joueur, ou le nombre de cases vides. Nous nous sommes rapidement rendu compte qu'il y aurait de nombreuses possibilités d'ajout de données, et il fallait trouver un mécanisme paramétrable pour que cet ajout ne soit pas trop pénible, et qu'il soit compatible avec tous les algorithmes.

Ainsi, après plusieurs tentatives avec des notations pointées et des listes, nous avons opté pour la création d'une classe : *dataSupp*.

III.1.2 DataSupp

La classe *dataSupp* contient :

1. Une fonction, d'ajout de données, qui a comme entête
 - Paramètre : Les données originales
 - Paramètre : La position d'ajout
 - Retourne : les données complétées
2. Une valeur numérique, qui représente le décalage

Les instances de cette classe sont des objets avec une fonction qui, pour chaque ligne, calcule de nouveaux attributs, et les ajoute à la position voulue. Le nombre de colonnes supplémentaires est stocké dans l'attribut *decal*, afin de pouvoir effectuer un second ajout sans écraser le premier.

Ces fonctions nomment aussi les colonnes afin de pouvoir les reconnaître. Elles sont assez simple, une dizaine de ligne en général.

Actuellement, nous en avons 9 :

Nom	Action	Par	Décalage en colonnes
Vide	Ajoute le nombre de cases vides	Joueur	2
Somme	Ajoute la somme des graines	Joueur	2
Sum1ou2	Ajoute le nombre de cases contenant 1 ou 2 graines	Joueur	2
Sum1ou2matrice	Ajoute une valeur booléenne qui décrit si la case contient 1 ou 2 graine	Case	12
unTour	Ajoute le nombre de case qui peuvent faire plus d'un tour	Joueur	2
posMax	Ajoute la position de la première case qui contient le maximum de graines	Joueur	2

nbVidesAvantPleines	Le nombre de cases vides avant la première pleine	Joueur	2
Bidoua	Fonction créée selon les principes du livre sur l'awalé (une sorte d'indice)	Case	12
nbGagne	Le nombre de graines gagnées si la case est joué	Case	12

III.1.3 Adaptation

Elles s'ajoutent très facilement à une fonction, il suffit de définir une liste d'objets de type `dataSupp`. Dans les fonctions, dès la réception des données, il faut les compléter à l'aide de la fonction *completeData*.

completeData est une fonction prend en paramètre les données et la liste des fonctions. Elle recopie les données dans une matrice plus grande, applique les fonctions une par une en décalant leur paramètre de position d'ajout grâce à l'addition successive des propriétés de décalage. Elle ajoute ensuite les données de fin, s'il y en a.

Ensuite, faut récupérer le décalage total apporté par l'ajout des données, et l'additionner aux indices utilisés dans les fonctions de classification (14 devient 14+decalage, ...).

Ainsi, on peut rapidement convertir un bot naiveBayes pour qu'il ajoute des données paramétrées. Il est très facile d'ajouter plusieurs classes de `dataSupp` dans la liste pour tester les résultats.

Par exemple, le bot `adlsupp` est un bot qui utilise LDA et qui lui ajoute des données. Afin de pouvoir créer de nombreux bots de ce type, nous avons rajouté la liste des fonctions en paramètres de la fonction création du modèle et de celle d'exécution. Ainsi, pour créer un bot `adlsupp` avec des données supplémentaires, il suffit de :

- créer une liste des classes `dataSupp` que l'on veut utiliser (*somme*, *sum1Ou2*, ...)
- créer un modèle en utilisant `dpln.adlsupp.create.model` en lui passant la liste
- créer la fonction d'évaluation (le bot) en lui passant le modèle et la liste

Il est donc possible de créer de nombreux bots `adlsupp` en simplement quelques lignes

III.2 LDA

Nous avons testé l'algorithme LDA en nous inspirant du fonctionnement de l'algorithme NB. IL a montré de très bons résultats face aux bots que nous avions à ce moment, ainsi nous avons décidé de lui ajouter des données.

Avec simplement les données de *somme*, le bot était encore plus puissant. Nous avons donc testé avec de nombreux attributs, en faisant combattre deux bots LDA ensemble, chacun avec différentes fonctions.

Nous avons aussi utilisé les fonctions d'évaluation afin de tester de nombreuses configurations. Il en est ressorti que plus on ajoutait de données, mieux il s'en sortait. Mais il a fallu bien choisir les données : en effet, quelques fonctions d'ajout rendaient le bot très faible.

Nous avons terminé avec la combinaison des fonctions : *vide*, *somme*, *posMax*, *nbGagne*.

Il a remporté tous ses matchs, sauf celui contre NB2, d'où son surnom : *AdlKiller*

III.3 Multibot

Une fois le championnat terminé, nous avons compté les points et récupéré les 5 bots les plus forts afin de les utiliser pour créer un bot qui les combine.

Notre but était de combiner les prédictions des 5 bots vainqueurs, en suivant l'algorithme de combinaison de classifieurs vu en cours :

- pour chaque bot, on prédit le coup joué
- on retourne le coup le plus joué

Nous avons procédé en 2 étapes : combinaisons des prédictions, puis création du multibot

III.3.1 Combinaison des prédictions

Afin de combiner les prédictions, nous avons écrit une fonction (*getCoupFromCoups*) qui prends en paramètre les 5 coups des 5 bots (impaire, plus facile pour départager), et qui retourne 1 coup. La combinaison se fait en plusieurs étapes :

- Addition des matrices de résultats pour avoir la somme pour chaque coup
- Récupération du coup le plus joué
- Comptage du nombre de fois où il est présent dans la matrice de résultat total
 - Si il n'est présent qu'une seule fois (pas d'égalité entre les coups), on crée une nouvelle matrice de coup qui le retourne
 - Sinon, on enlève 2 résultats de 2 bots afin qu'il n'en reste plus que 3 (nombre impair, à nouveau), ceux des 3 bots les plus forts, et on rappelle la fonction
 - Si cela ne fonctionne toujours pas (toujours égalité des coups), on retourne le résultat du premier bot – le plus fort.

Ainsi, nous avons créé une fonction légèrement récursive, qui combine les résultats de 5 bots

Voici un exemple d'exécution

Récupération des coups pour chaque bot. Ici, bot1 et 2 jouent C1, bot2 joue C2, bot3 et 4 jouent c5, bot5 joue C6

	row.names	C1	C2	C3	C4	C5	C6
1	bot1	1	0	0	0	0	0
2	bot2	1	0	0	0	0	0
3	bot3	0	1	0	0	0	0
4	bot4	0	0	0	0	1	0
5	bot5	0	0	0	0	1	0

Calcul du score pour chaque coup

	C1	C2	C3	C4	C5	C6
1	2	1	0	2	1	0

Calcul du score maximum

2

Calcul du nombre de fois où il est présent

2 fois

Il est présent plus d'une fois, donc on supprime les résultats des bots 4 et 5.
On rappelle la fonction

	row.names	C1	C2	C3	C4	C5	C6
1	bot1	1	0	0	0	0	0
2	bot2	1	0	0	0	0	0
3	bot3	0	1	0	0	0	0
4	bot4	0	0	0	0	0	0
5	bot5	0	0	0	0	0	0

Calcul du score pour chaque coup

	C1	C2	C3	C4	C5	C6
1	2	1	0	0	1	0

Calcul du score maximum

2

Calcul du nombre de fois où il est présent

1 fois

Création du vecteur résultat

	C1	C2	C3	C4	C5	C6
1	1	0	0	0	0	0

III.3.2 Création du bot

Une fois la fonction créée, il fut facile de créer un bot qui importe les 5 meilleurs bots, calcule leurs coups et retourne le résultat de la fonction `getCoupFromCoups()`. Ce bot c'est avéré très efficace contre la grande majorité de tous les autres, mais plus faible que `AdlKiller`.

III.4 Programmation

Nous avons essayé de programmer de la manière la plus propre possible. Mais durant notre phase de développement, nous avons beaucoup dupliqué les bots pour les tester avec d'autres algorithmes, ce qui résulte en de nombreuses redondances de code dans les différents bots. La fonction d'exécution, principalement, est très généralement la même. Nous pourrions l'enlever de tous nos bots (quand elle est générique) et la remplacer par une autre.

De plus, de nombreuses lignes pourraient être factorisées en fonctions qui serait appelées à différentes endroits, mais l'IDE (R studio) ne se prête pas bien à la refactorisation. Ainsi, nous ne pouvons pas suivre les étapes habituelles (tests de code -> refactorisation -> test -> livraison) car la refactorisation est impossible. Ainsi, il nous faudrait tout remplacer partout, tout re-tester, et nous occuper des erreurs (qui sont très obscures dans R-Studio). Nous avons donc factorisé quelques fonctions importantes, et n'avons pas eu le temps de factoriser toutes les fonctions. C'est aussi pour cela que de nombreux commentaires sont dupliqués.

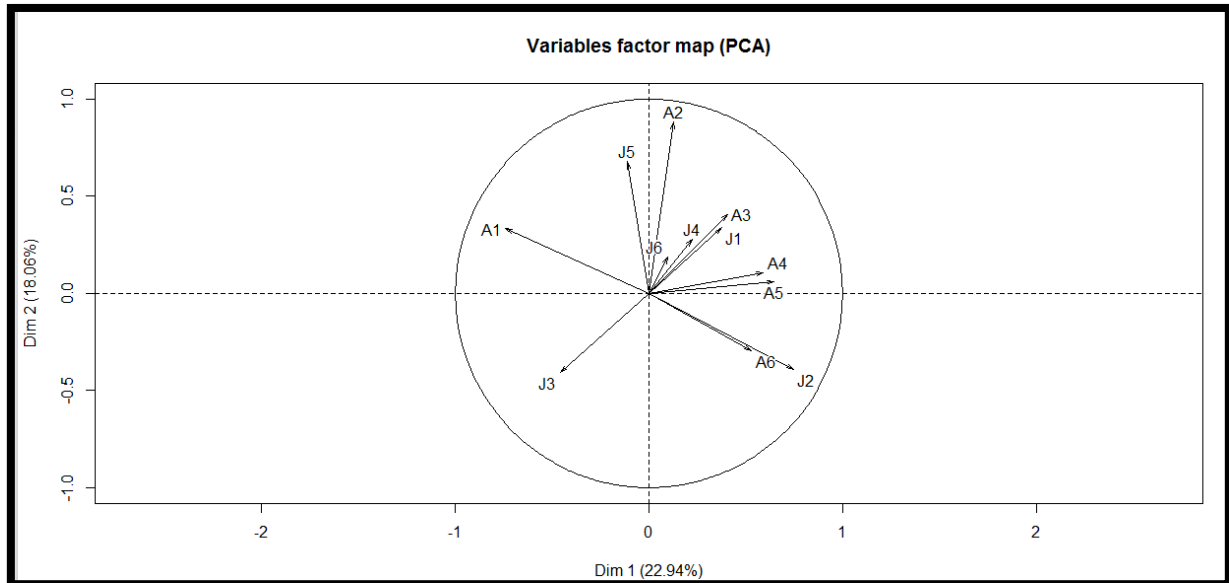
Nous avons tout de même essayé d'adopter des nomenclatures pour les commentaires et les noms des bots, afin de nous retrouver dans la totalité des fichiers.

Capture du championnat.xlsx

10/11

Affichage de l'évaluation

ACP



ACP

