TELECOM Nancy

Intelligence artificielle

Mini-projet : La ruée vers l'or

Réalisé par :
KESLICK Malaury

Professeur :
BOUGRAIN Laurent

2020

1 Introduction

Ce rapport présente le travail effectué dans le cadre du mini-Projet du module d'Intelligence Artificielle dispensé par Mr. Laurent Bougrain aux élèves de deuxième année de TELECOM Nancy. Le but de ce projet est de concevoir et d'entraîner un ou plusieurs modèles prédictifs pour prédire au mieux la présence ou l'absence d'or sur des sites géologiques. Pour cela, nous avions à disposition une base de données de plus de 1600 sites, permettant la partie apprentissage. Une fois un modèle retenu, il nous fallait alors l'utiliser sur les 400 sites proposés, afin de déterminer lesquels étaient les plus susceptibles de recueillir de l'or.

Je vais donc vous présenter dans ce rapport mes choix d'implémentation et mes résultats, avant de conclure sur ce travail.

Table des matières

2 Choix d'implémentation

2.1 Langage et bibliothèques utilisées

Pour ce travail j'ai choisi d'utiliser le langage Python, puisque c'est un langage avec lequel je suis plutôt très familière. J'ai utilisé les bibliothèques *pandas* et *sklearn*, la première permettant de faire du traitement de données, et la seconde de faire de la fouille de données. Les fonctionnalités utilisées seront explicitées dans les parties suivantes.

2.2 Traitement des données

J'ai donc utilisé la bibliothèques *pandas* afin de récupérer les données des fichiers fournis, puis de finalement stocker les résultats obtenus. La fonction *read_csv* de cette bibliothèque aura été très pratique puisqu'elle permet de parser directement nos fichiers texte selon ses virgules, de supprimer la première ligne (nom des colonnes), et de nommer les colonnes résultantes dans la matrice obtenue. Par exemple, l'appel :

train = pd.read_csv("gisementLearn.txt",skiprows=1,header=None,names=["X_COORD", "Y_COORD","BOUGUER","GRTOPISO","PROF_MOHO","S_DEPTH","TOPO_ISO","BENPENTE","DIST_SEIS","DIST_SUBS","DIST_VOL","NBR_SEISM","DIST_112","DIST_135","DIST_157","DIST_180","DIST_22","DIST_45","DIST_67","DIST_90","AGE","ROCK","OR"])

permet de récupérer un tableau de tableau contenant toutes les données de *gisementLearn.txt*, et l'appel *train["OR"]* renvoie uniquement la dernière colonne de ce tableau.

Un autre traitement a alors été nécessaire afin de pouvoir mener à bien le processus de fouille des données. En effet les types des valeurs du tableau précédemment récupéré ne sont pas homogènes. Les colonnes "AGE", "ROCK" et "OR" sont des chaînes de caractères, lorsque tout le reste est quantitatif. Puisque ces trois attributs n'ont respectivement que 3, 6 et 2 valeurs possibles, j'ai donc décidé de les remplacer par des entiers ($\{1,2,3\}$,$\{1,2,3,4,5,6\}$ et $\{0,1\}$). Le traitement inverse est opéré sur les résultats obtenus avant de les inscrire dans le fichier final.

Enfin, une fois la prédiction sur les données de *gisementTestNolabel.txt* effectuée, j'utilise la fonction *to_csv* afin de récupérer les résultats dans le fichier *gisementTest.txt*.

2.3 Fouille de données

La fouille des données fournies a été faite grâce à la librairie *sklearn*. J'ai tout d'abord séparé les données fournies en une partie apprentissage (80%) et une partie test (20%). J'ai alors choisi de tester deux modèles prédictifs dont je compare ensuite l'efficacité afin de renvoyer le meilleur résultat possible.

Le premier modèle est celui de l'arbre de décision, gérée par la partie *DecisionTreeClassifier* de *sklearn*. Je l'ai testé pour des valeurs de *min_samples_split* allant de 2 à 49, afin d'avoir l'un des meilleurs résultats possibles. Vous trouverez un exemple d'exécution dans la partie suivante.

Le second modèle est celui de la forêt aléatoire, gérée par *RandomForestClassifier*. Encore une fois, min_samples_split évolue entre 2 et 49, et j'ai choisi de fixer n_estimators à 100, afin d'avoir quelque chose de relativement complet sans rendre le programme trop chronophage. Vous trouverez également un exemple d'exécution dans la partie suivante.

3 Résultats

3.1 Arbre de décision

```
Tree accuracies :
[0.8929663608562691, 0.8776758409785933, 0.8623853211009175, 0.8685015290519877, 0.8715596330275229, 0.8685015290519877,
 0.8746177370030581, 0.8623853211009175, 0.8654434250764526, 0.8623853211009175, 0.8562691131498471, 0.8593272171253823,
 0.8623853211009175, 0.8623853211009175, 0.8593272171253823, 0.8685015290519877, 0.8654434250764526, 0.8685015290519877,
 0.8654434250764526, 0.8654434250764526, 0.8654434250764526, 0.8654434250764526, 0.8654434250764526, 0.8654434250764526,
 0.8654434250764526, 0.8654434250764526, 0.8623853211009175, 0.8654434250764526, 0.8685015290519877, 0.8685015290519877,
 0.8654434250764526, 0.8715596330275229, 0.8715596330275229, 0.8685015290519877, 0.8685015290519877, 0.8685015290519877,
 0.8715596330275229, 0.8715596330275229, 0.8746177370030581, 0.8776758409785933, 0.8776758409785933, 0.8776758409785933,
 0.8868501529051988, 0.8776758409785933, 0.8776758409785933, 0.8899082568807339, 0.8899082568807339, 0.8899082568807339]
```

FIGURE 1 – Accuracy des arbres décisionnels pour min_samples_split allant de 2 à 49

3.2 Forêt aléatoire

```
Forest accuracies :
[0.908256880733945, 0.9051987767584098, 0.9051987767584098, 0.9143730886850153, 0.9051987767584098, 0.9113149847094801,
0.9174311926605505, 0.9113149847094801, 0.9051987767584098, 0.9051987767584098, 0.908256880733945, 0.9143730886850153, 0
.9113149847094801, 0.908256880733945, 0.9113149847094801, 0.9174311926605505, 0.9174311926605505, 0.9174311926605505, 0.
9113149847094801, 0.9113149847094801, 0.9174311926605505, 0.9143730886850153, 0.9143730886850153, 0.9174311926605505, 0.
9174311926605505, 0.9174311926605505, 0.9204892966360856, 0.9204892966360856, 0.9204892966360856, 0.9204892966360856, 0.
9174311926605505, 0.9174311926605505, 0.9235474006116208, 0.9204892966360856, 0.9204892966360856, 0.9204892966360856, 0.
9174311926605505, 0.9204892966360856, 0.9204892966360856, 0.9204892966360856, 0.9235474006116208, 0.9235474006116208, 0.
9235474006116208, 0.9204892966360856, 0.9235474006116208, 0.9204892966360856, 0.9204892966360856, 0.9204892966360856]
```

FIGURE 2 – Accuracy des forêts aléatoires pour min_samples_split allant de 2 à 49

3.3 Résultat final

Ci-dessous sont présentés les résultats de deux simulations, et un tableau récapitulatif des résultats d'exécution de 15 simulations différentes.

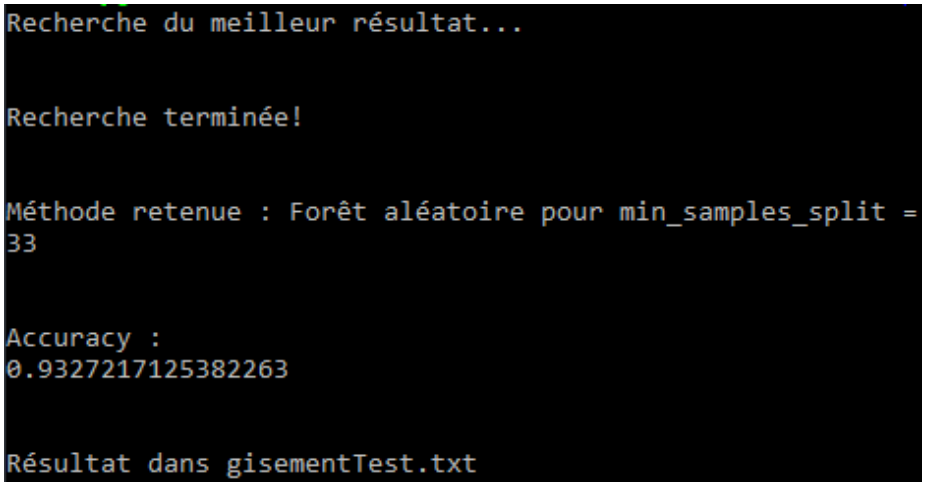

FIGURE 3 – Résultat de simulation

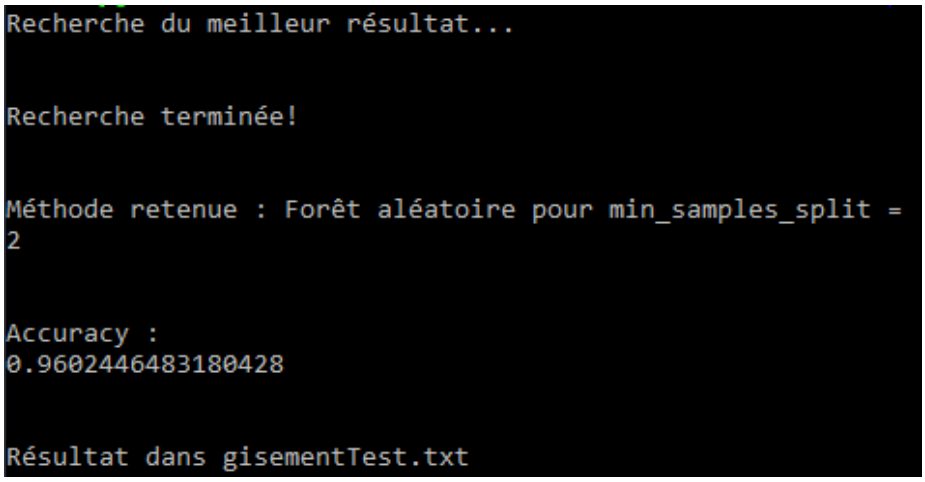

FIGURE 4 – Autre résultat de simulation

Execution	Méthode retenue	min_samples_split	Accuracy
1	Forêt aléatoire	33	0,932721713
2	Forêt aléatoire	3	0,920489297
3	Forêt aléatoire	7	0,923547401
4	Forêt aléatoire	13	0,911314985
5	Forêt aléatoire	2	0,960244648
6	Forêt aléatoire	27	0,957186544
7	Forêt aléatoire	9	0,902140673
8	Forêt aléatoire	3	0,93883792
9	Forêt aléatoire	6	0,920489297
10	Forêt aléatoire	4	0,926605505
11	Forêt aléatoire	36	0,935779817
12	Forêt aléatoire	13	0,917431193
13	Forêt aléatoire	5	0,917431193
14	Forêt aléatoire	8	0,923547401
15	Forêt aléatoire	14	0,95412844
	Moyennes :	12,2	0,929459735

FIGURE 5 – Tableau récapitulatif de plusieurs simulations

On remarque donc que le modèle des forêts aléatoires est toujours plus concluant que le modèle des arbres de décision. Néanmoins, l'accuracy moyenne obtenue n'est vraiment pas assez satisfaisante pour pouvoir dire que les résultats obtenus sont fiables.

4 Conclusion

Pour conclure, j'ai pu relever que les forêts aléatoires étaient plus efficaces que les arbres de décision. Néanmoins, je regrette de ne pas avoir eût plus de temps à consacrer à ce projet en ces temps difficile, et j'aurais aimé pouvoir m'intéresser à d'autres modèles comme les K-means par exemple.