Sylvain Combettes

Julien Muller

Etudiants en Master 2
Ecole des Mines de Nancy
Département Ingénierie Mathématique

sylvain.combettes@mines-nancy.org
julien.muller@mines-nancy.org

https://github.com/sylvaincom/C-for-data-science

Rapport de projet C/C++ | Machine Learning & Deep Learning

Dernière modification : 26 janvier 2020

Table des matières

Introduction

Notre projet porte sur le Machine Learning ainsi que le Deep Learning. En effet, nous sommes tous les deux en 3A au sein du département IM et nous souhaitons tous deux devenir Data Scientists. Nous avions choisi cet électif afin d'apprendre un nouveau langage de programmation et de pouvoir l'appliquer en particulier à la Data Science. La plupart des Data Scientists travaillent sur Python ou R et peu de librairies ont été développées en C, mais la force du langage C est sa rapidité d'exécution, d'où notre intérêt.

Le but de ce rapport est de compléter notre code source en C/C++ : apporter quelques éléments théoriques basiques sur le Machine Learning ainsi que quelques explications sur la structure des codes.

Comme décrit sur la page web de cet électif [1], les trois objectifs de ce cours sont :

- Comprendre de manière abstraite le mécanisme d'exécution des programmes.
- Être capable de comprendre le fonctionnement d'un programme.
- Être capable d'écrire ou de modifier un programme écrit en C ou C++.

Tous ces points ont été traités en cours/TP et dans ce projet, nous nous sommes focalisés sur le dernier point. En effet, nous avons écrit quelques programmes C/C++ nous-mêmes de A à Z, mais nous nous sommes également inspirés de codes trouvés sur internet que nous avons modifiés selon nos objectifs personnels.

L'ensemble de nos codes est déposé sur GitHub : `https://github.com/sylvaincom/C-for-data-science`.

1 L'algorithme des K-moyennes (ou *K-means*)

Pour les codes et explications supplémentaires, se reporter au dépôt GitHub : `https://github.com/sylvaincom/C-for-data-science`, dans le dossier `kmeans`. Nous avons codé cet algorithme de A à Z.

Notre rapport commence par un algorithme d'apprentissage non supervisé : les K-moyennes.

Une visualisation du fonctionnement de l'algorithme est donnée figure 1.

L'algorithme des K-moyennes est l'algorithme de clustering (regroupement) le plus connu. Notre but est de former des groupes de points "proches" entre eux, qui se "ressemblent". Il faut visualiser chaque étape avec la figure 1. Les étapes sont :

1. Nous sélectionnons d'abord un nombre K ($K = 3$ ici) de groupes à utiliser et nous initialisons aléatoirement les coordonnées des points centraux. Pour déterminer le nombre de classes à utiliser, il est bon de jeter un coup d'œil rapide aux données et d'essayer d'identifier tout groupe distinct. Les points centraux sont des vecteurs de la même longueur que chaque vecteur de point de données (ici en dimension 2).
2. Chaque point de données est classé en calculant la distance entre ce point et le centre de chaque groupe, puis en classant le point dans le groupe dont le centre est le plus proche.
3. Sur la base de ces points classés par groupe, pour chaque groupe, nous recalculons le centre du groupe en prenant la moyenne de tous les vecteurs du groupe.
4. Répéter ces étapes pour un nombre déterminé d'itérations ou jusqu'à ce que les centres de groupe ne changent pas beaucoup entre les itérations (4 étapes suffisent dans notre exemple).

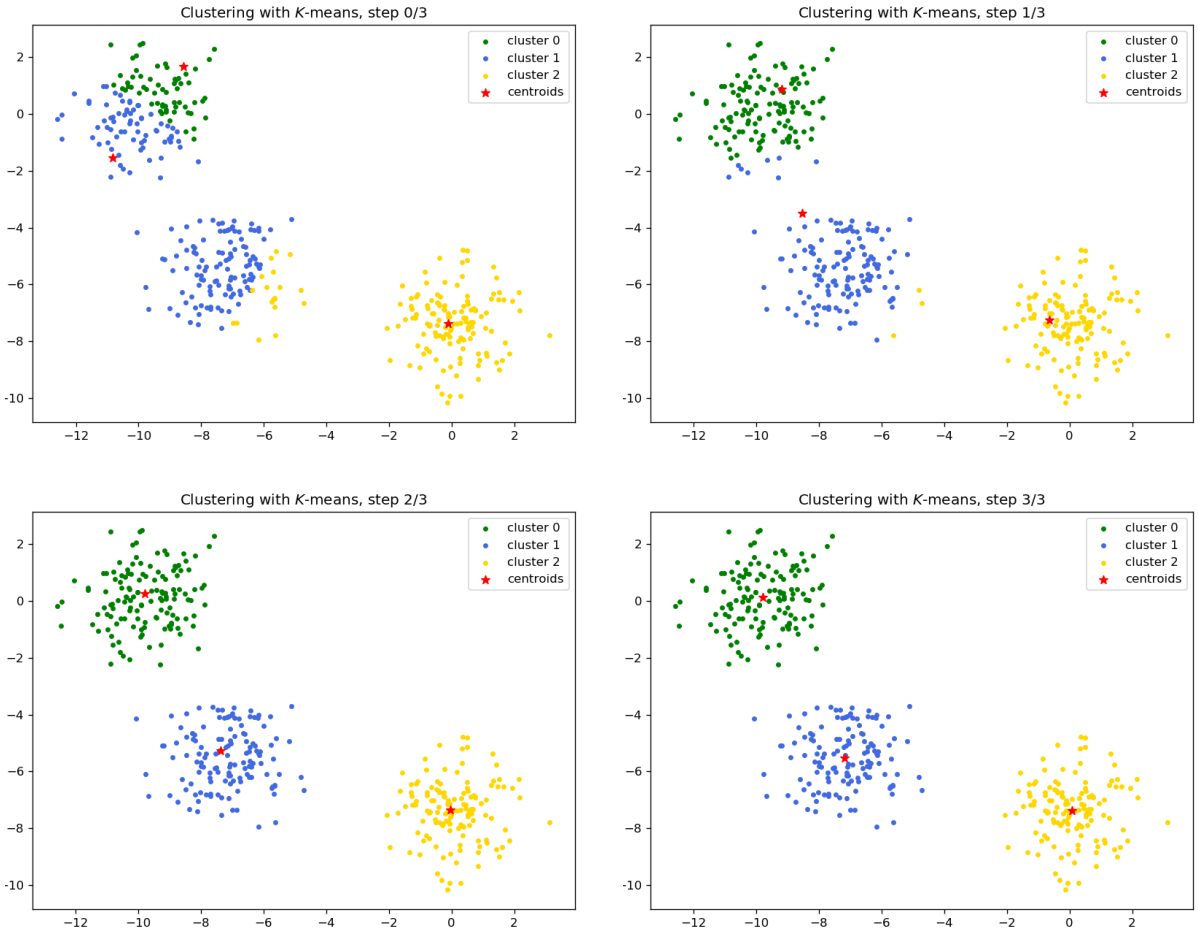

FIGURE 1 – Visualisation des étapes de l'algorithme de regroupement des K-moyennes sur des données en 2 dimensions avec 3 groupes distincts. Les groupes Les graphiques sont tirés de nos travaux personnels dans le cadre du cours de Deep Learning.

2 L'algorithme des K plus proches voisins (ou *K-nearest neighbors*)

Pour les codes et explications supplémentaires, se reporter au dépôt GitHub : `https://github.com/sylvaincom/C-for-data-science`, dans le dossier `kNN`. Nous avons codé cet algorithme de A à Z.

La méthode des K plus proches voisins (ou *K-nearest neighbors*) est une méthode d'apprentissage supervisé.

Prenons l'exemple de la figure 2 pour décrire l'algorithme des 5 plus proches voisins ($K = 5$) du graphique de droite.

- Sur le graphique de gauche, nous avons des données en 2 dimensions avec un label qui est la couleur rouge, verte ou bleue (d'où "supervisé"). Ces données constituent notre base d'apprentissage. Chaque couleur signifie le groupe auquel le point appartient. Notre but est de prédire la couleur d'un nouveau point.
- Pour une nouvelle observation (un nouveau point), dont on ne connaît pas encore la couleur, nous ne disposons que de ses coordonnées dans l'espace (x', y'). À partir de (x', y'), on calcule la distance entre ce nouveau point et tous les autres points (ceux de la base d'apprentissage).
- On ne garde que les 5 points les plus proches de notre nouveau point (d'où l'appellation "K plus proches voisins"). On regarde la couleur des 5 plus proches voisins et on fait un vote : on attribue au nouveau point la couleur majoritaire parmi ses 5 plus proches voisins.
  Sur le graphique de droite, les zones de couleur représentent le résultat de l'algorithme : le vote majoritaire pour de nouveaux points (appartenant à cette zone). Par exemple, un nouveau point tombant dans la zone bleue se verra attribuer la couleur bleue. (Le graphique comporte quelques erreurs "inévitables".)

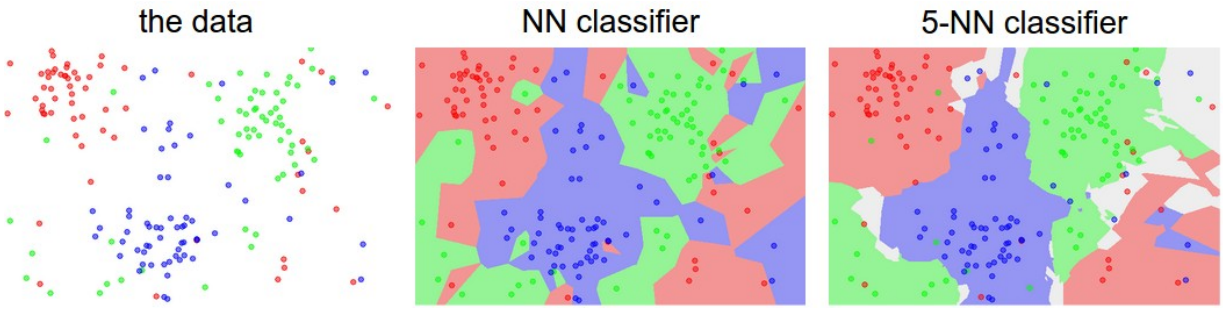

FIGURE 2 – Un exemple de la différence entre le 1-*Nearest Neighbor* et un classificateur 5-*Nearest Neighbor*, utilisant des points en 2 dimensions et 3 classes (rouge, bleu, vert). Les régions colorées sont les prédictions des classifieurs pour de futurs nouveaux points. Les régions blanches montrent les points qui sont classés de manière ambiguë (c'est-à-dire que les votes de classe sont à égalité pour au moins deux classes).

Source : `http://cs231n.github.io/classification/`

3 Réseau de neurones artificiels (ou *artificial neural network*)

Pour les codes et explications supplémentaires, se reporter au dépôt GitHub : `https://github.com/sylvaincom/C-for-data-science`, dans les dossiers `NN-tutorial-miller` et `NN-iris-data` (à lire dans cet ordre). En particulier, les `README.md` complètent ce rapport concis.

Pour cette section, notre travail s'inspire d'un tutoriel de Dave Miller paru en 2011 [2]. Nous sommes dans le cadre du Deep Learning avec apprentissage supervisé. Un réseau de neurones prend des inputs (par exemple des images de chien et de chats) et renvoie un output qui est la prédiction (l'image est un chien ou un chat).

Le code en C++ (environ 400 lignes) permet simuler un réseau de neurones afin de prédire les résultats de XOR (eXclusive OR) (voir table 1). Nous n'avons pas codé nous-mêmes un réseau de neurones de A à Z en C/C++ car il s'agit une tâche complexe (même avec le langage Python que nous maîtrisons mieux que le C/C++). Nous nous sommes appropriés ce code et nous l'avons modifié afin qu'il réponde à nos besoins.

Le code sur lequel nous nous basons est constitué d'un seul fichier `.cpp` à exécuter à partir d'un Terminal, en langage C++, qui comprend deux classes `Neuron` et `Net` (et également `TrainingData` pour lire les données d'entraînement).

Nous aurions pu diviser le fichier `.cpp` en plusieurs headers `.h` et plusieurs codes `.c` pour chaque classe, comme fait habituellement en TP, mais nous avons fait préféré conserver la structure originelle parce qu'elle est plus "universelle" et nous a permis de manipuler le compilateur `g++`. De plus, nous avons effectué cette division précédemment, pour les K-moyennes ainsi que les K plus proches voisins.

Par rapport au cours, on retrouve les notions clés suivantes :

- Une classe est constituée d'attributs (variables) et de méthodes (fonctions).
- Les éléments qui constituent une classe peuvent être publics ou privés. S'ils sont publics, on peut les utiliser n'importe où dans le code. S'ils sont privés, seule la classe peut les utiliser. Par défaut, les éléments sont en privé.
- La règle d'encapsulation est respectée : tous les attributs d'une classe doivent toujours être privés, il faut toujours passer par des méthodes pour modifier les attributs.
- On définit les méthodes en dehors de la classe, qui ne contient uniquement les déclarations de fonction et des attributs.
- On utilise les objets de classes dans le `main()`.

Nous avons également exploré davantage certains concepts par rapport au cours :

- Nous avons découvert le compilateur `g++`. Après quelques recherches, le compilateur `g++` (pour le langage C++) est très similaire au compilateur `gcc` vu en cours.
- Le mot-clé `struct`, utilisé en langage C mais également en C++, qui permet de créer des classes. La seule différence avec le mot-clé `class` est que, par défaut, les méthodes et attributs sont publics au lieu de privés.
- Le type `vector`, qui est un conteneur séquentiel qui encapsule les tableaux de taille dynamique.

Finalement, nous avons réussi à utiliser le code pour prédire l'output de XOR (*eXclusive OR*) qui signie "OU exclusif". Voir table 1. Les inputs sont binaires de dimension 2 et les outpus sont binaires de dimension 1.

Nous avons également pu adapter l'algorithme aux données Iris. Ces données sont décrites sur le lien GitHub : les inputs sont de dimension 4 (dans $\mathbb{R}^4$) et les outputs sont de dimension 1, à valeurs dans $\{0,1,2\}$. (Nous avons notamment dû modifier la fonction d'activation du réseau de neurones.)

Nous avons également pu modifier les hyper-paramètres du modèle. l'architecture du réseau de neurones : le nombre de neurones dans la couche cachée. Nous avons également pu modifier le "learning rate", la fonction d'activation et la fonction d'erreur.

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 1 – Table de vérité de XOR (ou exclusif). 0 signifie `False` et 1 signifie `True`

Ainsi, nous avons rempli nos objectifs : nous avons pu modifier la dimension des données inputs et l'ensemble des valeurs à prédire (les outputs) ainsi que les hyper-paramètres du modèle : ce sont les éléments fondamentaux dans un réseau de neurones.

4 Cython : le meilleur compromis entre Python et C/C++ ?

Un Data Scientist doit-il utiliser Python ou le C/C++ ?

Par rapport à Python, C/C++ a deux principaux avantages :

- Le temps d'exécution est plus court.
- Le langage C est plus bas niveau donc il est plus facile de communiquer avec le hardware.

Par rapport à C/C++, Python a deux principaux avantages :

- De nombreuses librairies très efficaces pour le Machine Learning et Deep Learning ont été implémentées en open-source : `scikit-learn`, `TensorFlow` et `PyTorch` notamment.
- Les Data Scientists n'étant pas réputés comme d'excellents développeurs, mais comme de meilleurs informaticiens que des mathématiciens et de meilleurs mathématiciens que les informaticiens, il leur plus facile de maîtriser Python que C/C++.
  En effet, le langage C est plus lourd que le langage Python dans le sens où il comporte plus de lignes : on gagne en temps d'exécution mais on perd en temps de développement. Le code en langage C étant très lourd en mémoire, le C++ essaie d'optimiser par rapport au C, notamment pour de grands projets. Notons que le C++ n'est pas un sur-ensemble du C... mais presque : tout ce qu'on a vu en cours est transposable du C au C++, mais il faut juste retenir que quelques concept du C n'existent pas dans C++.

Ainsi, un bon compromis entre Python et C/C++ (entre facilité de développement et temps d'exécution) semble être Cython. En un mot, Cython permet d'écrire du Python qui est traduit en C pour être compilé : "on accélère Python". La seule différence avec Python est qu'il faut préciser le type lors de la déclaration d'une variable. Un post introductif intéressant à ce sujet est celui de George Seif sur *Towards Data Science* [3].

D'après la documentation officielle de Cython, « *Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself. [...] The Cython language is a superset of the Python language that additionally supports calling C functions and declaring C types on variables and class attributes.* »

Conclusion

Dans ce projet, nous avons exploré 3 méthodes en machine/deep learning avec le langage C/C++ : les K-moyennes (non supervisé), les K plus proches voisins (supervisé) ainsi que les réseaux de neurones (supervisé). Nous avons pu coder de A à Z ou nous avons adapté ces algorithmes. Nous avons également évoqué les avantages que Cython peuvent présenter.

Références

[1] Pierre-Etienne Moreau, Cédric Zanni. Introduction au C/C++. École des Mines de Nancy. 2019/2020.
`https://www.depinfonancy.net/cours-%C3%A9lectifs/ces7aj`

[2] David Miller. *New Video Tutorial : Make a Neural Net Simulator in C++. Tech Notes : A Technical Blog by David Miller.* 4 février 2011.
`http://www.millermattson.com/dave/?p=54`

[3] George Seif. *Use Cython to get more than 30X speedup on your Python code Towards Data Science.* 25 juilllet 2019.
`https://towardsdatascience.com/use-cython-to-get-more-than-30x-speedup-on-your-python-code-f6cb337919b6`