UNIVERSITÉ DE SHERBROOKE
Faculté de sciences
Département d'informatique

Implémentation de classifieurs pour le tri feuilles d'arbre

IFT712 - Techniques d'apprentissage

Marc-Antoine Genest (14 079 588)
Mamadou Mountagha Bah (18 098 915)

Sherbrooke, Québec, Canada 13 avril 2019

Table des matières

1 Introduction

L'augmentation de la puissance de calcul et le développement de nouveaux algorithmes mathématiques ont fait naître, à la fin des années cinquante, un nouveau domaine informatique : l'apprentissage automatique [2]. Depuis, celui-ci s'est appliqué sur de nombreux problèmes, dont celui de la classification d'images. Dans le présent rapport, le problème de classification de feuilles d'arbre sur la base de données *Kaggle* [3] sera abordé, en testant six différents classificateurs et en testant leur performances. D'abord, les méthodes de classification utilisées et le programme de tests seront présentés. Ensuite, les résultats d'identification de feuilles des classificateurs seront comparés entre eux, puis avec ce qui semble être l'état de l'art dans la littérature. Le but est d'explorer différentes méthodes de classification, d'en tirer leurs avantages et désavantages, et de valider s'il est possible de reproduire ce qui s'est fait jusqu'à maintenant.

2 Classification

Le problème de classification d'images se pose depuis plusieurs années, et plusieurs stratégies ont été trouvées pour augmenter la fidélité d'un classificateur, comme le pré-traitement d'images [4] et la segmentation [5]. De plus, un large éventail de choix de classificateurs existe, lesquels ayant chacun leurs avantages et désavantages, en fonction du type de problème. Pour ce qui est de la classification de feuilles ou d'identification de maladie sur les feuilles, les algorithmes couramment utilisés sont les machines à vecteurs de supports (SVM), les perceptrons multicouches (MLP), les ensembles de forêts aléatoires (Random Forest), les analyses de voisins les plus proches (KNN), les algorithmes de *Naive Bayes* (NB) et l'analyse de discriminant linéaire (LDA) [6][7].

2.1 Méthodes de classification et paramètres [1]

Pour ce qui est de l'implémentation, la bibliothèque *scikit.learn* a été utilisée, permettant de facilement choisir les hyperparamètres à ajuster pour la problématique de classification de feuilles d'arbre. Ayant un grand nombre de classes, et des données bien segmentées (textures, formes, etc), certaines entrées du programme sont laissées à celles par défaut, étant considérées comme moins importantes pour l'atteinte de l'objectif. Les prochaines sous-sections décrivent les algorithmes et les paramètres contrôlés.

2.1.1 SVM

De façon général, les machines à vecteur de support trouve l'hyperplan qui sépare le mieux les frontières de chaque classe, et de façon à ce que la distance frontière-hyperplan soit la plus grande possible. C'est donc un algorithme qui peut être efficace pour les problèmes à plusieurs dimensions, et ce même si le nombre de données d'entraînement est plus petit que le nombre de dimension. L'utilisation de noeuds est également possible, permettant d'être utiliser et efficace pour différentes distribution de données d'entraînement. C'est cependant une méthode coûteuse en temps de calcul, qui a beaucoup de chance d'*over-fitting* si les noeuds et paramètres choisis ne sont pas adéquats.

Les hyperparamètres choisis variables pour cette méthode sont la pénalité sur le terme d'erreur (C) et le noeud (kernel) ainsi que chacun de leur paramètre. Ces paramètres permettront de choisir la meilleure régularisation et la meilleure fonction de séparation en fonction des données d'entraînement.

2.1.2 MLP

Les MLP, à leur tour, tentent plutôt de réduire le nombre de dimensions des données d'entrée grâce à des couches de neurones. Une série d'opération est exécutée entre chaque donnée d'entrée, avec différents poids et différentes fonctions, augmentant ou réduisant à chaque étape la dimensionnalité du problème. À la fin de ces opération, le nombre de dimension est réduit à celui demandé par la fonction de prédiction f(x). Le grand avantage de ce modèle est qu'il peut apprendre des modèles hautement non-linéaire. Cependant, il possède un grand nombre de paramètres, pouvant mené à plusieurs minima locaux et à des différences de résultats lors de l'initialisation aléatoire.

2.1.3 Random Forest

Pour ce qui est du Random Forest, c'est une méthode par ensemble qui fait *pousser* des *branches* et des *feuilles* (prédictions) avec différents poids et différentes longueurs, lui permettant de faire des prédictions. Une donnée d'entrée passe donc au-travers les opérations de la branche, pour finalement trouver une *feuille*. La particularité du Random Forest est la façon avec laquelle les différents poids sont choisis pour bâtir une branche, et l'ensemble partiel de paramètres choisis pour voter pour quelles branches constitueront l'arbre. L'ajout de plusieurs arbres et branches, dû au facteur aléatoire, permet de diminuer la variance du modèle, et de fournir de meilleurs

prédictions. Son avantage principal est sa rapidité à entraîner et sa capacité à empêcher de sur-entraîner en augmentant simplement la taille de la forêt. Le désavantage est que pour avoir de meilleures prédictions, il faut une plus grande forêt, mais que le gain devient très faible à partir d'un certain seuil. Le temps de calcul peut donc devenir problématique pour une grande précision.

Les hyperparamètres choisis variables pour cette méthode sont le nombre d'arbre dans la forêt, ainsi que le nombre de caractérisques à considérer au début de l'algorithme. Cela permettra de trouver le nombre optimal d'arbres sans tomber dans un minimum local, tout en minimisant le temps de calcul.

2.1.4 KNN

En ce qui a trait aux KNN, c'est un algorithme sans modèle, à la différence des autres algorithmes vus jusqu'ici. En effet, il s'agit simplement de savoir, en une région de l'espace, quelle est la classe majoritaire. C'est donc un modèle simple, mais qui est très dépendant de la répartition des données d'entrées.

Les hyperparamètres choisis variables pour cette méthode sont le nombre de voisins à considérer, le poids à considérer en fonction de la distance, et le nombre de feuilles. Cela permettra de trouver la meilleure classification possible sans se soucier de la distribution des données.

2.1.5 Naive Bayes

Les Naive Bayes, eux, portent bien leur noms : ils appliquent « naïvement » la loi de Bayes, avec l'hypothèse que chaque probabilité est indépendante, aux données d'entraînement. De ce calcul est trouvé une fonction de type *maximum a posteriori*, retournant la classe dans laquelle une donnée test a le plus de chances de se retrouver. Malgré une hypothèse forte d'indépendance entre les paramètres, l'article [8] montre que, sous certaines conditions, cette méthode de classification peut être le meilleur choix à faire.

Les hyperparamètres choisis sont l'hypothèse de la distribution, soit gaussienne ou multinomiale, ainsi que leurs hyperparamètres respectifs, permettant de trouver la meilleure classification entre deux types de distribution assumée pour les données d'entrée.

2.1.6 LDA

La classification par LDA, tant qu'à elle, prend comme hypothèse que les données dans une classe sont distribuées de façon gaussienne. Elle a ensuite pour but de trouver la surface (linéaire dans le cas de la LDA) séparant le mieux les données de deux classes. Les avantages de cette méthode sont qu'il n'y a aucun hyperparamètre à ajuster (à l'exception de l'algorithme utilisé pour solutionner le problème), qu'elle est donc simple à utilisée, et qu'elle fournie d'excellents résultats dans certains cas. À son désavantage, elle n'est applicable qu'aux problèmes séparables linéairement, et est très dépendante de la distribution des données d'entraînement.

Comme mentionné plus haut, les seuls hyperparamètres pouvant être testés sont les algorithmes mathématiques utilisés, permettant d'être plus ou moins sensible à certaines caractéristiques des données.

3 Méthodologie

Cette section va décrire la structure pour laquelle nous avons opté pour structurer notre programme ainsi que la manière de l'exécuter

3.1 Description et fonctionnement du programme

Nous avons opté pour un code modulaire on a donc divisé notre code en 3 fichiers :

- un fichier **dataManagement.py** qui contient toute la logique sur les données du chargement en passant par le pré-traitement jusqu'à la division en données d'entraînement et de test. Ce fichier contient la classe **dataGenerator** qui à son tour contient les méthodes ***generate_data***, qui récupère les données à partir des fichiers de données **train.csv** et **test.csv** et nous renvoie les données d'entraînement et de test pour l'entraînement, ***encoder*** nous servira par la suite a encoder les données de test fournies par kaggle et pouvoir faire un test et de soumettre nos résultats.
- Un fichier **methodsClassifier.py** qui contient toute les méthodes qui contient toutes les méthodes qui sont nécessaires pour tout tout ce qui a attrait à l'entraînement, la validation croisée le test et la précision obtenu par le classifieur instancié. Il faut noter aussi que nous avons fait appel à deux méthodes de recherches d'hyperparamètres un random grid search pour initialiser les paramètres puis un grid search plus défini pour trouver les meilleurs paramètres possibles. Outre les méthodes d'entraînement et de prédiction

classique on y retrouve aussi les méthodes de recherche d'hyperparametres. Pour ce qui est de l'appréciation de la performance d'une méthode de classification donnée nous avons opté pour la précision qui est bien adapté dans notre cas.

— Le dernier fichier c'est le fichier principale **main** qui instancie ces deux classes et en appelle les méthodes appropriées au besoin.

Le diagramme de la 1 donne une structure simplifiée des classes et fichiers du projet.

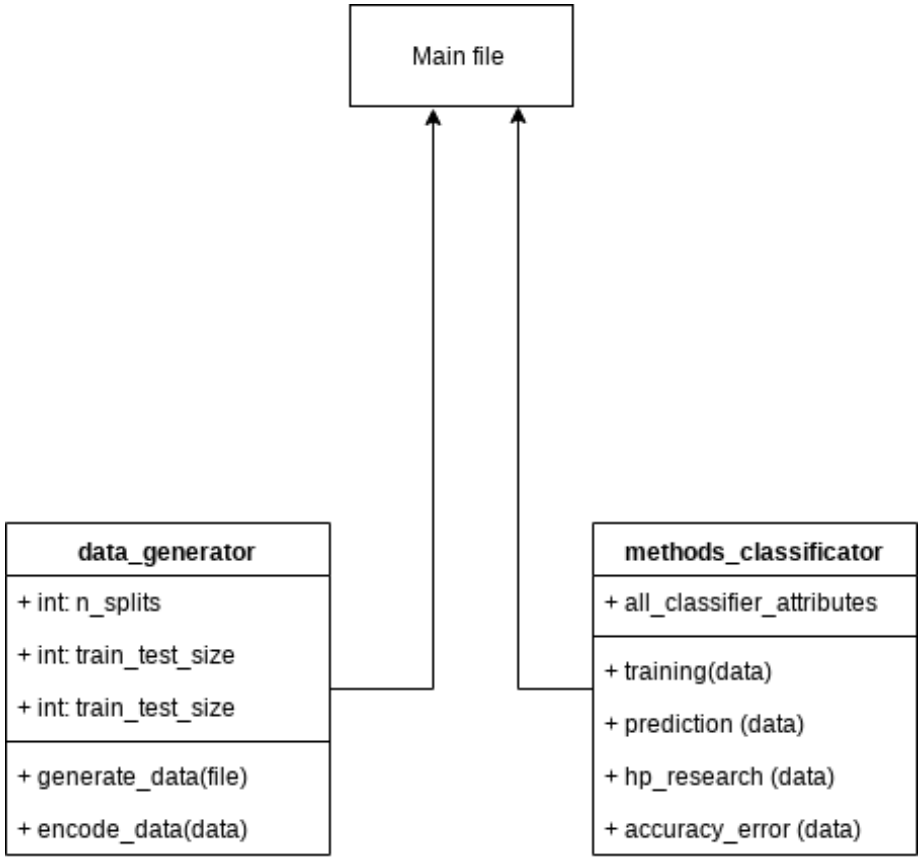

Figure 1 – Titre

3.1.1 Entrées

Le programme reçoit ses entrées via la ligne de commande. Nous avons considéré que celui qui testera le programme aura une machine sur laquelle tourner linux ou macOS. Pour le cas de windows il faudra ouvrir le fichier principale et fixer les valeurs en dur ou alors laisser celles données en défaut ; elles marchent bien. Il y a donc deux appels possibles :

— un premier appel qui permet de fournir la méthode de classification à utiliser et spécifier spécifier si on compte utiliser la recherche d'hyperparamètres. En guise d'exemple si on veut utiliser la méthode SVM et faire une recherche d'hyperparamètres l'utilisateur va entrer dans le terminal la ligne suivante : ***python3 classifier.py –method=SVC –research_hyper_parameter=True***

— un deuxième appel au lieu de faire une recherche d'hyperparamètres donne la possibilité à l'utilisateur d'entrée les paramètres de la méthode de classification choisie. Etant donné le nombre élévé de paramètres nous avons fixé un certain nombre de paramètres. Ces derniers peuvent être retrouvés en tapant la commande ***python3 classifier.py –help*** . Nous avons opté pour les arguments longs pour pas prêté à confusion vu le nombre élévé

de paramètres. Pour la méthode SVC par exemple il pourrait appeler une commande qui ressemble à ***python3 classifier.py –method=SVC –kernel=rbf –gamma=5***. Bien évidemment l'utilisateur n'est pas tenu obligé de fournir tous les arguments juste ceux qu'il veut, tous les autres auront une valeur par défaut. Pour comprendre comment appeler les différents classifieurs avec leurs paramètres respectifs nous suggérons fortement de se servir de l'argument **–help** .

3.1.2 Sorties

En sortie nous aurons selon la méthode de classification choisie et le type d'éxecution (avec recherche d'hyperparamètres ou pas) un affichage des meilleurs hyperparamètres, la précision sur les données d'entrainement et de test.

3.2 Description de la plateforme de tests

Cette section décrira la liste des paramètres utilisés et testés, ainsi que les tests exécutés, permettant de comparer les méthodes entre elles et de fournir assez de données pour comparer les résultats à la littérature.

3.2.1 Distributions pour la recherche d'hyperparamètres

Comme mentionné dans la section précédente, la recherche d'hyperparamètre se fait grâce à deux algorithmes l'un à la suite de l'autre, permettant d'économiser du temps et de faire une bonne recherche. Le tableau 1 montre les distributions testés par le programme en fonction de la méthode de classification choisie.

3.2.2 Procédure de test

Pour évaluer quel classificateur est le plus approprié pour le problème de classification de feuilles d'arbres, une série de tests à été choisie, présentée au tableau 2. Pour chacun de ces tests seront notés l'erreur d'entraînement, l'erreur de test ainsi que le temps de calcul.

Ces tests permettront d'évaluer les classificateurs selon plusieurs critères, en répondant aux questions suivantes :

— Quel classificateur fourni la meilleure fidélité ?

Méthode	**Paramètre**	***Random Search***	***Grid Search***
SVM	`kernel`	[linear, poly, rbf, sigmoid]	best
	`C`	int[1, 50]	float[best ±5]
	`degree`	int[1, 10]	float[best ±5]
	`coef0`	int[1, 50]	float[best ±5]
	`gamma`	int[1, 20]	float[best ±5]
MLP	`hidden_layer_sizes`	int[(30-50, 30-50)]	best
	`activation`	[identity, logistic, tanh, relu]	best
	`alpha`	float[1×10^{-5}, 1×10^{-1}]	float[best/(0.5-2)]
Random Forest	`n_estimators`	int[10, 100]	int[best ±5]
	`max_features`	[log2, sqrt, 1.0]	best
KNN	`n_neighbors`	int[5, 30]	int[best ±5]
	`weights`	[uniform, distance]	best
	`leaf_size`	int[10, 50]	int[best ±5]
Multinomial NB	`alpha`	float[0, 5]	int[best ±1]
Gaussian NB	-	-	-
LDA	-	-	-

Table 1 – Ensembles des distributions testées par le *Random Search* et le *Grid Search* lors de la recherche d'hyperparamètres. À noter que « best » réfère au résultat ayant la plus petite erreur trouvé par le *Random Search*.

Test	**Nb d'itérations**	**Taille de l'ensemble d'entraînement**
1	3	60%
2	6	60%
3	3	80%
4	5	80%
5	8	80%

Table 2 – Plateforme de tests faite pour chaque classificateur implémenté.

— Est-ce que le temps de calcul peut être contraignant ?
— Quelle est la dépendance des résultats avec la taille de l'ensemble d'entraînement ?
— Quelle est la dépendance des résultats avec le nombre de validation croisée ?

Le but étant de déterminer le meilleur classificateur, il sera possible d'identifier s'il y a un modèle suprême dominant tous les autres en terme de fidélité et de temps de calcul, ou si le choix du classificateur peut dépendre des circonstances dans lesquelles on se trouve.

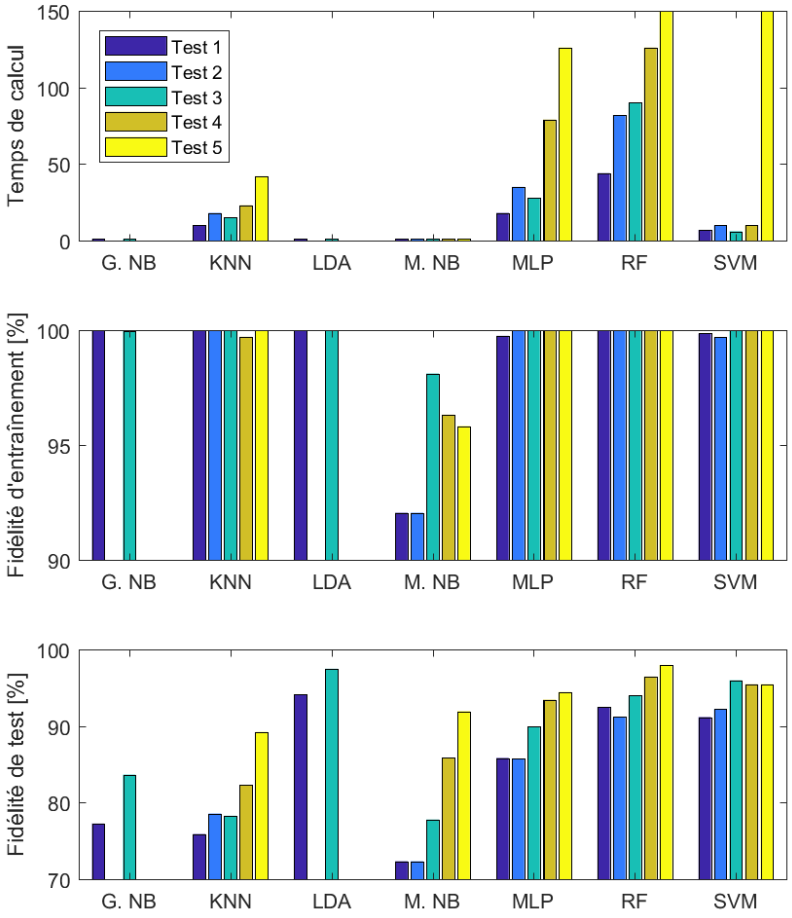

Figure 2 – Figure comparative des différents classificateurs implémentés, montrant le temps de calcul et la fidélité pour chacun, et où chaque barres représente un test effectué.

4 Résultats

Suite à la réalisation de la plateforme de tests présentée à la section précédente, la figure comparative 2 a été montée, permettant la visualisation rapide des résultats.

À noter que les valeurs des axes « y » ont été modifiées pour bien voir l'entièreté des résultats (les valeurs manquantes de temps de calcul sont de 428 et 189 secondes pour le RF et le SVM respectivement). De plus, les classificateurs « G. NB » et « LDA » n'ont que deux barres puisqu'ils n'ont pas d'hyperparamètres à ajuster.

4.1 Évaluation des classificateurs

D'abord, comme observation générale, il est à noter que les classificateurs ont tous fournis de meilleurs résultats en augmentant le nombre de données d'entraînement (sur les cinq barre, les deux de gauches sont à 60 % de la base de données totale, alors que les trois dernières sont pour 80 %). Ensuite, à l'exception du classificateur SVM, l'augmentation du nombre d'itérations pour la recherche d'hyperparamètres augmente également la fidélité de l'algorithme. Ces deux observations sont biens ce à quoi on s'attendait : une recherche plus exhaustive et augmenter le nombre d'entraînement permettent forcément au modèle de trouver un meilleur minimum local pour la fonction de coût (à l'exception du SVM, qui dépend davantage de la distribution des valeurs).

À la lueur des résultats, les méthodes de classification de Gaussian Naive Bayes (G. NB) et de plus proches voisins (KNN) sont à exclure, fournissant les pire résultats. De plus, pour des raisons de complexité, de temps de calcul, et ne pas pour autant fournir de meilleurs résultats, le perceptron multi-couches (MLP) est également considéré à éviter. Pour ce qui est du Multinomial Naive Bayes (M. NB), bien que les résultats ne soient pas les meilleurs, il est possible d'observer une augmentation rapide de la fidélité avec le nombre d'itérations, sans nécéssité une plus grande puissance de calcul. Aucune conclusion ne peut donc être tirer pour ce classificateur, mais le modèle ne semble pas être en saturation d'apprentissage, et pourrait potentiellement fournir de meilleur résultats.

La liste des meilleurs classificateurs pour la classification de feuilles d'arbres se réduit donc aux LDA, RF et SVM. Il y a donc, tout en premier lieu, le LDA, qui permet d'atteidre une fidélité entre 94 et 98 % sans ajustement d'hyperparamètre, et avec des temps de calcul négligeable. Ces propriétés font de cette méthode celle à privilégiée pour tout problème séparable linéairement. Par la suite, le RF, bien qu'exigeant en temps de calcul, fourni une fidélité record de 97.5 %. Le SVM, quant à lui, peut fournir de bons résultats avec peu de temps de calcul (voir test 3, fidélité des tests), mais nécessite un meilleur ajustement de paramètres. En effet, la raison expliquant un meilleur résultat pour un plus faible nombre d'itération s'explique par le fait que les nombres « tirés » au hasard par le *Random Search* ont été plus fructueux lors d'un des essais. Le SVM étant sensible à ces hyper-paramètres, il suffit de restreindre davantage les distributions de recherche pour gagner en fidélité.

Pour conclure, les classificateurs NB, KNN et MLP ont été exclus car ils fournissent de moins bons résultats pour une plus grande difficulté d'utilisation. Finalement, le classificateur LDA a été déterminé comme étant le meilleur de tous, le RF est conseillé si la puissance et le temps de calcul sont disponibles, et les SVM sont conseillés si les hyperparamètres à ajuster ne sont pas

totalement inconnus.

4.2 Validation des résultats

Maintenant que l'analyse entre les classificateurs implémentés a été faite, il est pertinent de valider si c'est également les tendances qui se trouve dans la littérature.

Dans le premier article cité ([6]), le RF est courroné gagnant avec 95 % de fidélité, suivi de près par le MLP, puis les SVM en dernier, avec 80 % de réussite. Cependant, le noeud utilisé pour leur machine à vecteurs de support n'est pas le même que celui choisi par notre recherche d'hyperparamètre. La grande différence entre ces deux classificateurs peut s'expliquer par ce choix. Pour ce qui est des deux autres, la légère différence peut être due au fait qu'ils ont beaucoup plus de données d'entraînement, ou qu'ils analysent des données différentes.

Dans le second ([7]), on note LDA à 98.7, RF à 97.5, NB à 95.0 et KNN à 77.5 %. Encore une fois, KNN est à exclure, LDA est dominant, et RF arrive à fournir de très bons résultats. Il est égaglement intéressant de voir que NB arrive finalement à avoir une bonne fidélité, ce qui justifierait la poursuite des tests de classification de feuilles d'arbres avec ce classificateur. Encore une fois, les fidélités plus élevées peuvent être dues au plus grand ensemble d'entraînement, et à des algorithmes plus poussés, décrit dans l'article.

Les résultats trouvés avec notre programme et notre bancs de tests sont donc assez semblables à ce s'est fait en terme de classification d'image, et fournissent une bonne base comparative pour le choix d'un classificateur.

5 Conclusion

Dans ce projet nous avons eu à tester plusieurs algorithmes de classification sous plusieurs angles dans le but de trouver le plus performant afin d'obtenir la précision la plus bonne possible et de pouvoir comparer avec ce qui existe dans la littérature. Nous avons joué sur la sélection de modèle bien fourni avec la librairie sklearn et nous avons obtenu des résultats plus ou moins bons selon la méthode choisie comme indiqué sur 2 Nous avons obtenu notre meilleur score avec la méthode **Random Forest** avec une précision de **97.98%** ce qui est plutôt normal vu que c'est un ensemble de méthodes. Nous avons soumis notre meilleur score sur kaggle. Cependant avec les avancées actuelles des méthodes de classification notamment les réseaux de neurones à convolution, obtenir un bon classement avec une méthode classique comme la notre s'avère

difficile voire impossible. Cependant on aurait peut être fait plus de progrès si nous avions un jeu de données plus important.

Références

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[2] Arthur lee samuel. history.computer.org, consulté : 10-04-2019.

[3] https://www.kaggle.com/c/leaf-classification, consulté : 10-04-2019.

[4] Vidya Manian Santiago Velasco-Forero. Improving hyperspectral image classification using spatial preprocessing. *IEEE Geoscience and Remote Sensing Letters*, 6, 2009.

[5] Francis Bach Zaid Harchaoui. Image classification with segmentation graph kernels. *IEEE **1-4244-1180-7***, 07.

[6] SanaUllah Manzoor Fahad Najeeb Muhammad Yasir Siddique Rafaqat Alam Khan Hidayat ur Rahman, Nadeem Jabbar Ch*. A comparative analysis of machine learning approaches for plant disease identification. *Advancements in Life Sciences*, 4, August 2017.

[7] A Chinnasamy Sudheer Reddy Bandi, Varadharajan A. Performance evaluation of various statistical classifiers in detecting the diseased citrus leaves. *International Journal of Engineering Science and Technology*, 5, February 2013.

[8] H. Zhang. The optimality of naive bayes. *Proc. FLAIRS.*, 2004.