FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

IFT712 - Techniques d'apprentissage

Projet - *Classifieur de feuilles végétales*

préparé par

Agathe Le Bouler (leba3207)

Gabriel Gibeau Sanchez (gibg2501)

Philippe Spino (spip2401)

présenté à

Pierre-Marc Jodoin

11 avril 2020

Table des matières

1 Mise en contexte **1**

2 Définition du projet **1**

3 Base de données **1**

4 Choix des modèles de classification **2**

5 Recherche des hyperparamètres **4**

6 Résultats **6**

7 Gestion de projet **8**

8 Conclusion et pistes de recherche future **10**

1 Mise en contexte

Ce projet a pour but de présenter différentes méthodes de classification dans le cadre d'un apprentissage supervisée.
Ce projet vise également à mettre en application des méthodes de cross-validation et de recherche d'hyper-paramètres dans le but de déterminer les modèles de classification les plus performants.

2 Définition du projet

Ce présent projet classifie un ensemble de feuilles végétales, composés d'informations extraites de 1584 images de feuilles végétales, selon leurs caractéristiques physiques. Les caractéristiques soit la forme, la marge (à échelle précise) et la texture des feuilles de l'ensemble, sont des vecteurs 3x64.
Nous avons développés un gestionnaire choisissant un classifieur spécifique ou bien l'ensemble de 7 classifieurs. Ce gestionnaire permet, une fois le/les classifieurs choisis, d'effectuer une recherche d'hyper-paramètres, d'entrainer un ou plusieurs modèle(s) et de générer les informations sur la performance du/des modèle(s).

3 Base de données

La base de données utilisée lors de ce projet est celle du challenge Kaggle de classification de feuilles d'arbres disponible à l'adresse suivante. Cette base de données est composée de deux fichiers, *train.csv* et *test.csv*, représentant respectivement les jeux de données d'entraînement et de test. Le jeu de données d'entraînement est pertinent car il fournit un ensemble annoté de 990 feuilles composé de 192 attributs.

4 Choix des modèles de classification

Le choix des différents classifieurs pour mener à bien ce projet a été réalisé dans l'optique d'essayer plusieurs familles de classifieurs. Nous nous sommes arrêtés à l'utilisation de six classifieurs multiclasses pour une application d'un apprentissage supervisée implémentés par la librairie Sklearn.

Perceptron MultiCouches

Le réseau de neurones de type Perceptron MultiCouches a pour but d'apprendre à établir un lien mathématique entre les données de l'ensemble d'entrée et les différentes étiquettes de classe, dans le cas d'une classification. De part la possibilité de disposer de multiples couches, le réseau a la capacité d'établir des modèles non-linéaires, qui pourrait se révéler intéressant sur notre jeu de données. Nous décidons d'utiliser la méthode *neural_network.MLPClassifier* implémentée par Sklearn, qui utilise la rétro-propagation ainsi que la fonction d'erreur de l'entropie croisée. Aussi, étant donné notre problème multi-classe, la fonction d'activation de la couche de sortie est une Softmax.

K-Plus Proches Voisins

La classification basée sur les voisins n'a pas comme but d'être la méthode la plus généralisable possible, elle se contente en effet de sauvegarder les exemples du jeu de données d'entraînement. Celle-ci peut cependant se révélée très performante et, au vu de la taille de notre jeu de données d'entraînement, son utilisation est envisageable. Nous avons décidé de tester la méthode des K-Plus Proches Voisins, implémentés par Sklearn avec neighbors.KNeighborsClassifier.

Analyse du Discriminant Linéaire

L'essai sur notre jeu de données d'entraînement de la méthode de l'analyse du discriminant linéaire permet de s'informer si ce jeu de données est effectivement bien linéairement séparable. Ce type de classification, étant de forme fermée (*closed-form*), démontre son efficacité lors de la phase d'entraînement. La méthode est implémentée par Sklearn sous la librairie *discriminant_analysis.LinearDiscriminantAnalysis*.

Bayes naïf gaussienne

Les méthodes de classification se basant sur l'application du théorème de Bayes font l'hypothèse qu'il exsite une indépendance conditionnelle entre chaque attribut sachant la valeur

de l'étiquette de classe. L'utilisation d'une estimation par Maximum A Posteriori est aussi utilisée afin de déterminer les distributions de chacune des étiquettes de classe et de chacun des attributs en sachant la valeur de l'étiquette de classe. Nous avons décider, en plus, de faire l'hypothèse que les données suivent des distributions gaussiennens dans l'optique d'utiliser la méthode de classification de Bayes naïf gaussien. Celle-ci se retrouve implémentée par Sklearn sous la méthode *naive_ bayes.GaussianNB*.

Forêts aléatoires

Dans le but de généraliser au mieux, les méthodes d'ensemble permettent de combiner les prédictions de multiples estimateurs en un seul estimateur. On s'intéresse tout d'abord aux méthodes basées sur la moyenne des prédictions de chaque estimateur, dont les arbres de décision aléatoire font partis. La méthode Sklearn *ensemble.RandomForestClassifier* a la particularité d'implémenter une méthode de type Boostrap, qui permet de décroître d'autant plus la variance de l'estimateur ayant souvent tendance à sur-apprendre.

Machine à vecteur de support

Les méthodes de machines à vecteur de support sont également implémentées dans Sklearn et permettent d'apporter une approche "un-contre-un" à la classification. Celles-ci montrent aussi de bons résultats lorsque la dimensionalité des données est importante, ce qui pourrait être le cas de notre jeu de données d'entraînement. On utilise alors la méthode Sklearn *svm.SVC* qui est spécifique à la classification.

Régresseur Ridge

L'utilisation d'une méthode de type régresseur Ridge permet de traiter le problème tel une tâche de régression à sorties multiples. Pour chaque classe correspond une sortie, celle dont la valeur est la plus élevée est donc la classe prédite par le modèle.

5 Recherche des hyperparamètres

Afin de déterminer les hyper-paramètres menant à des performances maximales des modèles de classification (meilleure justesse d'entraînement et de validation), nous avons décidé d'utiliser une méthode de validation croisée.
Pour cela, nous avons pu la méthode Sklearn *model_selection.GridSearchCV*, qui prend en paramètre un estimateur (notre modèle de classification), un espace de paramètres à tester, une fonction d'évalutation (dans notre cas, la justesse).
On s'intéresse maintenant aux hyper-paramètres des différents modèles pour notre validation croisée.

Perceptron MultiCouches

- **Taille des couches cachées** : Facteur faisant varier le nombre de neurones composant une couche cachée.
- **Initialisation des taux d'apprentissage** : Valeur du taux d'apprentissage à l'initialisation du réseau.
- **Algorithme d'optimisation** : Choix de l'algorithme d'optimisation pour les mises à jours des poids du réseau (*adam*, *sgd*.
- **Fonction d'activation** : Choix de la fonction d'activation utilisée par les couches cachées (*relu*, *logistic*).

K-Plus Proches Voisins

- **Nombre de voisins** : Valeur du nombre de voisins utilisé pour statuer le jeu de données.
- **Poids** : Mesure de poids utilisé pour la prédiction (*uniform*, *distance*).
- **Algorithme** : Choix de l'algorithme utilisé pour déterminer les plus proches voisins (*ball_tree*, *kd_tree*, *brute*, *auto*).
- **Taille des feuilles** : Valeur affectant la vitesse de contruction et de recherche des voisins. Utilisé par les algorithmes *ball_tree* et *kd_tree*.
- **Puissance p** : Permet le choix de la métrique entre la distance de Manhattan et la distance Euclidienne.

Analyse du Discriminant Linéaire

- **Algorithme** : Choix de l'algorithme pour le calcul des distributions (*svd*, *lsqr*, *eigen*)
- **Nombre de composants** : Valeur déterminant le nombre de composants utilisé pour la réduction de dimensionalité.

— **Seuil** : Valeur du seuil spécifiquement utilisé pour l'algorithme *svd*.

Bayes naïf gaussienne

— **Lissage des variables** : Valeur du lissage qui est une proportion de la variance devant être ajoutée afin de stabiliser les résultats des calculs.

Forêts aléatoires

— **Nombre d'estimateurs** : Valeur du nombre d'arbres utilisé pour construire l'arbre.
— **Profondeur maximale** : Valeur de la profondeur maximale d'un arbre.

Machine à vecteur de support

— **Paramètre de régularisarion C** : Valeur déterminant la force de régularisation qui est la proportionnelle inverse de C.
— **Noyau** : Choix du type de noyau utilisé par l'algorithme (*linear*, *poly*, *rbf*, *sigmoid*)
— **Gamma** : Valeur du coefficient utilisé par le noyau, spécifiquement pour les noyaux *poly*, *rbf* et *sigmoid*.

Régresseur Ridge

— **Paramètre de régularisarion alpha** : Valeur déterminant la force de régularisation permettant de réduire la variance du modèle.

6 Résultats

Les résultats de la recherche des hyper-paramètres ont permis de déterminer, dans un premier temps, les valeurs et choix des hyper-paramètres résultant des scores de justesse les plus élevés pour chacun des modèles. Aussi, nous avons pu déterminer des scores de justesse sur un jeu de données de validation afin de s'assurer que les modèle généralisaient correctement.

TABLE 1 – Présentation des scores de justesse les plus performants des modèles

Modèle	Justesse d'entraînement (%)	Justesse de validation (%)
MLP	100.00	95.96
KNN	100.00	97.47
LDA	100.00	97.98
GNB	99.87	96.97
RF	100.00	98.48
SVM	100.00	95.45
Reg	99.37	91.92

Ces résultats ont été établis selon les choix et valeurs des hyper-paramètres suivants :

Perceptron MultiCouches

- **Taille des couches cachées** : 50
- **Initialisation des taux d'apprentissage** : 0.1
- **Algorithme d'optimisation** : *adam*
- **Fonction d'activation** : *logistic*

K-Plus Proches Voisins

- **Nombre de voisins** : 1
- **Poids** : *uniform*
- **Algorithme** : *ball_tree*
- **Taille des feuilles** : 10
- **Puissance p** : 1

Analyse du Discriminant Linéaire

- **Algorithme** : *svd*
- **Nombre de composants** : 89

— **Seuil** : 0.1

Bayes naïf gaussienne

— **Lissage des variables** : 0.001

Forêts aléatoires

— **Nombre d'estimateurs** : 450
— **Profondeur maximale** : 30

Machine à vecteur de support

— **Paramètre de régularisarion C** : 1000
— **Noyau** : *rbf*
— **Gamma** : 0.1

Régresseur Ridge

— **Paramètre de régularisarion alpha** : 0.001

7 Gestion de projet

Afin d'assurer le suivi des tâches du projet, le service git gratuit de Microsoft (`https://github.com/AgatheLB/leaf_classification_IFT712`) a été utilisé. Il permet non seulement d'utiliser le logiciel de contrôle de versions Git, mais également de créer un tableau de type trello tel que présenté à la figure 1. La méthode kanban dans sa forme la plus simple a été employée pour assigner et suivre les tâches de développement.

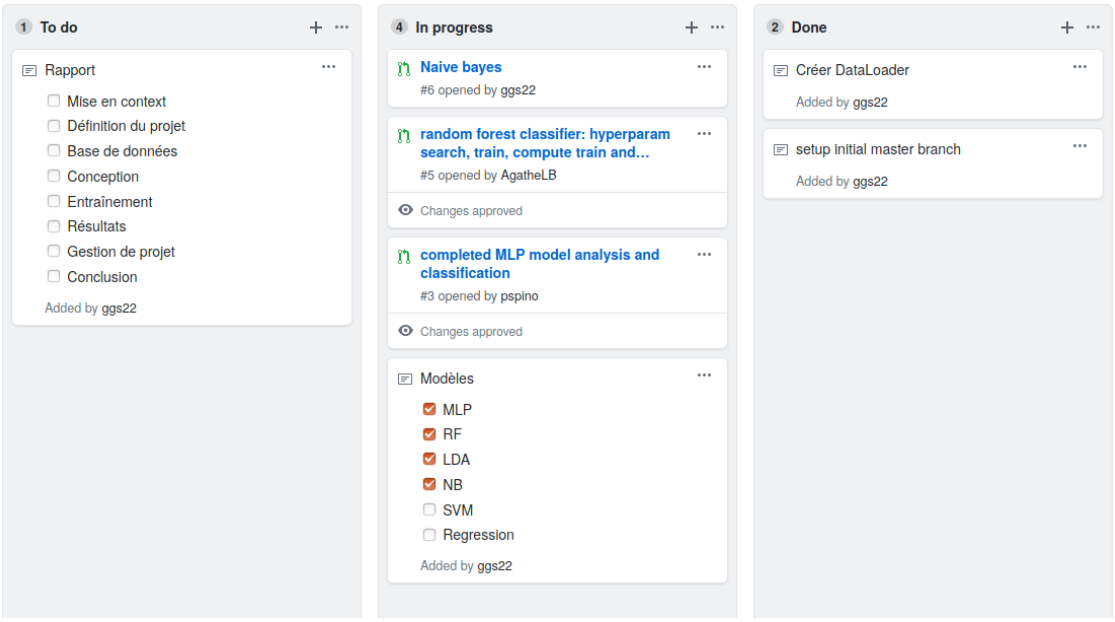

FIGURE 1 – Tableau Trello du projet Classifieur de feuilles végétales

Chaque tâche de développement avait une branche dédiée. Lorsque la tâche était complétée, une *pull request* était créée via l'interface web. Chaque *pull request* a fait l'objet d'une révision par au moins un autre membre de l'équipe. Une fois approuvée et fusionnée, la branche était supprimée.

Tel que dicté par les bonnes pratiques, il était convenu d'éviter les *commits* introduisant trop de changements. Cela afin de simplifier la tâche de révision, et bien cerner le *scope* des tâches. Il a également été décidé de conserver l'historique entier pour chaque *merge*. La figure 2 montre un aperçu de l'arborescence Git du projet.

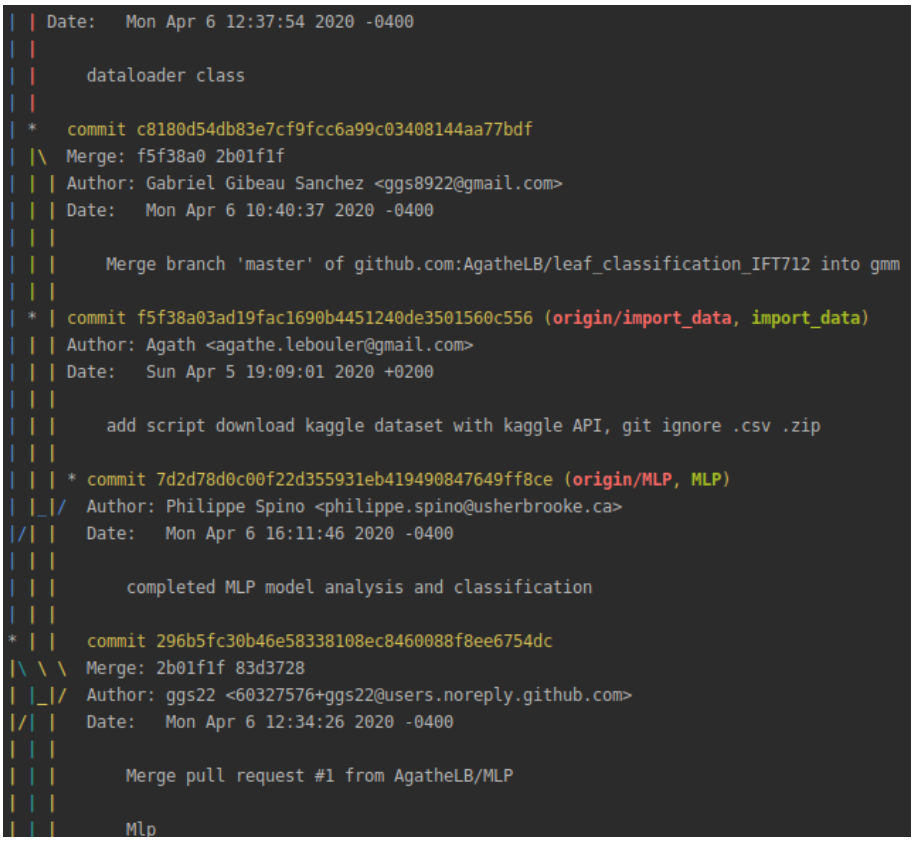

FIGURE 2 – Apperçu de l'arborescence du Git

8 Conclusion et pistes de recherche future

Notre équipe a pu développer un gestionnaire de classifieurs ainsi que leur implémentation propre. Il permet de faire une recherche d'hyper-paramètres permettant de maximiser le score de justesse du modèle. Celui-ci permettrait alors d'effectuer la classification des feuilles végétales présentes dans l'ensemble de données *leaf-classification* que l'on retrouve sur le site web **kaggle**.
En rétrospective, la plupart des modèles présentent de bonnes performances, avec souvent une justesse de 100% pour l'entraînement et majoritairement au-delà des 95% pour la validation. On dénote tout de même que les trois modèles les plus performants selon leur justesse de validation sont : *Random Forest*(98.48%), *LDA*(97.98%) et *KNN*(97.47%).

Une approche plus globale pourrait être envisagée afin d'obtenir des modèles plus performants avec une combinaison de modèles de classification. Chaque modèle serait entraîné séparemment puis sauvegardé. Pour les prédictions, un vote serait alors effectué entre les différents modèles afin d'établir la prédiction majoritaire. Pour le choix des modèles de classification à combiner, nous pourrions utiliser le même modèle selon différents hyper-paramètres. Afin d'avoir une combinaison de modèles la plus performante possible, des stratégies de *bagging* et *boosting* pourraient être utilisées selon le type de modèles utilisés (tendance à sur ou sous-apprendre).