

Techniques d'apprentissages IFT 712

# Projet de session

Decembre, 2020

---

<b><u>Equipe</u></b>	<b><u>Matricule</u></b>
Tariq CHAAIRAT	20 088 326
Oumaima EL HAYANI MOUSTAMIDE	20 134 999

<b>Introduction:</b>	<b>2</b>
<b>Démarche scientifique:</b>	<b>2</b>
Choix et Preprocessing des données (nettoyage et organisation):	2
On a ensuite 3 autres classes :	7
Choix et application des algorithmes de classification	7
<b>Analyse des résultats:</b>	<b>8</b>
Résultats sans la validation croisée:	8
Accuracy:	8
Log Loss:	9
F1-score:	10
Résultats avec la validation croisée:	10
Accuracy:	11
Log Loss:	11
F1-score:	12
Test sur Submission:	12
<b>Conclusion:</b>	<b>13</b>
<b>Lien Github du projet:</b>	<b>13</b>
<b>Source:</b>	<b>13</b>

## 1. Introduction:

L'apprentissage automatique (machine learning) est un vaste domaine d'étude qui permet, essentiellement, aux machines d'améliorer leurs performances à résoudre des tâches sans être explicitement programmées pour chacune, c'est-à-dire atteindre une bonne prédiction. Cet apprentissage comporte généralement deux phases:

- A. Estimer un modèle à partir des données d'entraînement lors de la phase de conception.
- B. Mise en production.

La classification est l'une des techniques de l'apprentissage automatique supervisé dont l'objectif principal est de prédire la classe d'appartenance de toute donnée d'entrée dans le processus de classification. Les méthodes de classifications comportent généralement trois phases : une première phase d'entraînement, une deuxième phase de validation et une troisième phase dédiée à la prédiction.

## 2. Démarche scientifique:

- Choix et Preprocessing des données (nettoyage et organisation):

On va d'abord se concentrer sur la compréhension de la base de données qu'on a choisi "Leaf Classification".

En effet, l'exploration des données nous permettra d'avoir une claire idée sur les modèles de classification à choisir due à sa nécessité.

Notre but est simplement de mieux comprendre et organiser les données avant d'appliquer les techniques de classification. Pour cela, on a créé une classe "Data\_Management" qui contient un ensemble de fonctions permettant de gérer les données.

La première fonction est la fonction `init_()` où on a importé les données "train" et "test" de la BDD leaf, puis on a utilisé **StratifiedShuffleSplit** pour diviser l'ensemble des données "train" en deux parties:

- X\_train et y\_train
- X\_test et y\_test

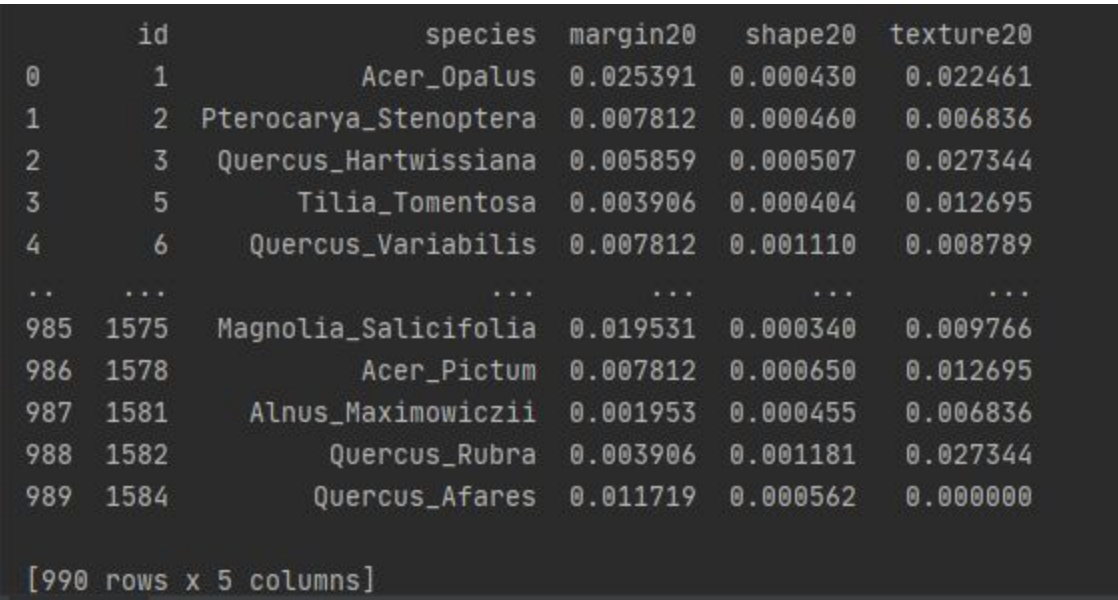
et **LabelEncoder** pour les organiser.

**Remarque: il y avait un problème concernant les chemins alternatifs pour l'importation des données train.csv et test.csv.**

**=> il faut mettre les chemins absolus pour que les l'importation s'effectue sans problème.**

1. `print_data()` pour afficher quelques données brutes de la base de données (figure 1).

Dès la première vue du résultat (figure 1), on constate que la première colonne décrit les identités des classes alors que la deuxième colonne correspond aux "species" des feuilles. En gros, on a trois caractéristiques différentes : les **margins**, les **shapes** et les **textures**. Cependant on notera que chacune de ces caractéristiques est composée de plusieurs sous-caractéristiques, par exemple shape comporte **shape1** jusqu'à **shape64**.



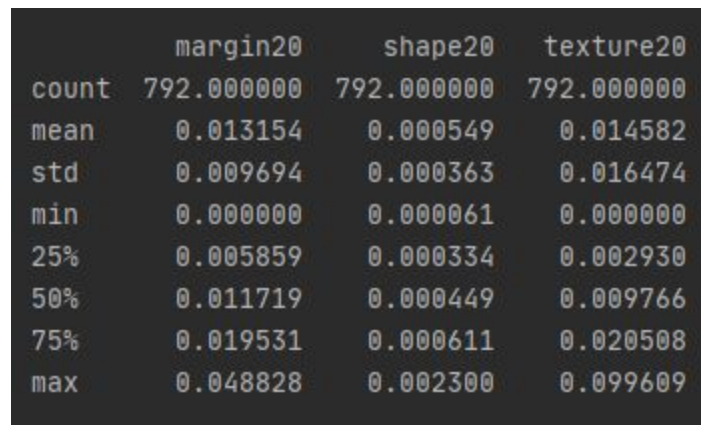
	id	species	margin20	shape20	texture20
0	1	Acer_Opalus	0.025391	0.000430	0.022461
1	2	Pterocarya_Stenoptera	0.007812	0.000460	0.006836
2	3	Quercus_Hartwissiana	0.005859	0.000507	0.027344
3	5	Tilia_Tomentosa	0.003906	0.000404	0.012695
4	6	Quercus_Variabilis	0.007812	0.001110	0.008789
..	...	...	...	...	...
985	1575	Magnolia_Salicifolia	0.019531	0.000340	0.009766
986	1578	Acer_Pictum	0.007812	0.000650	0.012695
987	1581	Alnus_Maximowiczii	0.001953	0.000455	0.006836
988	1582	Quercus_Rubra	0.003906	0.001181	0.027344
989	1584	Quercus_Afares	0.011719	0.000562	0.000000
[990 rows x 5 columns]					

Figure 1: Résultats de la fonction print\_data()

2. La fonction `get_shape()` permet de savoir les dimensions des données, autrement dit savoir le nombre de lignes x le nombre de colonnes dans la base de données:  
Les données "train" ont pour dimensions 990 x 194. 990 est le nombre de species et 194 est le nombre des caractéristiques individuelles dans les colonnes i.e. 'id', 'species', 'margin 1-64', 'shape1-64', 'texture1-64'. Et d'après la fonction info() de la librairie Pandas on peut savoir le types des données: {id=> int64}, [[margin 1-64, shape 1-64, texture 1-64]=> float64], {species => object/category}

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 990 entries, 0 to 989  
Columns: 194 entries, id to texture64  
dtypes: float64(192), int64(1), object(1)
```

3. Ensuite, on a affiché les statistiques en relation avec les données telles que la moyenne, la déviation standard, les percentiles, le minimum, le maximum..., avec la fonction `data_desc()` qui applique la fonction `describe()` de la librairie Pandas (figure 2)



	margin20	shape20	texture20
count	792.000000	792.000000	792.000000
mean	0.013154	0.000549	0.014582
std	0.009694	0.000363	0.016474
min	0.000000	0.000061	0.000000
25%	0.005859	0.000334	0.002930
50%	0.011719	0.000449	0.009766
75%	0.019531	0.000611	0.020508
max	0.048828	0.002300	0.099609

Figure 2: Résultats de la fonction `data_desc()`

Les caractéristiques **textures** ont une variation beaucoup plus importante comparées aux caractéristiques **margin** et **shape**, alors que sa valeur moyenne est presque comparable à celle du **margin** mais beaucoup plus élevé que la moyenne du **shape**. Ce qui nous mène à conclure qu'une transformation de données brutes est nécessaire avant l'application des algorithmes de classification. Cette transformation vise à mieux centrer la distribution conduisant à une meilleure "accuracy" de ces algorithmes.

4. La fonction `scale()` qui transforme les données dans l'intervalle [0,1] à l'aide de la fonction `preprocessing.MinMaxScaler()` de sklearn.
5. Il est nécessaire de connaître le nombre de classe (species) dans les données d'entraînement pour les algorithmes de classifications, donc on utilise la fonction `class_distribution()` qui retourne le nombre d'instance dans chaque species:

```
species
Acer_Capillipes      10
Acer_Circinatum      10
Acer_Mono             10
Acer_Opalus          10
Acer_Palmatum         10
..
Tilia_Tomentosa      10
Ulmus_Bergmanniana   10
Viburnum_Tinus       10
Viburnum_x_Rhytidophylloides  10
Zelkova_Serrata       10
Length: 99, dtype: int64
```

Figure 3: Résultats de la fonction `class_distribution()`

L'utilisation de la fonction `groupby()` nous permet d'identifier que dans les données feuilles, les objets sont distribués d'une manière équitable dans les 99 classes (10 objets dans chaque classe)

6. L'étude de corrélation entre les données est aussi une étapes importante dans un projet d'apprentissage automatique:

```
margin10  shape10  texture10  margin20  shape20  texture20
margin10    1.000   -0.009    0.101    0.620    0.026   -0.124
shape10    -0.009    1.000   -0.022    0.004    0.809    0.059
texture10    0.101   -0.022    1.000    0.210   -0.004   -0.253
margin20    0.620    0.004    0.210    1.000    0.053   -0.155
shape20    0.026    0.809   -0.004    0.053    1.000    0.014
texture20   -0.124    0.059   -0.253   -0.155    0.014    1.000
None
```

Figure 4: Résultats de la fonction `correlation_matrix()` qui calcul la matrice de corrélation

D'après la figure 4, on peut bien remarquer qu'il y a une forte corrélation entre les mêmes caractéristiques et une faible, et même négative dans des cas, corrélation entre les caractéristiques différentes ( exemple: corrélation négative entre shape10 et margin10).

- **Visualisation des données:**

La visualisation graphique est aussi importante pour une vision plus claire de la dispersion des données et la distribution des caractéristiques dans la BDD.

Ainsi la figure 5 est le résultat de la fonction `visualization()` où on a utilisé la fonction `seaborn.pairplot()` qui nous permet d'expliciter les relations entre les données par paires.

D'après ce résultat, on peut conclure que les données sont gaussiennes pour la caractéristique *shape*, mais cela n'est pas assez clair pour les deux autres.

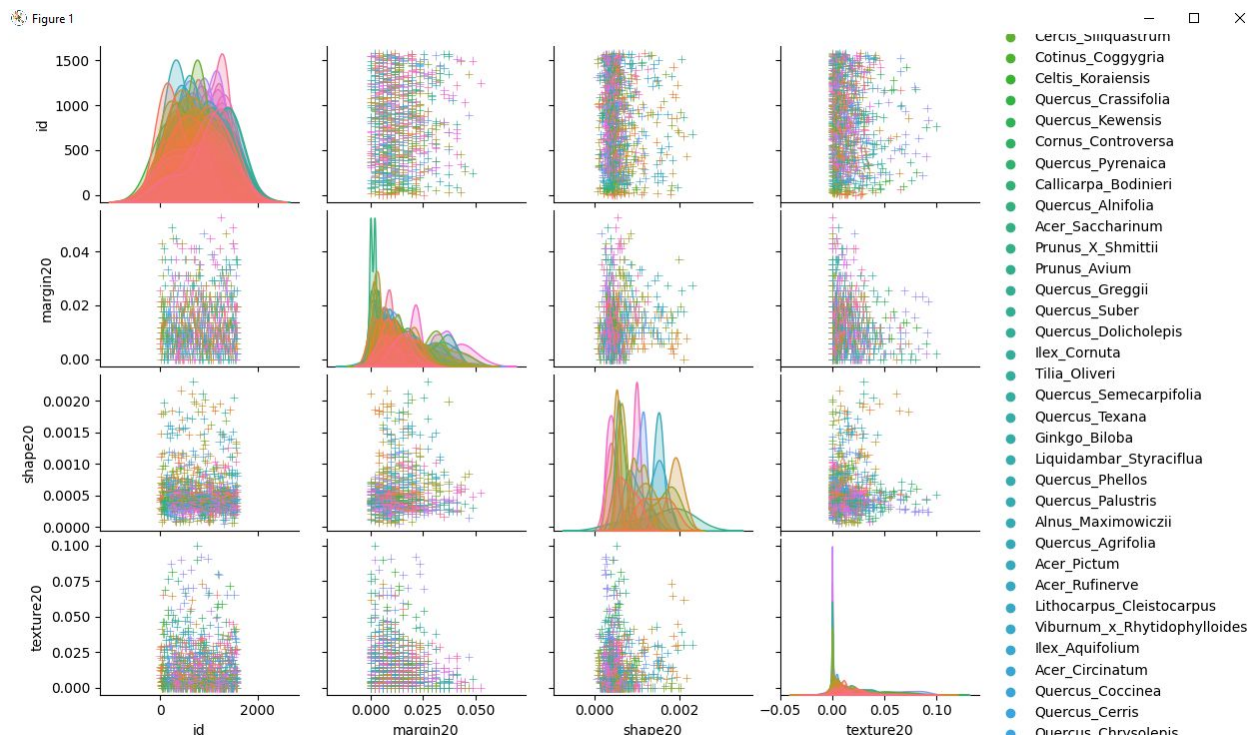


Figure 5: Résultats de la fonction `visualization()`

On a ensuite 3 autres classes :

- ❑ **Cross\_Validation** : cette classe héritant de **Data\_Management** gère la recherche d'hyper-paramètres pour tous les classifieurs (on s'est limité à 2 paramètres par classifieurs, on explique la raison plus loin dans le rapport) via la validation croisée
- ❑ **Classifieurs** : cette classe hérite de **Cross\_Validation**, elle gère l'initialisation des classifieurs avec les paramètres par défauts dans le cas où le **booléen cross\_val** est à False. Dans le cas contraire, elle les initialise avec les meilleurs paramètres issus des validation croisée faites dans la classe antérieure.
- ❑ **Graphique**: qui gère le calcul des métriques et l'affichage de celles-ci sous forme d'histogramme qui représente les résultats avec les données transformées et les données non transformées(cette classe hérite de Classifieurs)
  - **Choix et application des algorithmes de classification**

Afin de tester l' "accuracy", "log loss" et "f1-score" des données, on doit choisir d'abord les algorithmes de classification qui vont nous fournir les meilleurs résultats.

Donc dans ce contexte, les six algorithmes choisis sont:

**I. SVM:**

Support Vector Machine — Machine à vecteurs de support

**II. KNN:**

K Nearest Neighbour — K plus proches voisins

**III. LR:**

Logistic Regression — Régression logistique

**IV. NN:**

Neural Networks — Réseaux neurones

**V. ADA:**

AdaCoostClassifier — Classificateur AdaBoost

**VI. RF:**

Random Forest — Forêt aléatoire



### 3. Analyse des résultats:

Les métriques utilisés pour faire les tests sur l'ensemble de données de validation qui font partie des données d'entraînement sont: **"Accuracy"** ou **"Justesse"**, **"Log Loss"** et **"F1-Score"**.

Ces tests ont pour but de trouver les meilleurs hyperparamètres de chaque algorithme de classification utilisé, d'où l'utilisation de la validation croisée.

Pour mettre en évidence les résultats obtenus, on a testé d'abord les algorithmes sans validation croisée sur les données transformées (scaled) et non transformées (non scaled), ensuite, on a affiché les résultats dans le même histogramme.

#### Résultats sans la validation croisée:

##### □ Accuracy:

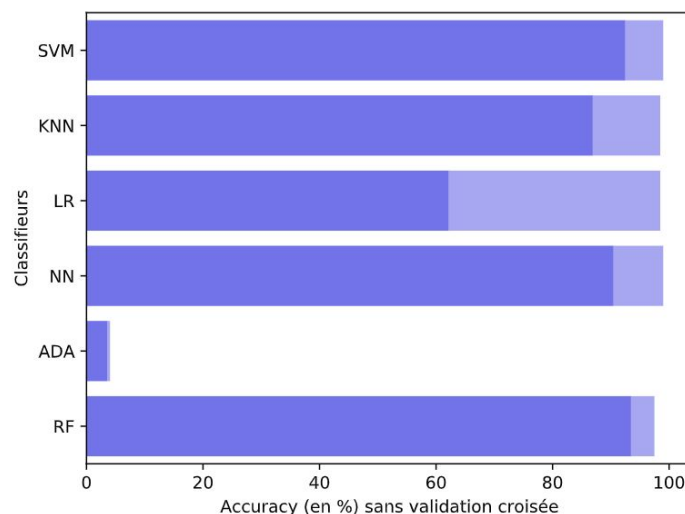


Figure 6: Accuracy des six algorithmes en % avec test sur les données

non transformées en bleu foncé et sur les données transformées en bleu pâle

=>L'histogramme obtenu contient 2 types de barre: Les barres en bleu clair représentent les données transformées alors que les barres en bleu foncé représentent les données non transformées. Comme on l'a déjà mentionné, la fonction `scale()` transforme les données de telle façon qu'elle retourne des valeurs dans l'intervalle [0,1].

On constate, d'après les résultats numériques qui suivent l'histogramme 6, que cette transformation a diminué l'accuracy du classificateur RF et ADA. Cependant, elle était favorable pour tous les autres classificateurs d'après l'augmentation remarquée dans la valeur de l'accuracy.

Donc l'algorithme le plus performant, c'est celui dont l'accuracy est la plus proche de la moyenne des accuracy des six classificateurs.

❑ Log Loss:

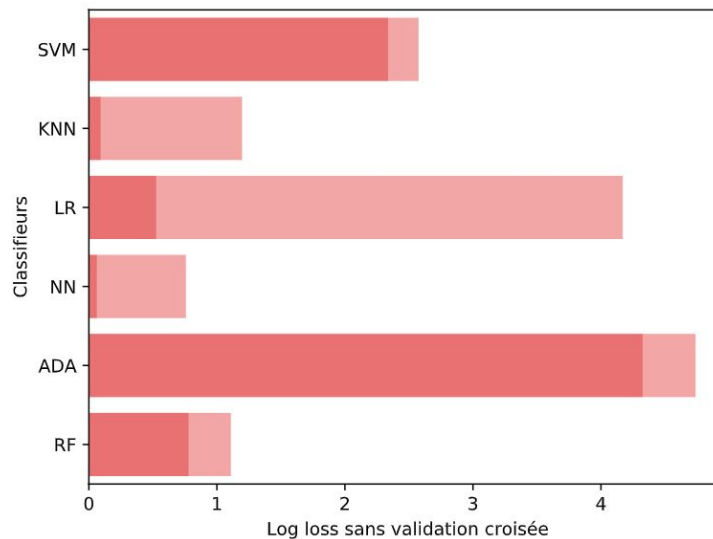


Figure 7: Log loss des six algorithmes avec test sur les données

*non transformées en rouge foncé et sur les données transformées en rouge pâle*

=> De la même façon, on fait le test pour “Log Loss”, avec test sur les données non transformées en rouge foncé et sur les données transformées en rouge pâle.

Donc on peut clairement remarquer qu’avec la transformation des données, Log loss a augmenté pour les classifieurs SVM, ADA et RF et a diminué pour KNN, NN et LR.

#### □ F1-score:

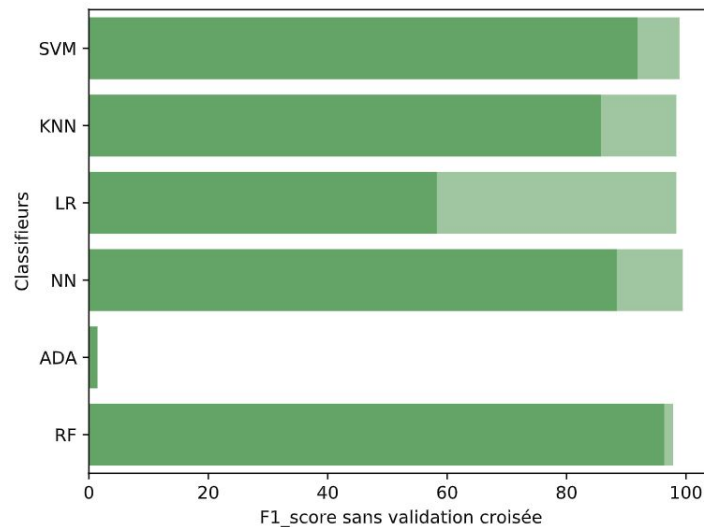


Figure 8: F1\_score des six algorithmes avec test sur les données

non transformées en vert foncé et sur les données transformées en vert pâle

=> On fait le test pour "F1\_score", avec test sur les données non transformées en vert foncé et sur les données transformées en vert pâle.

Après avoir effectué le même test qu'on a fait pour les autres métriques (accuracy et log loss), on remarque que le F1\_score augmente pour tous les classificateurs sauf pour le ADA et RF qui diminue.

#### **Résultats avec la validation croisée:**

Le but de tester les données avec une validation croisée est d'obtenir les meilleurs hyper-paramètres pour chaque classificateur. On a ici choisi de rechercher maximum 2 hyper-paramètres par classifieurs afin d'éviter que cette recherche ne prenne trop de temps. Une possible évolution du projet consisterait à moduler cette recherche d'hyper-paramètres dans un premier temps afin d'avoir la meilleure combinaison de paramètres possible ou encore de rechercher tous les paramètres jugés intéressants par classifieurs mais là on mettrait à profit la précision au détriment du temps de recherche des paramètres.

### □ Accuracy:

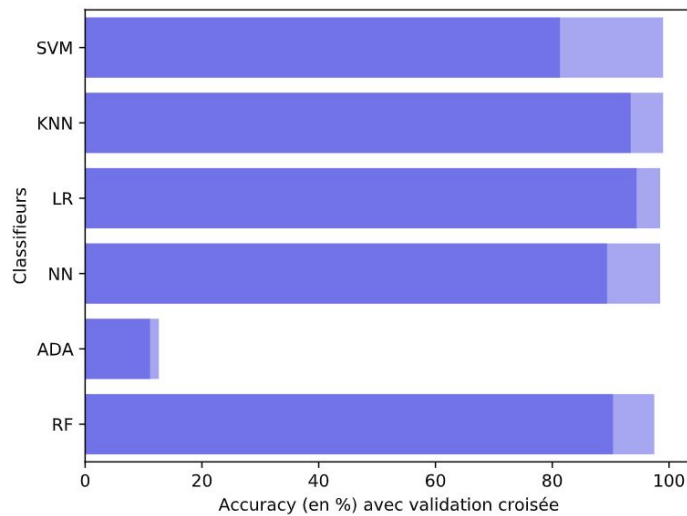


Figure 9: Accuracy des six algorithmes en % avec test sur les données

*non transformées en bleu foncé et sur les données transformées en bleu pâle*

=>D'après une revue rapide des résultats, on constate que cette fois l'“accuracy” augmente pour tous les classifieurs sauf le RF (elle diminue pour RF).

### □ Log Loss:

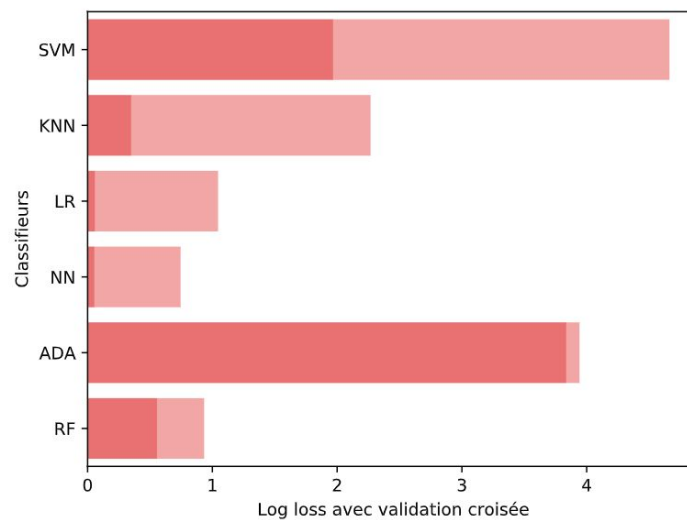


Figure 10: Log loss des six algorithmes avec test sur les données

*non transformées en rouge foncé et sur les données transformées en rouge pâle*

=> Log Loss diminue avec la transformation des données pour les classificateurs SVM, KNN, LR et NN et augmente pour les classificateurs ADA et RF.

#### ❏ F1-score:

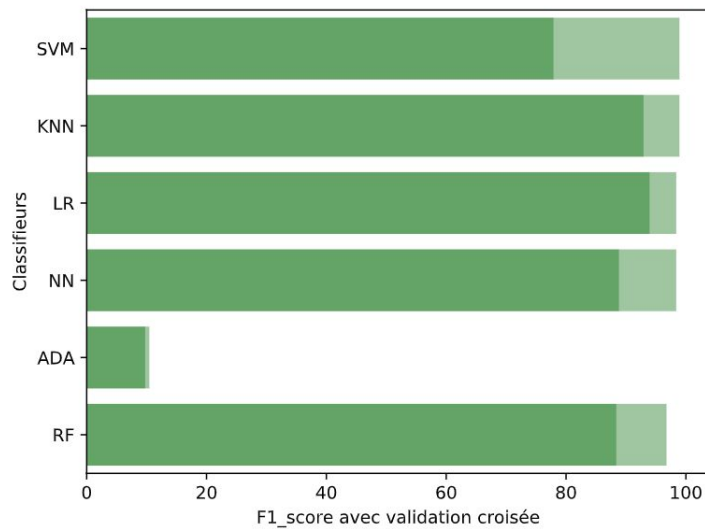


Figure 10: F1\_score des six algorithmes avec test sur les données

non transformées en vert foncé et sur les données transformées en vert pâle

=> Le F1\_score augmente avec la transformation des données pour les classificateurs SVM, KNN, LR, ADA et NN et diminue pour le classificateur RF.

#### Test sur Submission:

Un dernier test est fait sur les résultats du classificateur NN (étant le classificateur qui a la meilleure valeur d'accuracy) en les déposant sur Kaggle.

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
submission.csv	2 minutes ago	0 seconds	0 seconds	0.08320
Complete				

Figure 11: Résultat du test dans le site Kaggle pour l'algorithme NN

Le score obtenu, qui représente la "loss", est de 0.08320 ce qui est un peu proche du résultat du test avec validation croisée => avec données transformées du Log Loss pour NN. Mais en

---

comparaison nos résultats avec les résultats présentés dans le site Kaggle sont assez loin en termes de classement.

#### 4. Conclusion:

Tout au long de ce projet, on a appliqué six algorithmes de classifications sur la base de données “Leaf-Classification” de Kaggle, puis on a testé pour les données respectivement transformées et non transformées sans validation croisée en premier temps et avec la validation croisée après.

Les résultats obtenus montrent qu’il y a dans la plupart du temps une augmentation dans les valeurs de métriques après la transformation des données et rarement une diminution ou elles restent constantes.

Et finalement, on peut conclure que les meilleurs résultats sont ceux obtenus en passant par la transformation des données suivie de la validation croisée.

#### 5. Lien Github du projet:

<https://github.com/togakhi/linearClassificationAlgorithms>

#### 6. Source:

- <https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn>
- <https://scikit-learn.org/stable/modules/preprocessing.html>
- [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.StratifiedShuffleSplit.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html)
- <https://scikit-learn.org/stable/index.html> (pour les algorithmes de classification)