
Projet de fin de session



UNIVERSITÉ DE
SHERBROOKE

TECHNIQUE D'APPRENTISSAGE
IFT 712

Étudiants (CIP) :

Corentin POMMELEC (pomc0601)

Chaimae TOUBALI (touc1402)

Tahar AMAIRI (amat0601)

Enseignant :

Pr. Martin Vallières

12 décembre 2022

Lien GitHub : <https://github.com/chaimae2000/Project-IFT712>

Table des matières

1	Introduction	2
2	Description du dataset	2
3	Démarche scientifique	3
3.1	Préparation des données	3
3.2	Modèles utilisés	4
3.2.1	Modèle basique	4
3.2.2	Modèle avancé	5
4	Implémentation	6
4.1	Structure du code	6
4.2	Format de configuration	7
5	Comparaison des résultats	9
5.1	Choix des métriques	9
5.2	Résultats obtenus	10
5.2.1	Sans modification des caractéristiques	10
5.2.2	Avec analyse en composantes principales	11
5.2.3	Avec sélection de caractéristiques	11
6	Gestion de projet	12
6.1	GitHub	12
6.2	Jira	13
7	Conclusion	14
8	Annexes	16
8.1	Erreur d'entraînement	16
8.2	Métriques de test	17
8.3	Courbe d'apprentissage	20

1 Introduction

Le projet du cours de technique d'apprentissage nous propose de tester différentes méthodes de classifications de données supervisées. Pour cela, nous utiliserons une base de données présente sur le site Kaggle provenant d'un challenge de classification de feuilles d'arbres. Cet ensemble de données est construit à partir d'images de spécimens de feuilles provenant de différentes espèces de plantes d'origine. De nombreuses caractéristiques nous sont fournies et permettent de définir chaque image.

Ainsi, à l'aide de la bibliothèque Scikit-learn destinée à l'apprentissage automatique, nous ferons appel à plusieurs algorithmes de classifications. Ces derniers nous permettront d'entraîner et de comparer l'efficacité de modèles capables de prédire l'appartenance d'une feuille à une espèce de plantes.

2 Description du dataset

L'observation est née d'une volonté de préciser quelle est l'espèce de l'arbre à laquelle appartient une feuille donnée de notre dataset, tel que les caractéristiques extraites des images des feuilles sont essentiellement 3 vecteurs de dimension 64 (margin, shape & texture). L'ensemble de données comprend environ 1 584 images de spécimens de feuilles (16 échantillons de chacun 99 espèces) converties en feuilles noires binaires sur fond blanc. Trois ensembles de caractéristiques sont également fournis par l'image : un descripteur contigu de forme, un histogramme de texture intérieure et un histogramme de marge à échelle fine. Pour chaque entité, un vecteur de 64 attributs est donné par échantillon de feuille.

Le jeu de données contient comme attributs :

- `id` : un identifiant correspondant à une image.
- `species` : à quel espèce de feuille appartient l'échantillon.
- `margin_1`, `margin_2`, `margin_3`, ..., `margin_64` : chacun des 64 vecteurs d'attributs pour la fonction de marge.
- `shape_1`, `shape_2`, `shape_3`, ..., `shape_64` : chacun des 64 vecteurs d'attributs pour l'entité de forme.
- `texture_1`, `texture_2`, `texture_3`, ..., `texture_64` : chacun des 64 vecteurs d'attributs pour la fonction de texture.

De plus, les données se composent de 990 lignes et 194 caractéristiques. Concernant la distribution des 99 espèces, chacune contient 10 éléments, donc les données sont distribuées équitablement.

3 Démarche scientifique

3.1 Préparation des données

Une règle bien connue en technique d'apprentissage est qu'environ 80% du temps est passé à rechercher les données, les nettoyer et les organiser. Comme nous l'avons détaillé, dans notre cas, nous avons l'avantage de travailler sur un jeu de données pré-conçu. En effet, d'une part des données cohérentes nous ont été fournies, mais en plus elles ont déjà été vérifiées et prétraitées comme nous l'indique le fait qu'il y ait eu l'élimination d'une espèce d'origine en raison de données incomplètes. Nous avons donc pu effectuer nos étapes de pré-processing en prenant en compte ce constat.

Globalement, les étapes à effectuer sont :

- La séparation aléatoire du jeu de données en deux parties, une partie va être utilisée pour l'entraînement de notre modèle et une autre partie va être utilisée pour tester l'efficacité de ce dernier.
- Gérer les données manquantes d'une façon ou d'une autre.
- Gérer les caractéristiques utilisées, c'est-à-dire notamment transformer les données catégoriques en données numériques afin qu'elles soient plus facilement manipulables. La réduction de dimensionnalité est également à envisager.
- Mettre les données à la même échelle en conservant l'écart entre chaque données en supprimant ou non les données aberrantes.

Ainsi, en premier lieu, nous effectuons la séparation de notre jeu de données et de test que nous utiliserons tout au long du projet.

Ensuite, compte tenu de notre jeu de données, l'ensemble est complet, il n'est donc pas nécessaire de le nettoyer des entrées vides. Il n'y a aucune valeur nulle pour la forme, la texture et la marge de l'entité.

Nous convertissons par la suite les étiquettes de classe en données. En ce qui concerne la réduction de dimensionnalité, nous testerons les différentes performances de nos modèles après une ACP et après une sélection des caractéristiques. L'ACP désigne une méthode de réduction de dimensions permettant de transformer des variables très corrélées en nouvelles variables dé-corrélées les unes des autres. L'information est ainsi résumée en un nombre restreint de variables synthétiques : les composantes principales. Dans notre cas, nous effectuerons un grid search pour déterminer le juste pourcentage d'informations expliquées (50%, 75% ou 85%) à conserver afin d'obtenir le meilleur apprentissage pour nos modèles. Finalement, l'ACP a pour conséquence attendue de simplifier nos données et ainsi d'améliorer la performance de certains algorithmes et leur rapidité d'exécution. La sélection de caractéristiques, quant à elle, consiste à trouver un sous-ensemble de variables pertinentes. On cherche alors à minimiser la perte d'information provenant de la suppression de variables. Pour faire cela, nous prenons en compte le poids accordé par chaque modèle aux différentes caractéristiques.

Ceci est rendu possible grâce à la méthode d'analyse de régression Lasso utilisant un terme de régularisation alpha que nous faisons varier (1.0, 2.5 ou 5.0) afin de conserver la valeur qui mène à la meilleure sélection pour le modèle concerné. L'objectif à terme est de chercher à simplifier le modèle, de réduire la durée de l'entraînement et d'améliorer la généralisation.

Enfin, il existe plusieurs façons de standardiser les données, notamment en fonction de la volonté de garder ou non les données aberrantes. Dans notre cas, conformément aux modèles que nous utiliserons, nous laissons la possibilité d'utiliser ou non la standardisation via le fonctionnement de la classe `StandardScaler` de scikit-learn. Cette dernière procède en supprimant la moyenne et en mettant à l'échelle la variance unitaire. Cela a pour effet de mettre les données à la même échelle, tout en conservant les informations des données aberrantes, mais en y étant moins sensible.

3.2 Modèles utilisés

Nous avons implémenté un total de 11 modèles que nous pouvons classer en deux catégories différentes : les modèles basiques et les modèles avancés. Les derniers permettent une différente approche des modèles basiques leurs permettant d'obtenir de meilleurs résultats.

3.2.1 Modèle basique

- **LDA & QDA** : L'analyse discriminante linéaire & quadratique sont deux modèles prédictifs se basant sur le théorème de Bayes et sont une extension du discriminant linéaire de Fisher. Ils n'utilisent aucun hyper-paramètres, cependant, ils se basent sur des hypothèses de normalité et d'homoscédasticité. Comme leur nom l'indique, LDA permet d'obtenir une surface de décision linéaire et QDA une surface de décision quadratique. Dans notre implémentation, nous n'avons pas vérifié les hypothèses, mais il existe de nombreux tests de normalité multidimensionnelle disponibles : Mardia test, Henze-Zirkler test, Royston test etc...
- **Régression logistique** : La régression logistique est un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien. Comme l'indique la documentation scikit-learn, la régression logistique nécessite une standardisation des données avant d'entraîner le modèle. La régression logistique implémentée par scikit learn utilise le gradient moyen stochastique afin de minimiser la perte. Son implémentation utilise une méthode qui lui est propre pour obtenir le taux d'apprentissage, telle que nous n'ayons pas à le spécifier. Ainsi, il n'y a aucun d'hyperparamètre à trouver pour ce modèle. On lui applique donc seulement un coefficient de régularisation C.
- **Decision Tree** : Comme son nom l'indique, ce modèle prédictif repose sur l'utilisation d'un arbre de décision. L'analogie d'un arbre se poursuit avec les feuilles qui sont les valeurs de la variable cible et les embranchements qui correspondent à des combinaisons de variables d'entrée qui mènent aux feuilles. Le decision tree possède plusieurs divers hyperparamètres. Dans notre cas, nous avons décidé de chercher à optimiser :

- Le critère, c'est à dire la fonction, "gini" ou "entropy", pour calculer l'incertitude
- Le nombre minimum de samples requis pour arriver à une feuille.
- Le nombre minimum de samples requis pour créer une règle de décision.
- La profondeur maximale de l'arbre.
- **KNN** : est un modèle discriminant d'apprentissage supervisé non paramétrique, qui utilise la proximité pour effectuer des classifications ou des prédictions sur le regroupement d'un point de données individuel. Bien qu'il puisse être utilisé pour des problèmes de régression ou de classification, il est généralement utilisé comme algorithme de classification, en partant de l'hypothèse que des points similaires peuvent être trouvés les uns à côté des autres. Par ailleurs, La valeur k dans l'algorithme k-NN définit le nombre de voisins qui seront vérifiés pour déterminer la classification d'un point de requête spécifique. De ce fait avant de créer une baseline de performance avec KNN , on cherche le K optimal pour une classification K-NN afin d'identifier le score correspondant.
- **SVM** :
 - **SVM Linear** : un modèle d'apprentissage automatique supervisé qui peut être utilisé pour les problèmes de classification ou de régression. Toutefois, il est surtout utilisé dans les problèmes de classification.
 - **SVM One Vs All** : Cette approche consiste à créer autant de SVM que de catégories présentes. De ce fait on applique donc le principe One against All à une classification linéaire SVM, en testant les paramètres :
 - Le coefficient de régularisation C.
 - Penalty L1 ou L2.
 - multi_class : "ovr" ou "rammer_singer".

3.2.2 Modèle avancé

Au-delà des modèles d'apprentissage "de base", il existe différentes méthodes permettant d'optimiser et d'améliorer ces derniers. Nous en avons sélectionné certaines. Parmi elles, nous pouvons citer les méthodes ensemblistes qui sont basées sur la volonté de combiner les prédictions de plusieurs classifieurs pour mieux généraliser et compenser les défauts existant sur les prédicteurs individuels.

- **Random Forest** : La forêt d'arbre aléatoire est une méthode d'ensemble qui fonctionne en créant une multitude d'arbres de décisions pour l'entraînement. Chaque arbre a un échantillon aléatoire indépendant différent. Random forest permet en général, d'améliorer les performances et de rendre les prédictions plus stables, mais ralentit également la vitesse de calcul. La forêt a presque les mêmes hyperparamètres qu'un arbre de décision, dans notre cas :
 - Le nombre minimum de samples requis pour arriver à une feuille.
 - Le nombre minimum de samples requis pour créer une règle de décision.
 - La profondeur maximale de l'arbre.
 - Le nombre d'arbres dans la forêt.

- **SGD** : Le classifieur par descente de gradient stochastique n'en est pas réellement un. En effet, ce dernier est plus exactement l'optimisation d'un classifieur linéaire. Il fonctionne ainsi sur différents classifieurs linéaires tels que SVM et la logistique régression. Nous l'utiliserons avec l'hyper paramètre alpha qui est une constante multipliant le terme de régularisation.
- **AdaBoost & Bagging** : Ces deux modèles font parties de l'apprentissage ensembliste : l'idée ici est de combiner plusieurs algorithmes de base pour former un algorithme prédictif optimisé. Dans notre cas, ces deux modèles seront utilisés avec la decision tree.
 - **Adaboost** permet de diminuer le biais en tentant de construire un modèle prédictif à partir des erreurs de plusieurs modèles plus faibles. On commence par créer un premier modèle ensuite un second, en essayant de réduire les erreurs du modèle précédent etc... Les hyper-paramètres qu'on a modifié sont :
 - Le nombre de decision tree utilisé par Adaboost, ce qui revient à modifier le nombre d'itération du modèle.
 - Le taux d'apprentissage.
 - Et finalement, la profondeur du decision tree utilisé par Adaboost.
 - **Bagging** permet de diminuer la variance d'un modèle basique en introduisant une randomisation dans sa procédure de construction. C'est un méta-estimateur qui entraîne un modèle basique à des sous-ensembles aléatoires de l'ensemble de données d'origine, puis agrège leurs prédictions individuelles (soit par vote, soit par moyenne) pour former une prédiction finale. Par ailleurs, la random forest rentre aussi dans cette catégorie ! Les hyper-paramètres qu'on a modifié sont :
 - La profondeur du decision tree utilisé par le classifieur Bagging.
 - Le nombre de decision tree utilisé par le modèle.
 - Le nombre d'échantillons à tirer du jeu de données pour entraîner chaque decision tree.
 - Et en complément, le nombre d'attributs à tirer du jeu de données pour entraîner chaque decision tree.

4 Implémentation

4.1 Structure du code

La structure de notre implémentation repose sur deux principes fondamentaux : le premier est d'utiliser au maximum la librairie sklearn afin d'éviter du code inutile et le second est qu'elle puisse être facilement scalable pour fonctionner avec tous les algorithmes de sklearn mais aussi avec différentes données. Pour cela, notre implémentation s'articule autour de 3 modules :

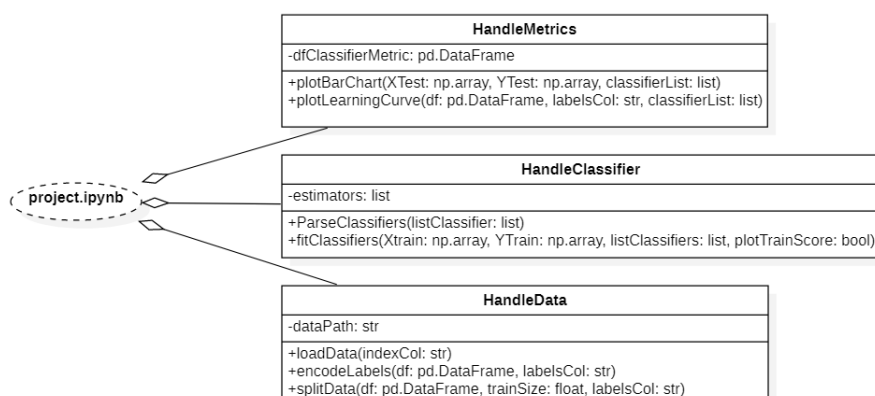


FIGURE 1 – Diagramme de l'implémentation

Le premier module `handleData` permet de charger les données en un data frame et de gérer différents aspects de celui-ci : changement des noms des labels en entier, séparer les données afin d'obtenir des données d'entrainements et de tests etc. . . Le second module `handleClassifier`, quant à lui, permet à l'aide d'une liste de configuration d'entraîner un modèle de `sklearn`. Pour cela, il utilise en interne une pipeline et un import automatique de la classe du modèle à entraîner. Suivant le fichier de configuration, le module peut effectuer plusieurs choses :

- effectuer des étapes de pre-processing : standardisation des données, réduction ou sélection des attributs. . .
- validation croisée
- recherche d'hyper paramètres à l'aide d'une croise validation combinée avec un grid search
- affichage du score de training en histogramme

Finalement, le dernier module `handleMetrics`, permet d'obtenir certaines métriques concernant la prédiction des modèles entraînés sur les données tests. Notre architecture est simple mais c'est dans cette simplicité qu'elle trouve son adaptabilité : il n'est pas nécessaire de créer une classe pour entraîner chaque modèle. En effet, car le module `handleClassifier` joue l'interface entre `sklearn` et ce que l'on veut effectuer. Pour cela, il traduit le format de configuration en une exécution de fonction `sklearn`.

4.2 Format de configuration

A l'aide de la méthode `fitClassifiers`, le module `handleClassifier` peut entraîner une liste de modèle. Pour cela, cette fonction prend en paramètre une liste de dictionnaire permettant de décrire la configuration dans laquelle on veut entraîner le modèle. Il est donc nécessaire d'utiliser un format mixant la syntaxe des fonctions de `sklearn` et une syntaxe inventée par nos soins :


```

{
    # sklearn classifier name (case sensitive !)
    "name": str,
    # a dict containing the parameters of the init function of the selected classifier
    # (need to match the sklearn class object init function)
    "config": dict, (optional)
    # to standardize, need to be set, otherwise an error is raised
    "preprocess": bool,
    # for feature selection or reduction
    "feature" : {
        # the feature method : "reduction" or "selection" (case sensitive !).
        # If not set correctly, an error is raised
        "option": str,
        # a dict containing the parameters of :
        # sklearn PCA function if option was set to reduction
        # sklearn SelectFromModel function if option was set to selection
        "config": dict
    }, (optional)
    # a dict containing the fit strategy, need to be set, otherwise an error is raised
    "fitStrategy": {
        # the validation method : "GridSearch" or "CV" (case sensitive !).
        # If not set correctly, an error is raised
        "option": str,
        # a dict containing the parameters of :
        # sklearn cross_validate function if option was set to CV
        # sklearn GridSearchCV function if option was set to GridSearch
        # Also, need at least the scoring parameter to be set
        # otherwise, an error is raised
        "config": dict
    }
}
}

```

A l'aide de ce dictionnaire, une pipeline est construite en interne pour chaque modèle :

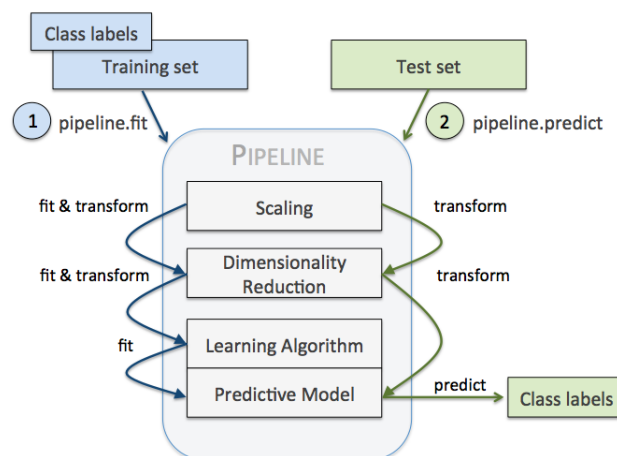


FIGURE 2 – Sklearn Pipeline

Celle-ci possède cette forme :

```
pipeline = Pipeline(steps=[("preprocess", .), ("ftr", .), ("clf", .)])  
# with prefixes :  
# preprocess : standardization step  
# ftr : feature selection or reduction step  
# clf : sklearn classifier object
```

Par conséquent, pour fixer les hyper-paramètres à tester lors de la grid search, on utilise les préfixes décrits auparavant comme ceci :

```
{  
    "name": "LogisticRegression",  
    ...  
    "param_grid": {  
        "clf__C": [0.01, 0.05, 0.1, 0.5, 1.0, 1.5, 2.0]  
    }  
    ...  
}
```

Dans cet exemple, on modifie le terme de régularisation d'une régression logistique lors d'une grid search.

5 Comparaison des résultats

5.1 Choix des métriques

Dans le but d'analyser au mieux la performance de nos algorithmes de classification, notamment pour pouvoir à termes les comparer entre eux, nous avons sélectionné diverses métriques adaptées. Ces dernières sont :

- L'accuracy, le rapport entre l'observation correctement prédite et le nombre total d'observations.
- La précision, le rapport entre les observations positives correctement prédites et le nombre total d'observations positives prédites.
- Le rappel, le rapport entre les observations positives correctement prédites et toutes les observations de la classe réelle
- Le F1-score, la moyenne pondérée de la précision et du rappel.
- L'aire sous la courbe ROC (Area Under The Curve, AUC). La courbe ROC montre le rapport entre le taux de vrais positifs du modèle et le taux de faux positifs. l'AUC, fournit une mesure globale des performances pour tous les seuils de classification possibles. Cette métrique est obtenue à partir d'un vecteur, contenu par le modèle après entraînement, qui renferme les probabilités d'appartenance à chaque classe pour chaque point de données. Cependant, tous les modèles ne possèdent pas ce vecteur, nous ne pouvons donc pas calculer l'AUC, ce sont ceux qui sont à nul sur le graphique.

Ces métriques sont parmi les plus utilisées en machine learning. Nous les avons choisies car elles permettent d'obtenir suffisamment d'informations pour avoir une vision globale permettant d'interpréter la performance de nos modèles. Nous avons, également, fait le choix de les afficher sous la forme de diagrammes à bande afin de faciliter leur lisibilité.

Cependant, nous ne devons pas oublier d'évaluer le sous-apprentissage et le sur-apprentissage de nos modèles. Pour cela, nous affichons la courbe d'apprentissage de chacun d'entre eux. Cette dernière est un tracé la valeur optimale de la fonction de perte d'un modèle pour un ensemble d'entraînement par rapport à cette fonction de perte évaluée sur un ensemble de test avec les mêmes paramètres que ceux produits par la fonction optimale.

5.2 Résultats obtenus

Pour effectuer cette analyse de résultats, nous nous appuyons sur les graphiques en annexes : [erreur d'entraînement](#), [métriques de test](#) et [courbe d'apprentissage](#).

5.2.1 Sans modification des caractéristiques

Globalement, nous remarquons que de nombreux modèles ont été très performants. En effet, sept classifieurs possèdent des scores au-dessus de 90% pour toutes les métriques de test, c'est le cas notamment des modèles dit linéaire tels que la régression logistique ou SVC. De plus, l'excellente performance du modèle LDA nous amène à penser que les données suivent une normale multidimensionnelle.

Ensuite, on retrouve comme attendu, la plupart des algorithmes d'optimisation, des modèles de régression linéaire, comme la régression logistique par descente de gradient stochastique, par exemple, qui possède elle aussi d'excellents résultats.

Moins attendu, les algorithmes ayant comme base les arbres de décision tels que random forest performant très bien malgré que le modèle d'arbre de décision classique, lui, reste moyen, c'est à dire entre 60% et 80% de score sur les différentes métriques. Un autre modèle ne correspondant pas à ces critères ne démerite pas, c'est l'algorithme K-NN qui obtient plus de 85% de score sur toutes les métriques.

Cependant, un des modèles surprend par ses résultats très médiocre aux alentours de 4%, c'est le cas du quadratic discriminant analysis. Une des premières pistes privilégiées est le sous-apprentissage. Cette hypothèse peut être rapidement confirmée en observant sa courbe d'apprentissage. Effectivement, un modèle en sous apprentissage peut être identifié à partir de la courbe d'entraînement uniquement. Dans ce cas précis, nous observons que la ligne est plate ce qui indique que le modèle n'a pas du tout appris l'ensemble de données d'apprentissage. Il est intéressant de comparer nos résultats avec un travail soumis pour ce challenge. Pour être sûr de sa pertinence, nous nous référons à la [soumission](#) ayant le plus de votes de la communauté Kaggle. Ce travail utilise sept algorithmes de classification que nous utilisons aussi, desquels nous pouvons tirer des comparaisons. Ainsi, il est aisé de remarquer que nos résultats sont très cohérents. En effet, les proportions sont équilibrés entre ce que nous obtenons et ce qui est obtenu sur la soumission Kaggle.

Nos modèles linéaires semblent même avoir mieux performés de notre côté avec par exemple SVC à 96% contre 82%. De plus, nous accordons une mention spéciale au modèle LDA qui se démarque d'un côté comme de l'autre avec des résultats excellents, tandis qu'à l'inverse le modèle QDA se démarque avec des résultats très médiocres et un sous-apprentissage.

5.2.2 Avec analyse en composantes principales

Intéressons nous à présent aux résultats obtenus à partir des composantes principales de l'ACP. La première remarque flagrante est qu'il n'y a eu aucune détérioration significative des résultats des modèles comparés à ceux obtenues sur les données initiales. En effet, tous les modèles jugés très performants auparavant le sont de nouveau. Par contre, comme souhaité, on observe une amélioration de certains modèles. C'est le cas notamment de l'algorithme K-NN qui était déjà jugé performant, mais qui devient excellent avec ACP, cela est sans aucun doute lié aux calculs de distances qui sont simplifiés.

Passant par exemple de 0.81 à 0.94 d'accuracy de train et de 0.87 à 0.97 d'accuracy de test. De même l'algorithme d'arbre de décision est amélioré d'environ 10% sur l'accuracy de train et sur toutes les métriques de test. Random forest fonctionnant à partir d'arbres de décisions déclare également une légère amélioration globale d'en moyenne 4%. Plus encore, nous pouvons relever que le modèle du quadratic discriminant analysis qui procurait des résultats médiocres, sur les données de tests, sans ACP est devenu viable sur les données de test avec des scores aux alentours de 60% sur toutes les métriques. Cependant, l'amélioration reste mesurée car l'accuracy d'entraînement reste très médiocre et nous décelons toujours un sous-apprentissage sur la "learning curve".

Ainsi, on remarque une amélioration des performances de certains modèles suite à l'ACP, mais il est également important de vérifier la rapidité d'exécution résultante. Pour rappel, nous avons utilisé grid search pour tester trois valeurs de variance expliquées différentes, ce qui est équivalent à effectuer trois fois les entraînements des modèles, et ce qui a donc pour effet de multiplier par trois le temps d'exécution. Cependant, si nous effectuons une moyenne du temps d'exécution de chaque algorithme, nous obtenons moins de 3 minutes pour l'entraînement de tous les modèles avec ACP contre environ 7 minutes 30 secondes sans ACP.

5.2.3 Avec sélection de caractéristiques

En ce qui concerne la sélection de caractéristiques, au préalable, on remarque que les résultats sont très semblables, en tout point, à ceux obtenues avec les caractéristiques initiales. En effet, on remarque des divergences peu notables de plus ou moins 5% sur les différentes métriques utilisées et cela sur tous les modèles confondus.

Cependant, si nous ne percevons pas d'amélioration significative, cela ne veut pas dire que la sélection de caractéristique est un échec et que nous ne devons pas l'utiliser. Effectivement, on rappelle que parmi les objectifs fixés il y a le fait de simplifier le modèle et de réduire la durée de l'entraînement.

Sur ces derniers points, il semblerait que c’est un succès car le temps d’exécution de l’entraînement de tous les modèles dans les mêmes conditions passe de plus de 7 minutes sur le jeu de données sans sélection de caractéristiques, à, en moyenne, environ 2 minutes, sur le jeu de données avec sélection.

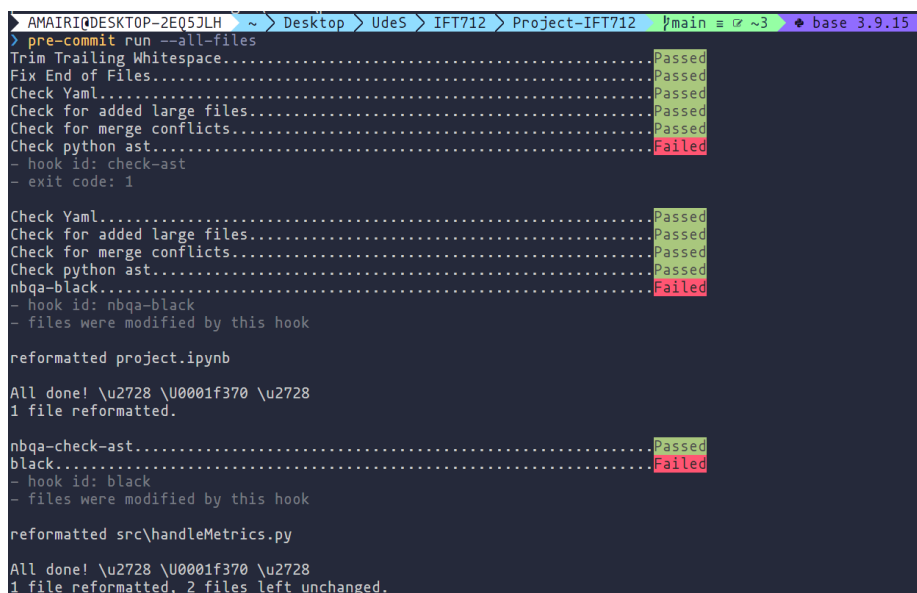
6 Gestion de projet

En tant qu’équipe composée de trois personnes, nous avons cherché à utiliser les outils de gestion de projets mis à notre dispositions de façon à ce qu’ils soient adaptés à la taille du projet et de notre équipe.

6.1 GitHub

Ainsi, afin de partager nos avancées et de sauvegarder notre code source sous forme de version, nous avons utilisé “git” via la plateforme [GitHub](#) notamment. Ce dernier faisant partie de la catégorie des logiciels de gestion de version décentralisés, c’est-à-dire qu’il nous permet de travailler à notre rythme grâce à la création de plusieurs dépôts. Ces dépôts correspondent chacun à des branches que l’on cherche à actualiser au fur et à mesure des mises à jour apportées par nos camarades. Pour cela, nous devons effectuer une revue du code proposé par chacun lors des merges de chaque branche au main. En général, chaque branche correspond à une fonctionnalité à implémenter comme l’interaction d’un modèle au projet.

Pour respecter le standard PEP8, nous utilisons le formateur [Black](#) à l’aide de [pre-commit](#), un git hook. De cette manière, à chaque commit, les fichiers sont vérifiés puis formatés par Black. Bien plus, d’autres hooks ont été installés pour vérifier par exemple les espaces vides, s’il y des conflits de merge entre branche etc...



```
AMAIRI@DESKTOP-2E05JLH ~ > Desktop > UdeS > IFT712 > Project-IFT712 |main = 17 ~3 |base 3.9.15
> pre-commit run --all-files
Trim Trailing Whitespace.....Passed
Fix End of Files.....Passed
Check Yaml.....Passed
Check for added large files.....Passed
Check for merge conflicts.....Passed
Check python ast.....Failed
- hook id: check-ast
- exit code: 1

Check Yaml.....Passed
Check for added large files.....Passed
Check for merge conflicts.....Passed
Check python ast.....Passed
nbqa-black.....Failed
- hook id: nbqa-black
- files were modified by this hook

reformatted project.ipynb

All done! \u2728 \U0001f370 \u2728
1 file reformatted.

nbqa-check-ast.....Passed
black.....Failed
- hook id: black
- files were modified by this hook

reformatted src\handleMetrics.py

All done! \u2728 \U0001f370 \u2728
1 file reformatted, 2 files left unchanged.
```

FIGURE 3 – Exécution de pre-commit

6.2 Jira

En complément du versionning apporté par git, nous utilisons le logiciel de gestion de projet “Jira” reposant sur le principe de “ticketing”. Ce principe correspond à la création de tickets correspondant chacun à une tâche particulière, par exemple l’implémentation d’un algorithme de classification. Dans notre cas, les tickets apportent comme information le statut de la tâche (à faire, en cours, fini), la personne assignée à cette tâche et potentiellement le lien existant entre plusieurs tâches. Cela nous permet donc de suivre l’avancée de chacun et du projet en général.

En fonctionnant sur ce principe d’un ticket par tâche, nous avons fait le choix de lier notre projet Jira à notre projet git et de créer une branche par ticket. De cette façon, il n’y a aucun conflit potentiel entre les tickets et les branches de chaque membre du groupe car chaque ticket est assigné à une seule tâche, elle-même assignée à une unique personne. Pour résumer, nous avons donc une branche main, qui contiendra le code final, ainsi qu’une branche, par ticket Jira, que l’on merge dans la branche main lorsque la tâche est finie.

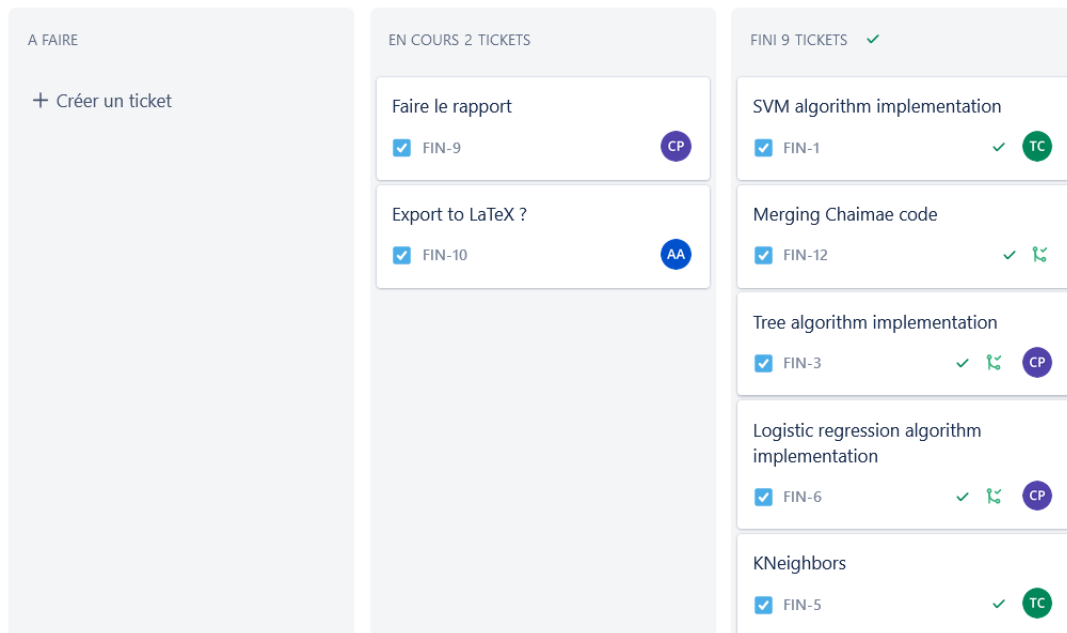


FIGURE 4 – Jira

7 Conclusion

Classifier	Best acc. Train	Feat. trimming ?	Best acc. Test	Feat. trimming ?
AdaBoost	0.90	YES (PCA)	0.93	YES (PCA)
DecisionTree	0.69	YES (PCA)	0.70	YES (PCA)
KNN	0.97	YES (PCA)	0.95	YES (PCA)
LDA	0.97	YES (PCA)	0.97	YES (PCA)
LinearSVC	0.97	YES (PCA)	0.98	YES (PCA)
Logic.Reg.	0.98	NO	0.98	NO
QDA	0.14	YES (PCA)	0.62	YES (PCA)
RF	0.77	YES (PCA)	0.86	YES (PCA)
SGD	0.95	NO	0.96	NO
SVC	0.98	YES (PCA)	0.98	YES (PCA)

TABLE 1 – Tableau récapitulatif

En guise de conclusion, d’après l’évaluation de performance des différents algorithmes que nous avons dû considérer, pour certains modèles on remarque qu’on peut bien booster la performance de justesse mais pour d’autres, cette métrique est bien limitée et il y a beaucoup moins de chances de les améliorer. De ce fait, en améliorant encore les paramètres, il y aura dans certains cas un problème de surajustement des modèles, ce qui entraînera encore plus à l’épuisement des performances.

Alors suite à ce projet, notre application des algorithmes mentionnés précédemment sur la BD proposée sur Kaggle (Leaf classification) a permis d’étudier les performances des modèles suite à certains critères et en variant certains hyperparamètres comme expliqué précédemment. Par ailleurs d’après le tableau récapitulatif, globalement, le meilleur résultat était le SVC One Vs All, grâce à son accuracy (0.98) proche de celle abordée au Kaggle. Ceci n’empêche que plusieurs autres modèles ont donné des résultats très satisfaisants. En effet, ces différents modèles évalués représentent des solutions prometteuses pouvant être implémentées dans différents domaines afin de faciliter les tâches humaines par l’atout de l’apprentissage.

Bibliographie

- [1] Zaid Alissa ALMALIKI. *Do you know how to choose the right machine learning algorithm among 7 different types?* URL : <https://towardsdatascience.com/do-you-know-how-to-choose-the-right-machine-learning-algorithm-among-7-different-types-295d0b0c7f60>.
- [2] Ahmad ANIS. *Easy Guide To Data Preprocessing In Python*. URL : <https://www.kdnuggets.com/2020/07/easy-guide-data-preprocessing-python.html>.
- [3] Michael FUCHS. *Introduction to SGD Classifier*. URL : <https://michael-fuchs-python.netlify.app/2019/11/11/introduction-to-sgd-classifier/>.
- [4] SHASHANK GUPTA. *Micro Average vs Macro average Performance in a Multiclass classification setting*. URL : <https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin>.
- [5] Tom KELDENICH. *Decision Tree Comment l'Utiliser et ses Hyperparamètres*. URL : <https://inside-machinelearning.com/decision-tree/>.
- [6] Ajitesh KUMAR. *MinMaxScaler vs StandardScaler – Python Examples*. URL : <https://vitalflux.com/minmaxscaler-standardscaler-python-examples/>.
- [7] Ajitesh KUMAR. *MinMaxScaler vs StandardScaler – Python Examples*. URL : <https://vitalflux.com/minmaxscaler-standardscaler-python-examples/>.
- [8] SCIKIT-LEARN. *API Reference - Metrics*. URL : <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>.
- [9] SCIKIT-LEARN. *Plotting Learning Curves and Checking Models' Scalability*. URL : https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html.
- [10] SCIKIT-LEARN. *Preprocessing data*. URL : <https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>.
- [11] WIKIPEDIA. *Learning curve (machine learning)*. URL : [https://en.wikipedia.org/wiki/Learning_curve_\(machine_learning\)](https://en.wikipedia.org/wiki/Learning_curve_(machine_learning)).

8 Annexes

8.1 Erreur d'entraînement

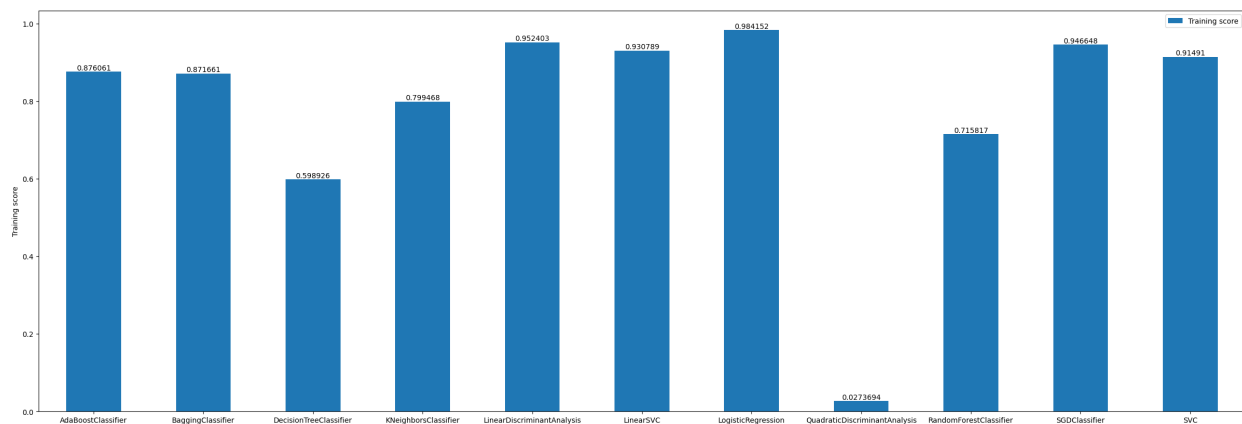


FIGURE 5 – W/o feature trimming

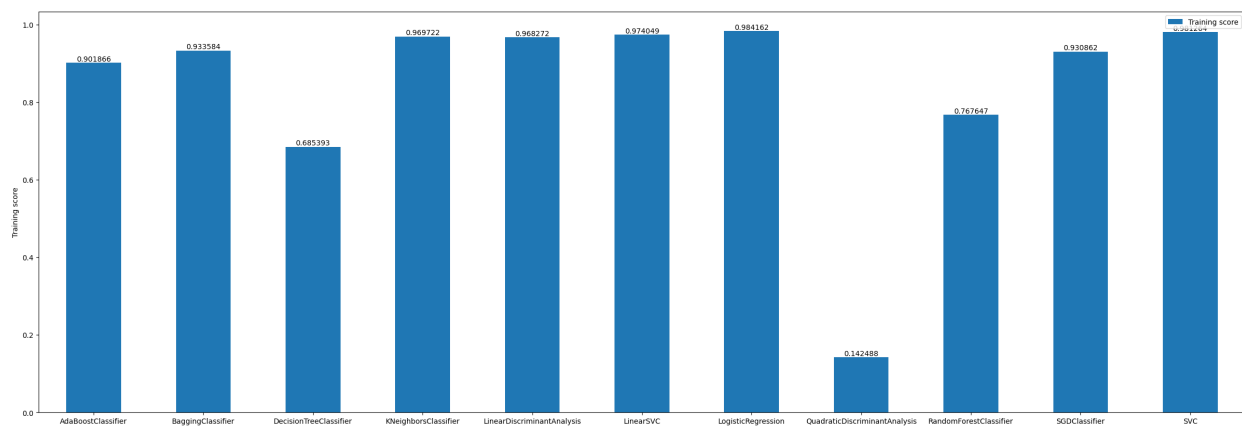


FIGURE 6 – Feature reduction w/ PCA

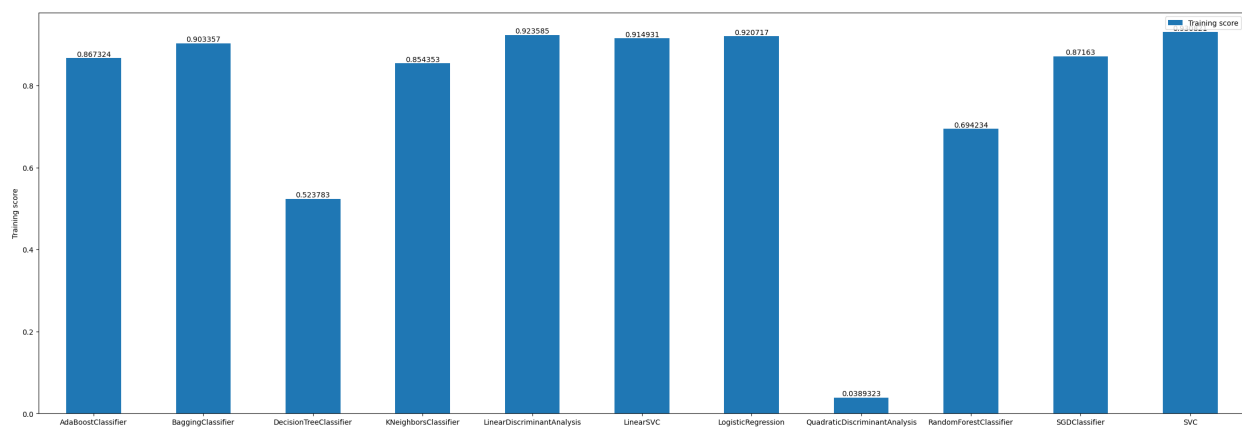


FIGURE 7 – Feature selection w/ Lasso regression

8.2 Métriques de test

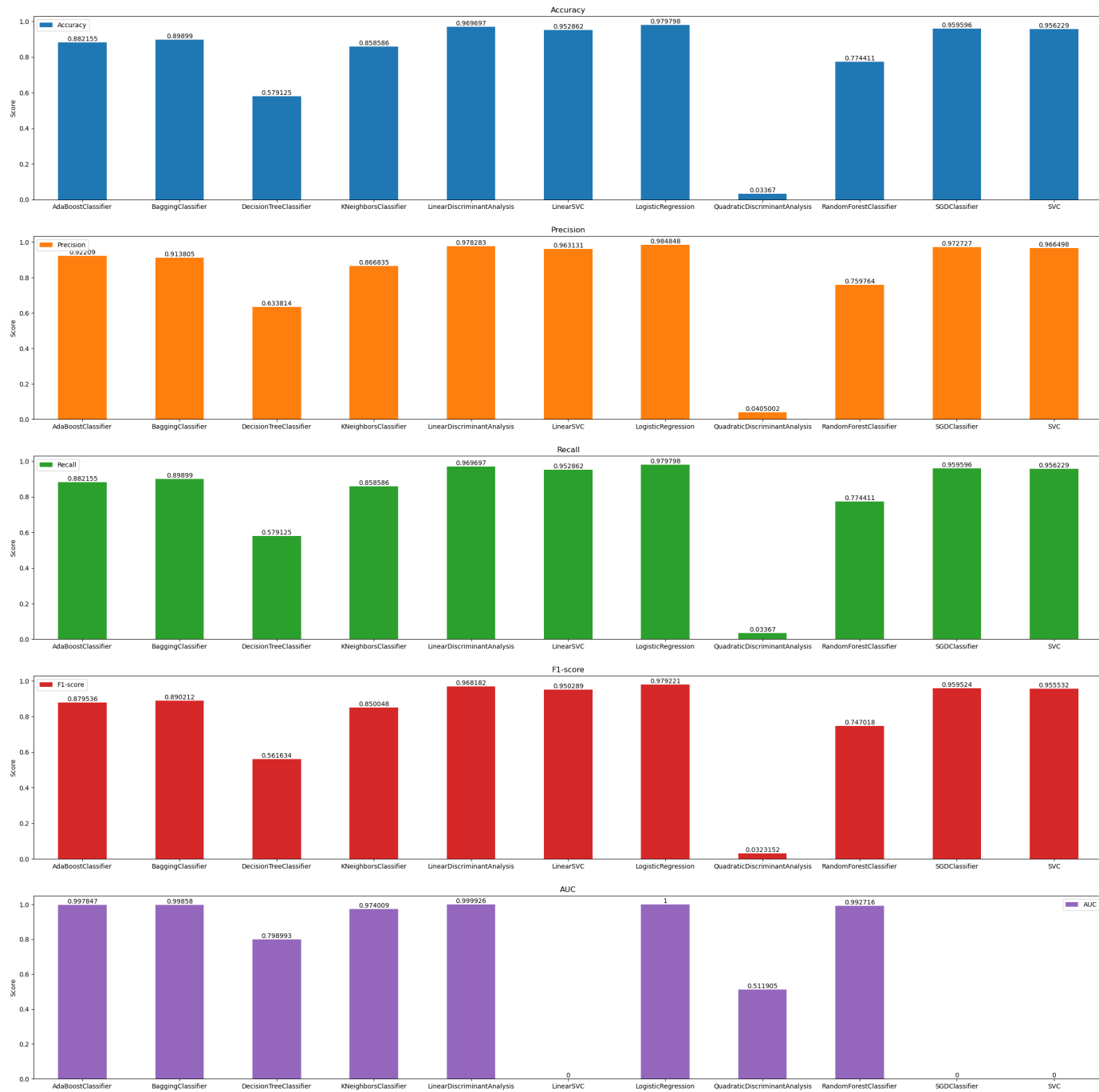


FIGURE 8 – W/o feature trimming

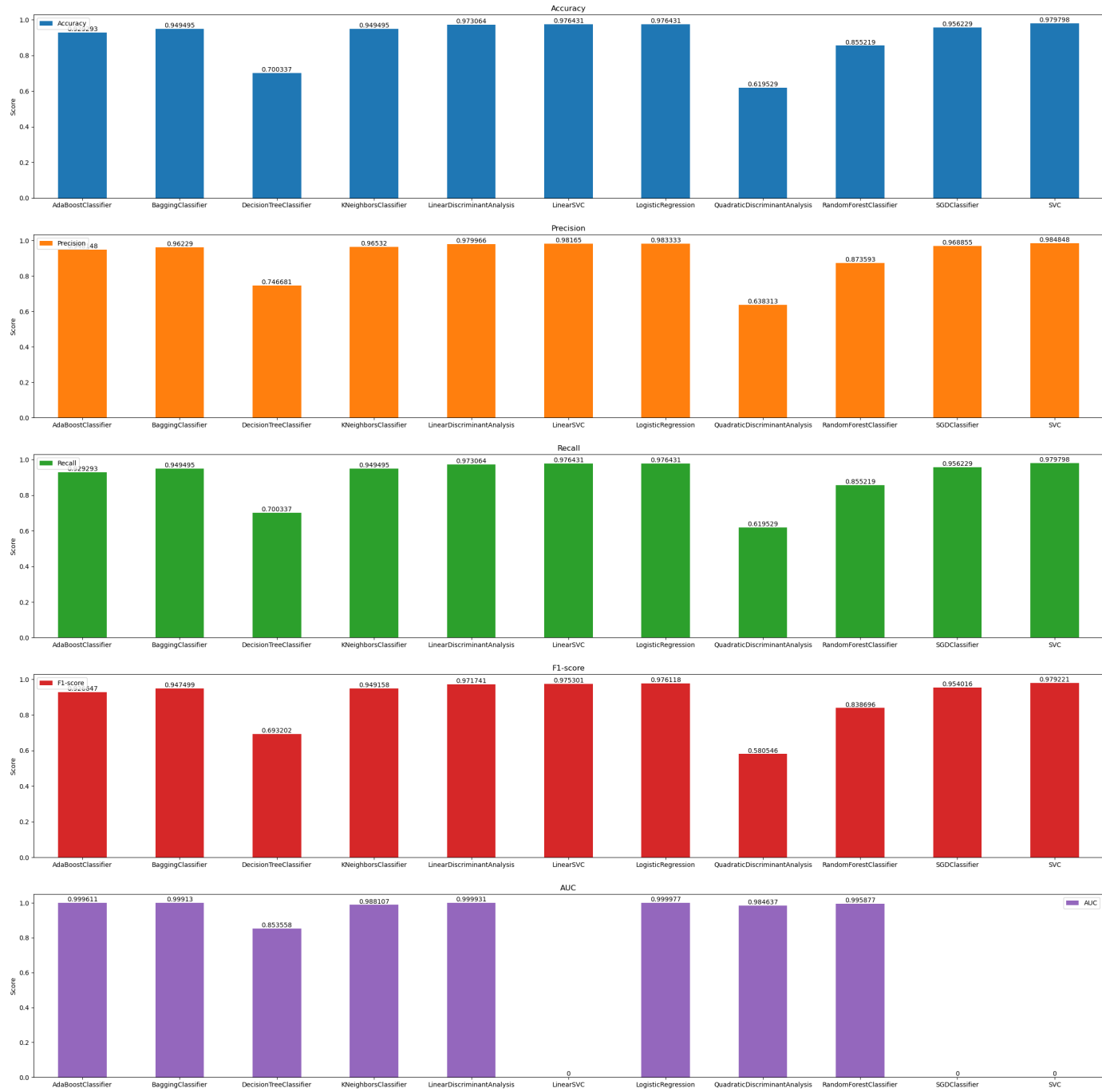


FIGURE 9 – Feature reduction w/ PCA

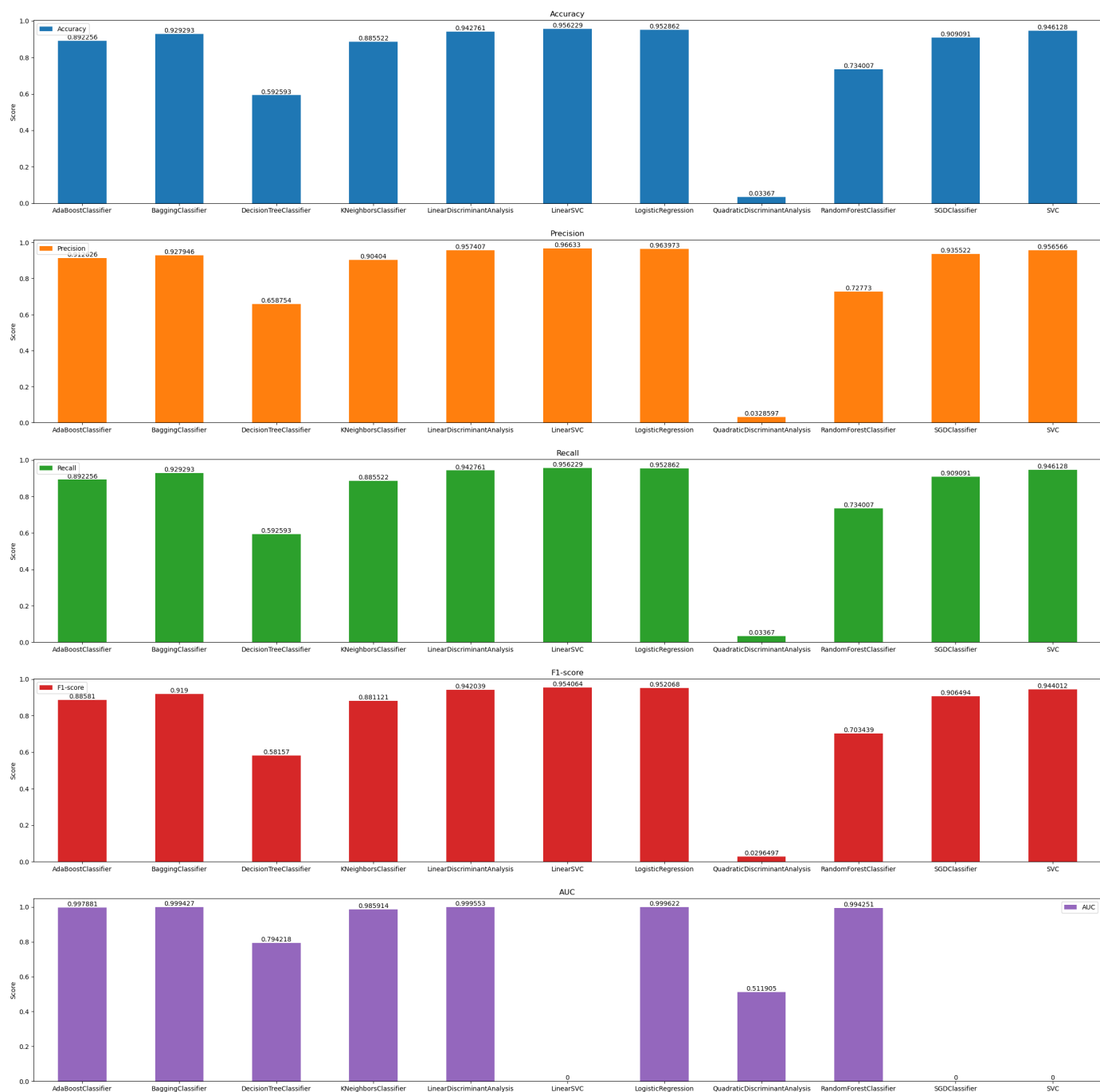


FIGURE 10 – Feature selection w/ Lasso regression

8.3 Courbe d'apprentissage

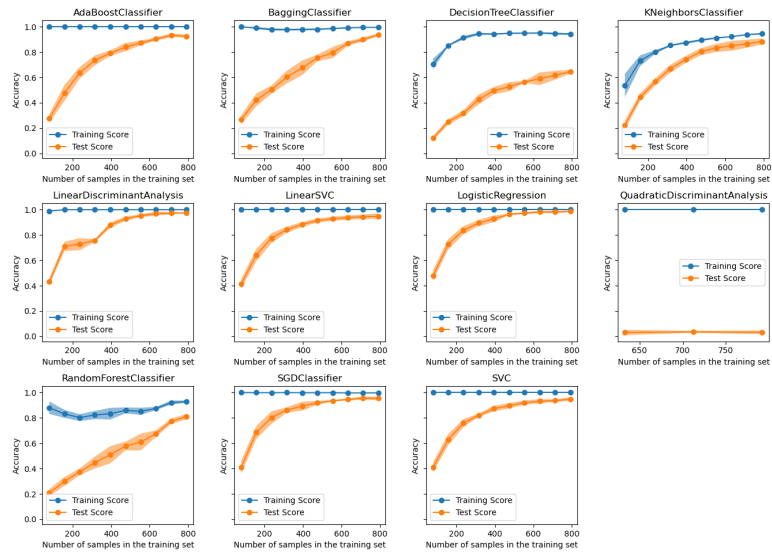
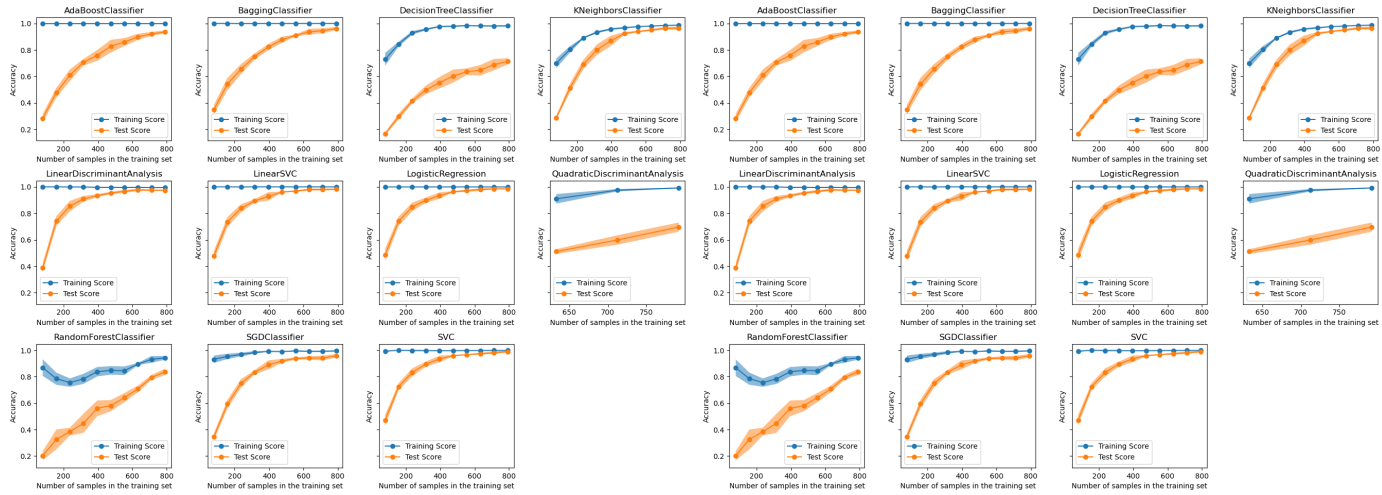


FIGURE 11 – W/o feature trimming



(a) Feature reduction w/ PCA

(b) Feature selection w/ Lasso regression

FIGURE 12 – W/ feature trimming