UNIVERSITÉ DE SHERBROOKE

Département informatique

IFT712 – Techniques d'apprentissage

Comparaison des méthodes de classification sur une base de données "Leaf-Classification"

Professeur:

Martin Vallières

Réalisé par :

BENHAMMOU Nouhayla 22 149 177
DERFOUFI Asmae 22 148 150
L'HASSNAOUI Kaoutar 22 148 702

Mail :

Nouhayla.Benhammou@USherbrooke.ca
Asmae.Derfoufi@USherbrooke.ca
Kaoutar.LHassnaoui@USherbrooke.ca

Année Universitaire 2022/2023

Abstract

In this project we have implemented eight different multi-class classification methods which are: Bayes naïf gaussienne, Support machine vector, Random Forest, Multi-layer Perceptron,Stochastic gradient descent,Linear Discriminant Analysis, AdaBoost, k nearest neighbors method, in training them on a database of leaves from different plant species. We started with data pre-processing and database splitting in 2 sets: training and test. For the implementation of the classifiers we have cross-validate using the search grid of the "scikit-learn" library and for the classification methods, we used the same library. In order to analyze the results, we visualized the different scores using the accuracy metric for each classifier. Finally, we will close the report with a synthetic conclusion.

Keywords: Classification, leaf-classification, Random forest, Mupport machine vector,k nearest neighbors, Stochastic gradient descent, AdaBoost,Bayes naive gaussian,Multi-layer Perceptron,Linear Discriminant Analysis.

Résumé

Dans ce projet nous avons implémenter huit différentes méthodes de classification multi-classes qui sont : Bayes naïf gaussienne, Support machine vector, Random Forest,Perceptron MultiCouches,Stochastic gradient descent,Anal Discriminante Linéaire,Adaboost,KNN (méthode de k plus proches voisins), en les entraînant sur une base de données des feuilles de différentes espèces de plantes. Nous avons commencé par un prétraitement des données et la division de la base de données en 2 ensembles : entrainement et test. Pour l'implémentation des classifieurs nous avons fait la validation croisée en utilisant la grille de recherche de la librairie « scikit-learn » et pour les méthodes de classification nous avons utilisé la même librairie. Pour l'analyse des résultats, nous avons visualisé les différents scores en utilisant la métrique de la justesse pour chaque classifieur. Finalement nous clôturer le rapport par une conclusion synthétique.

Mots clés: Classification, leaf-classification, Random forest, Machine à vecteurs de support,K-plus proches voisins,Stochastic gradient descent, AdaBoost Bayes naive gaussienne, regression ridge,perceptron multi-couches, Analyse Discriminante Linéaire.

Contents

List of Figures

Chapter 1

CONTEXTE GÉNÉRAL

1 Description du projet

Dans le cadre d'un projet de session du cours IFT712 Technique d'apprentissage, nous sommes amené à explorer une base de données des feuilles des plantes, et d'implémenter des modèles de classification basée sur l'apprentissage automatique. afin de les comparer pour trouver le bon modèle qui rendra l'étude des feuilles des plantes plus simple et efficace. Notre classification va se basée sur les algorithmes suivants :

- SVM.
- KNN.
- Random Forest.
- Perceptron MultiCouches.
- AdaBoost.
- Bayes naïve gaussienne.
- Stochastic Descent Gradient.
- Linear Discriminent Analysis

2 Problématique

La problématique majeure que recèle notre projet est de comment bien classifier les feuilles d'arbres de la façon la plus simple et efficace qui soit.

3 Objectifs

Notre objectif est de bien exploiter la base de données fournies et nos acquis de cours afin de tester les différents algorithmes pour pouvoir détecter le celui qui répond au meilleur à notre problématique.

Chapter 2

ANALYSE & CONCEPTION

1 Exploitation de données

Notre projet a pour objectif de classifier un ensemble de feuilles d'arbres,contenant des informations extraites de 1584 images de feuilles. La base de données exploitée a été choisie par notre professeur et est disponible sur Kaggle subdivisée en ensemble d'entrainement et un ensemble de test.Le fichier d'entrainement contient 990 échantillons de 194 caractéristiques, les deux première caractéristiques sont l'identificateur Id et l'espèce : la colonne « species ».Les 192 restants sont tous des caractéristiques sous forme de vecteurs 3x64, et décrivent la forme,la texture des feuilles,et les différentes marges.Ces vecteurs sont extraits depuis la base de données qui contient les différentes images de feuilles.Il faut noter que le nombre total de classes est de 99 classes, et par conséquent chaque classe est illustrée par 10 échantillons.

2 Algorithmes choisis

La sélection des algorithmes de classification que nous allons implémenter a été faite à partir d'une panoplie d'algorithmes connus pour leur usage en classification.Grace à l'exploitation de la bibliothèque Sklearn lors de l'implémentation , nous allons évaluer la justesse de classification de chacun d'eux sur notre base de donnée.Afin de fertiliser davantage nos acquis en intelligence artificielle,nous avons essayer au maximum de découvrir de nouveaux algorithmes non exploités lors de nos séances de cours.

2.1 SVM

Support Vector Machines (SVM) , un algorithme de classification rapide et fiable qui fonctionne très bien avec une quantité limitée de données à analyser.Les machines à vecteurs reposent souvent sur l'utilisation de noyaux qui permettent de séparer les données en les projetant dans un espace vectoriel de plus grande dimension et en

utilisant la technique de maximisation de marge qui permet de garantir une meilleure robustesse face au bruit.SVM est efficace lorsque la dimensionalité des données est assez importante.On note que nous allons utiliser la méthode Sklearn **svm.SVC** qui est appliquable lors de la classification.

2.2 KNN:

L'algorithme des k plus proches voisins ou « k-Nearest Neighbors » est aussi un algorithme d'apprentissage supervisé qui est utilisé pour la classification comme ppour la régression. Pour prédire la classe d'une nouvelle donnée, il cherche ses k plus proches voisins en utilisant une métrique de distance donnée,par défaut il adopte la distance euclidienne.Nous avons décidé de tester la méthode des K-Plus Proches Voisins, implémentés par Sklearn avec **neighbors.KNeighborsClassifier**

2.3 MLP

Le perceptron multi-couches MLP sera notre troisième algorithme qui permet en cas de classification d'établir une liaison mathématique entre nos données d'entrée et les différentes étiquettes de classe.Il est construit de façon à ce qu'il contient plusieurs couches constituées d'un nombre variable de neurones. Dans le perceptron multicouche à rétropropagation, les neurones d'une couche sont reliés à la totalité des neurones des couches adjacentes. Ces liaisons sont soumises à un coefficient altérant l'effet de l'information sur le neurone de destination. Nous allons alors exploiter la méthode **neural_network.MLPClassifier** implémentée par Sklearn, qui adopte la rétropropagation et la fonction d'erreur de l'entropie croisée. De plus, vu que notre problème multi-classe, la fonction d'activation de la couche de sortie sera une Softmax.

2.4 Random Forest

Les forêts aléatoires ou bien Random Forest ,sont des méthodes qui permettent d'obtenir des modèles prédictifs pour la classification et éventuellement pour la régression. Ils se basent sur la mise en œuvre des arbres indépendants qui génèrent plusieurs prédicteurs.Chaque arbre dispose d'une vision parcellaire du problème du fait d'un double tirage aléatoire.Finalement tous ces arbres de décisions indépendants sont assemblés pour faire notre prédiction.La méthode Sklearn **ensemble.RandomForestClassifier** sera la clé pour implémenter cet algorithme.

2.5 Bayes naïve gaussienne

La classification naïve bayésienne est un type de classification bayésienne probabiliste simple basée sur le théorème de Bayes en se basant sur des hypothèses d'indépendance conditionnelle entre chaque attribut sachant la valeur de l'étiquette ou bien le label de la classe.En somme cet algorithme prédit la classe à laquelle apprtient la donnée de test en prenant en considération les probabilités conditionnelles d'appartenance des autres données.La méthode équivalente pour cet algorithme est SKlearn **naive_bayes.GaussianNB**.

2.6 AdaBoost

La classification à travers AdaBoost permet d'adapter d'abord le classificateur à l'ensemble de données d'origine,ensuite crée des copies supplémentaires du classificateur sur le même ensemble de données tel que les poids des instances mal classées sont réajustés de manière à ce que les classificateurs suivants traitent les cas les plus difficiles.
La méthode équivalente pour cet algorithme est **sklearn.ensemble.AdaBoostClassifier**.

2.7 Stochastic descent gradient

La classification par SGD permet l'apprentissage par mini-batch.Le mini-batch essaie de trouver un équilibre entre la qualité de la descente de gradient et la vitesse de SGD.
La méthode équivalente pour cet algorithme est **sklearn.linear_model.SGDClassifier**
.

2.8 Linear discriminanat Analysis

C'est un classificateur avec une limite de décision linéaire, généré en ajustant les densités conditionnelles de classe aux données et en utilisant la règle de Bayes.Le modèle adapte une densité gaussienne à chaque classe, en supposant que toutes les classes partagent la même matrice de covariance.Le modèle ajusté peut également être utilisé pour réduire la dimensionnalité de l'entrée en la projetant dans les directions les plus discriminantes. La méthode exploitée est **sklearn.discriminant_analysis.LinearDiscrimi**

3 Conception UML

La conception de notre projet est l'une des étapes préliminaires les plus imporatntes. Elle nous permet de bien dresser le plan de travail, la modularité du système,et fixer soigneusement une structuration cohérente des fonctionnalités et des données

de notre projet.Nous avons choisi de concevoir un diagramme de classe en UML (Unified Modeling Language).Grâce à ce modèle il nousest possible de représenter simplement un problème, un concept et le simuler.Il faut noter que le diagramme de classes est un diagramme structurel, une partie de la famille qui modélise les relations statiques d'un système dans un état stable. C'est un outil essentiel pour la programmation orientée objet (POO) comme l'est le cas dans notre projet.Ci-dessous la conception en diagramme de classes que nous avons proposée:

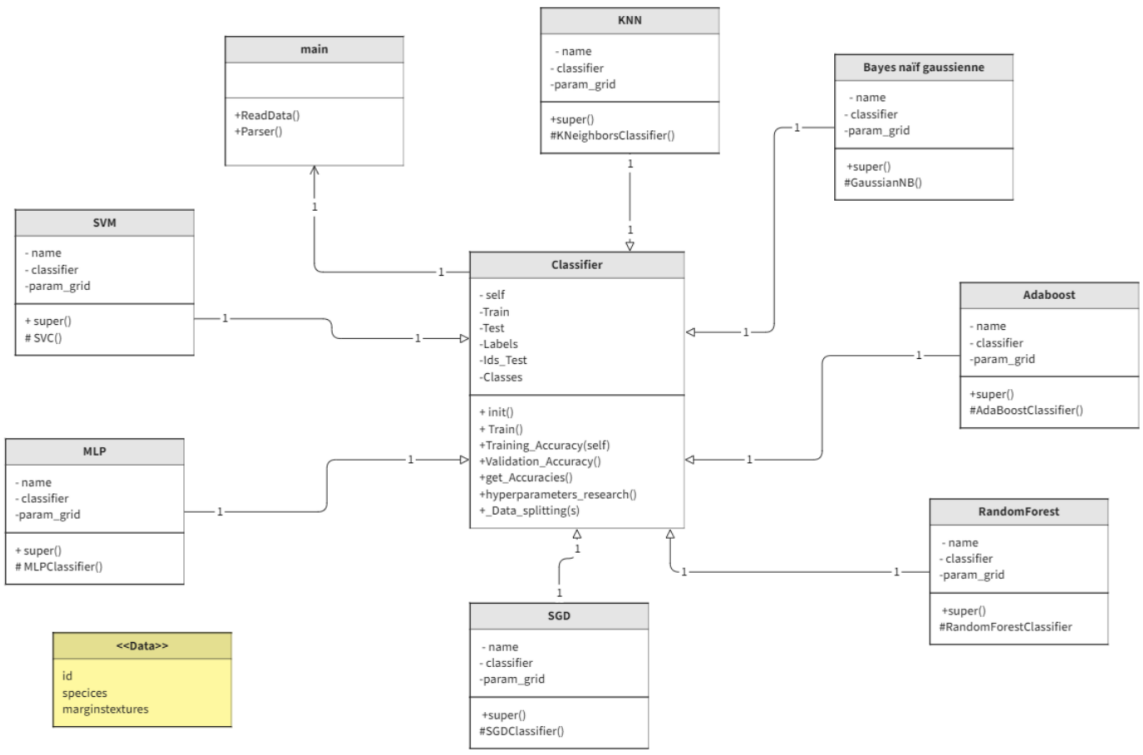


Figure 2.1: Diagramme de classe

4 Gestion de projet

4.1 Trello

La bonne gestion d'un projet est la clé pour sa réussite.Pour garantir cela, nous avons exploiter Trello qui est un outil de gestion de projet en ligne.Nous avons commencer par fixer les taches à faire puis les répartir équitablement entre les membres du groupe.Ensuite nous les avons subdivisés suivant quatre semaines et nous avons fixer des dates pour la réalisation de chaque semaine.Une marge de flexibilité était admise en cas de difficulté affrontée.Il faut noté que nous nous réunissons deux fois

par semaine pour voir nos avancements et résoudre les éventuels blocages.Ci-joint le lien vers le dashboard Trello de notre projet:
`https://trello.com/b/AsvTyJd9/projet-techniques-dapprentissage`

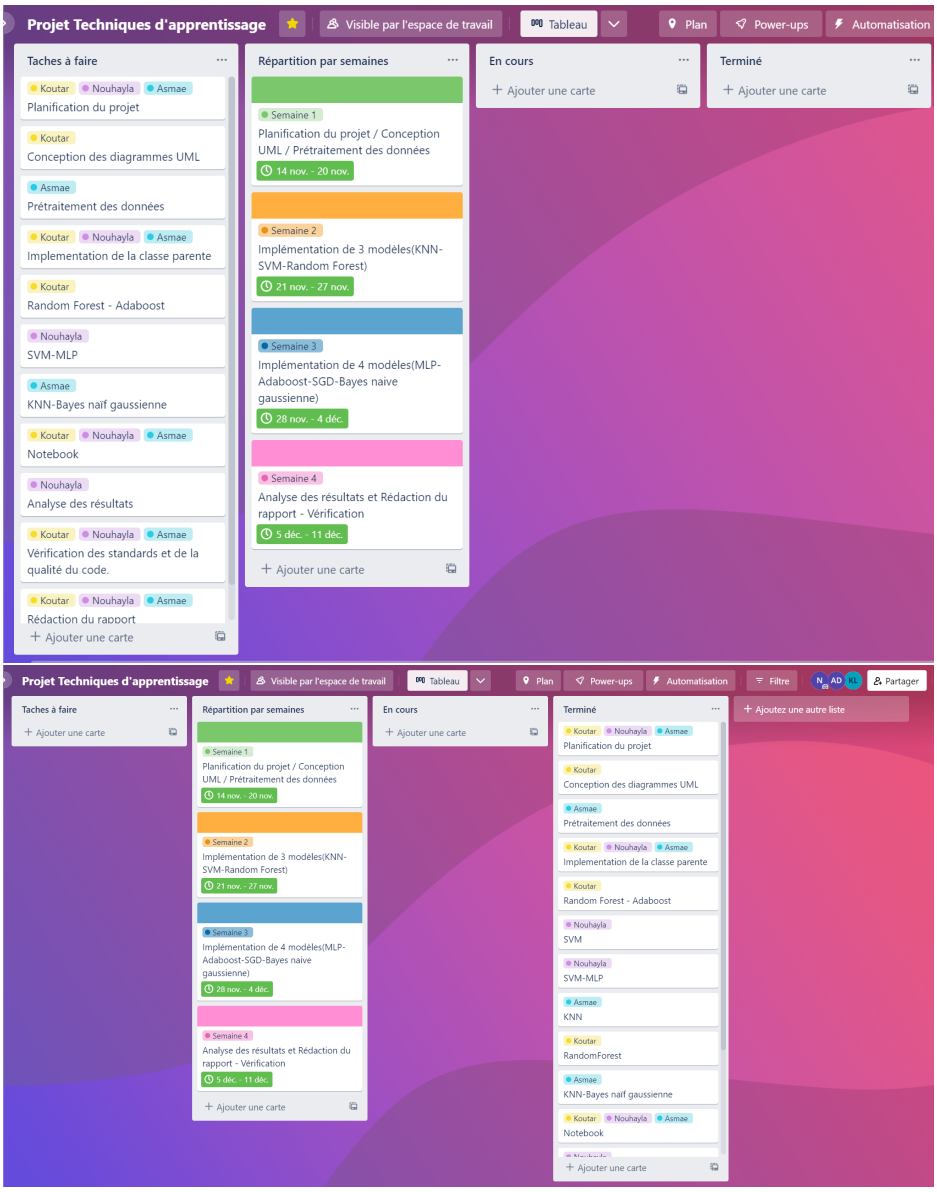


Figure 2.2: Trello

4.2 Diagramme de Gantt

Toujours dans la gestion de projet, nous avons effectuer un diagramme de Gantt pour pouvoir tracer la planification détaillée du projet et aussi de visualiser dans le temps les diverses tâches composant un projet.
Il noter que le classifieur LDA 'Linear discriminent analysis' n'a pas été programmé,mais vu que nous avons fini le travail avant le temps plani-

fié,nous avons alors décidé d'exploiter le temps qui reste dans l'implémentation de ce nouveau modèle pour tester aussi ces résultats.Vous n'allez donc pas trouver ce classifieur ni sur Trello ni sur notre Diagramme de Gantt ni sur notre diagramme de classe.

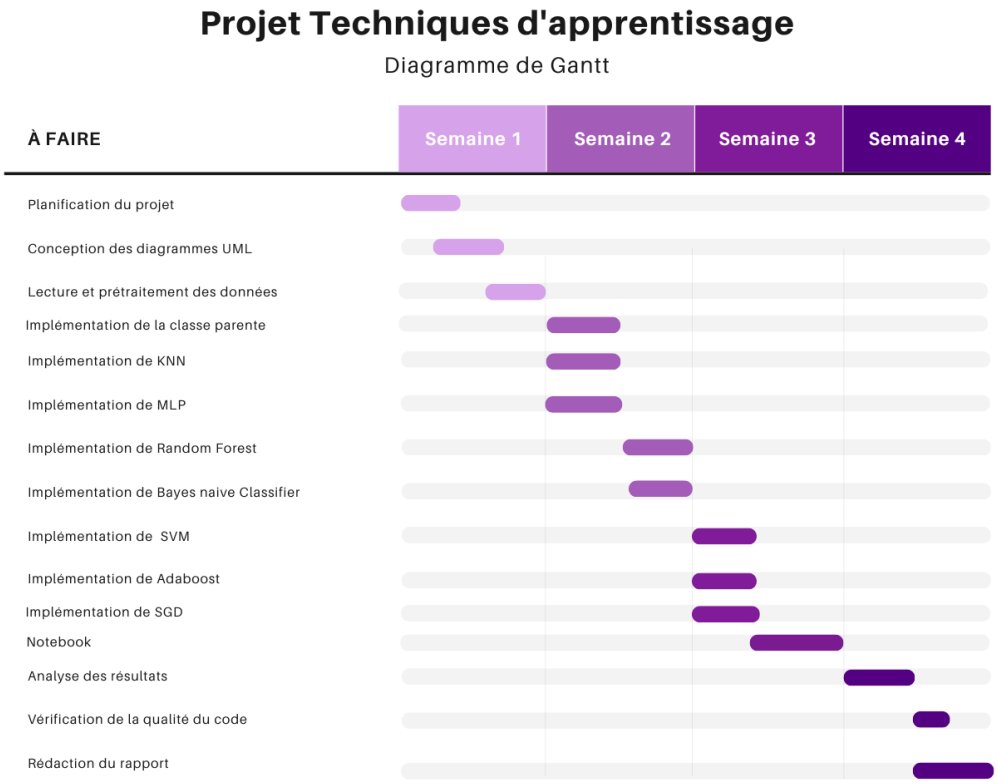


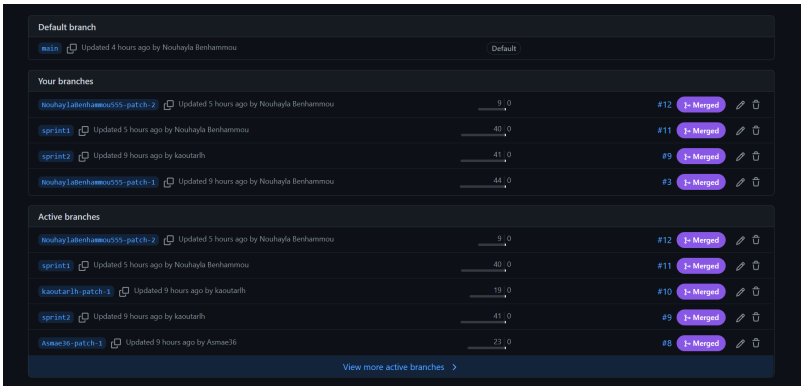
Figure 2.3: Diagramme de Gantt

4.3 Github

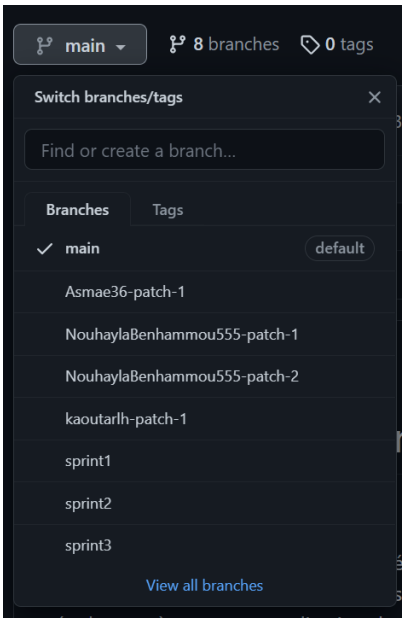
Afin d'avoir une vue globale de l'avancement du groupe, nous avons opté pour Github qui nous a énormément facilité la tache de coordination.Par le biais des push et pull request nous avions l'oeil sur tout changement du code, surtout au niveau des variables, les nom des fonctions,et bien d'autres détails.Nous étions capable de s'entraider lors des blocages vu que nous avions tous une idée des implémentations des autres membres.Ci-joint le lien vers notre projet:
`https://github.com/Asmae36/Technique_dapprentissage`

Figure 2.4: Github

Il faut noter que nous avons veiller à créer des branches pour éviter de pousser du code dans le master sans merge.En l'occurence,nous avons essayer de pousser chaque code lorsqu'il est finalisé afin d'éviter les changements fréquents sur le meme fichier et ceci pour faciliter la tache de révision via l'historique.Par conséquent, nous n'avions poussé les différnents fragements de code que lorsque nous étions sure qu'ils étaient correctes.



(a) Opération du Merge.



(b) Branches utilisées

Figure 2.5: Opérations Github

Chapter 3

REALISATION

1 Démarche scientifique

Ce projet s'inscrit dans la programmation orientée objet(POO).C'est la raison por laquelle il faut exploiter les notions de classes,d'héritages, d'instances, de méthodes et d'attributs.Nous avon donc choisi de construire une classe parente dans laquelle nous allons implémenter des fonctions d'entrainement de données,de splitting,de calcul de justesse respectivement d'entrainement et de validation, et bien évidement une fonstion de recherche de paramètres.Ensuite nous allons implémenter nos modèles de classification chacun indépendemment des autres.Chaque modèle va hériter de la classe parente ce qui va nous faciliter l'implémentation.Un fichier main est obligatoire pour lancer le programme et pour trouver le meilleur classifieur pour notre base de données.Nous vous invitons à revoir notre diagramme de classe présenté précedemment.

2 Gestion des données

Avant de commencer l'implémentation, il faut manipuler les données, les nettoyer,les réorganiser sous la forme avec laquelle on souhaite travailler.Le fichier main qui permet de lire les données depuis le fichier « train.csv ».La bibliothèque pandas a été exploitée pour lire les données sous forme de datafraems.Par la suite avons élminerla colonne des identificateurs et des espèces des palntes moyennant la méthode drop().Une étape sui suit c'est de convertir en valeurs numériques la colonne des espèces.Les labels sont extraits grace à la fonction transform() qui est utilisée pour appeler la fonction sur self produisant une série avec des valeurs transformées et qui a la même longueur d'axe que self.En ce qui concerne les classes nous pouvons les extraire facilement en utilisant classes np.array(data.classes_).On note que les données sont subdivisés en 2 ensembles : données d'entrainement et données de test.

3 Implémentation

3.1 Entrainement

Premièrement on divise les données en 10 parties en conservant le pourcentage d'échan- tillons pour chaque classe (stratification) pour effectuer la validation croisée. Ensuite on a utilisé la classe GridSearchCV qui implémente la fonction d'entraînement pour chaque classifieur, avec plusieurs valeurs des hyperparamètres, pour qu'on puisse récupérer les valeurs optimales des hyperparamètres en utilisant la recherche par grille à travers la validation croisée sur la liste des paramètres. Puis on entraîne chaque modèle et on calcule sa justesse, et comme résultat on retourne le meilleur classifieur avec les meilleurs hyperparamètres, ainsi que les justesses d'entraînement et de validation.

3.2 Recherche des hyperparamètres

Dans l'apprentissage automatique, un hyperparamètre est un paramètre dont la valeur est utilisée pour contrôler le processus d'apprentissage.Pour notre projet nous avons hoisi d'utiliser une méthode de validation croisée afin d'identifier les hyper-paramètres conduisant à la meilleure performance possible des modèles de classification.

Nous avons fait usage de la méthode **sklearn.model_selection.GridSearchCV**. Elle cherche exhaustivement des valeurs de paramètres spécifiées pour un estimateur. GridSearchCV implémente une méthode "fit" et une méthode "score".Les paramètres de l'estimateur utilisé pour appliquer ces méthodes sont optimisés par une recherche de grille à validation croisée sur une grille de paramètres.

Dans ce qui suit, nous allons voir les hyperparamètres choisis pour chacun de nos modèles.Il faut noter que la selection de valeurs d'hyperparamètres présentée a été fruit de plusieurs test jusqu'à trouver la meilleur combinanaison pour chaque modèle.

- **SVM:**

  - C : C'est le paramètre de régularisation. La force de la régularisation est inversement proportionnelle à cette constante. C doit obligatoirement etre positive.
  - gamma : C'est la valeur du coefficient passée au noyau que nous avons choisi d'exploiter.

- kernel : Spécifie le type de noyau à utiliser dans l'algorithme. Si aucun n'est donné, 'rbf' sera utilisé par défaut. Si un appelable est donné, il est utilisé pour pré-calculer la matrice du noyau à partir des matrices de données.Nous avons choisi les noyaux linear, poly, rbf, sigmoid.

- **KNN:**

  - n_neighbors: Identifie le nombre de voisins utilisé.Nous l'avons fixé à 5 voisins.
  - weights:C'est une fonction de pondération utilisée dans la prédiction. Ces valeurs que nous avons implémenter sont:
    1. 'uniforme' : poids uniformes. Tous les points de chaque voisinage sont pondérés de manière égale.
    2. 'distance': pondère les points par l'inverse de leur distance. dans ce cas, les voisins les plus proches d'un point de requête auront une plus grande influence que les voisins qui sont plus éloignés.
  - leaf_size:Taille des feuilles transmise à BallTree ou KDTree. Cela peut affecter la vitesse de construction et de requête, ainsi que la mémoire requise pour stocker l'arborescence.Nous lui avons attribué les valeurs suivantes: [10, 20, 30, 40, 50]
  - algorithm:Choix de l'algorithme utilisé pour déterminer les plus proches voisins(ball_tree, kd_tree, brute, auto),brute, kd_tree).
  - p:Paramètre de puissance pour la métrique de Minkowski. Lorsque p = 1, cela équivaut à utiliser manhattan_distance (l1) et euclidienne_distance (l2) pour p = 2.

- **Random Forest:**

  - n_estimators: c'est le nombre d'arbres dans la forêt,nous lui avons attribué les valeurs suivantes:[250, 300, 350].
  - max_depth : Correspond à la profondeur maximale de l'arbre fixée à [15,20,35,30].

- **Perceptron MultiCouches:**

  - hidden_layer_sizes: C'est la taille des couches cachées.Donnée par ces valeurs: [(50,), (60,), (70,),(80,), (90,), (100,)]
  - activation : Correspond à la couche d'activation utilisée dans la/les couches masquées:'relu', 'logistic','tanh'.

- learning_rate_init : Le taux d'apprentissage initial utilisé. Il contrôle la taille du pas dans la mise à jour des poids:[0.01, 0.001, 0.0001,0.00001]
- solver : Le solveur pour l'optimisation du poids.Nous avons choisi les deux solveurs 'adam' et 'sgd'.

• **Bayes naïf gaussienne:**

- var_smoothing : Portion de la plus grande variance de toutes les caractéristiques qui est ajoutée aux variances pour la stabilité du calcul, fixée à [0.01,0.001, 0.0001,0.00001,0.02, 0.002, 0.0002,0.00002].

• **AdaBoost:**

- n_estimators : Le nombre maximal d'estimateurs auquel l'amplification est terminée. En cas d'ajustement parfait, la procédure d'apprentissage est arrêtée prématurément.Donné à la plage suivante [40,50,60,70].
- learning_rate: Correspond à la pondération appliquée à chaque classifieur à chaque itération de boosting.Si le taux d'apprentissage est élevé alors a contribution de chaque classifieur augmente.Nous l'avons fixé aux valeurs suivantes [1,0.1,0.01,0.001,0.0001].

• **Stochastic Gradient Descent:**

- loss: Correspond à la fonction de perte à utiliser.Le programme a à choisir entre les fonctions suivantes:['hinge','perceptron','huber'].
- penalty: Correspond à la pondération appliquée à chaque classifieur à chaque itération de boosting.Si le taux d'apprentissage est élevé alors a contribution de chaque classifieur augmente.Nous l'avons fixé aux valeurs suivantes [1,0.1,0.01,0.001,0.0001].

• **Linear Discriminant Analysis:**

- solver: Correspond au solveur à utiliser, ces valeurs possibles sont: 1. 'svd' : décomposition en valeurs singulières (par défaut). Ne calcule pas la matrice de covariance, ce solveur est donc recommandé pour les données avec un grand nombre de caractéristiques.

  2. 'lsqr' : solution des moindres carrés. Peut être combiné avec un estimateur de rétrécissement ou de covariance personnalisé.

  3. 'eigen' : décomposition des valeurs propres. Peut être combiné avec un estimateur de rétrécissement ou de covariance personnalisé.

– tol: Seuil absolu pour qu'une valeur singulière de X soit considérée comme significative, utilisé pour estimer le rang de X.Nous lui avons attribué les valeurs suivantes:[0.0001,0.001,0.01,0.1].

4 Résultats & Expérimentations

4.1 Justesses des modèles

L'implémentation des modèles nous a permis de tester plusieurs combinaisons des hyperparamètres et nous avons obtenu les résultats suivans pour les différents classifieurs implémentés.De plus nous avons pu extraire les meilleurs selection d'hyperparamètres pour chacun des modèles.Les valeurs sont en pourcentages.

Classifier	Training Accuracy	Validation Accuracy	Testing Accuracy
SVM	100	95.45	93.44
KNN	100	95.45	94.82
Random Forest	100	95.96	98.61
Bayes naïve gaussian	100	94.44	94.82
AdaBoost	44.44	35.86	37.63
SGD	78.28	68.69	70.06
MLP	100	95.96	94.57
LDA	100	96.97	97.47

Figure 3.1: Résultats des justesses en pourcentages.

Voici la meilleur selection d'hyperparamètres que nous avons pu trouver:

SVM		
C	gamma	kernel
100	0.0001	linear

KNN				
neighbors	algorithm	Leaf size	weights	p
1	Ball_tree	10	distance	1

MLP			
activation	Hidden_layers	Learning_rate_init	Solver
tanh	90	0.01	adam

RandomForest	
Max_depth	N_estimators
30	350

NaiveBayesGaussienne
Var_smoothing
0.002

SGD	
loss	Penalty
hinge	L1

Adaboost	
Learning_rate	N_estimators
0.01	70

LDA	
solver	Tol
svd	0.1

Figure 3.2: Meilleurs hyper-paramètres

4.2 Analyse des résultats

Nous remarquons que globalement nos algorithmes mènent à une bonne classification.Nous avons obtenus des résultats de justesses qui dépassent majoritairement 90%.Ceci indique que nos classifieurs arrivent à bien manipuler nos données et y trouver la meilleure fonction de classification qui permet au mieux l'attribution des données à leurs classes respectives.

De très bons résultats ont été données par Random Forest Classifier et Linear Discriminent Analysis. La raison pour laquelle le modèle Random Forest fonctionne si bien est le fait qu'il combine relativement arbres de décisions non corrélés,fonctionnant ainsi en tant que comité surpasseront n'importe lequel des modèles constitutifs individuels.Par conséquent,les arbres constituant le comité se protègent mutuellement de leurs erreurs individuelles.
En ce qui concerne le classifieur Linear Disciminent Analysis.Il est considéré comme technique de réduction de dimensionnalité, couramment utilisée dans l'étape de prétraitement dans la classification. Cette opération de réduction de dimentionnalité est la raison pour laquelle nous avons pu réduire l'ensemble de données d'entrée qui est de grande dimension sur un espace de dimension inférieure. L'objectif de ce classifieur est de le faire tout en ayant une séparation décente entre les classes et en réduisant les ressources et les coûts de calcul.

D'autre part, on remarque que le classifier Adaboost mène à un sous-apprentissage. Le modèle de données est incapable de capturer avec précision la relation entre les variables d'entrée et de sortie,générant un taux d'erreur élevé à la fois sur l'ensemble d'apprentissage.Cela s'est produit lorsqu'un modèle est trop simple, ce qui peut être le résultat d'un modèle nécessitant plus de temps d'entrainement, plus de fonctionnalités d'entrée ou moins de régularisation.Dans ce cas AdaBoost n'arrive pas à établir une bonne manipulation des données, ce qui entraîne des erreurs d'entrainement et de mauvaises performances du modèle.Par conséquent, ce modèle ne peut pas bien généraliser à de nouvelles données lors de la phase de test, et ne peut donc pas être exploité pour la tâche de classification de notre base de données.

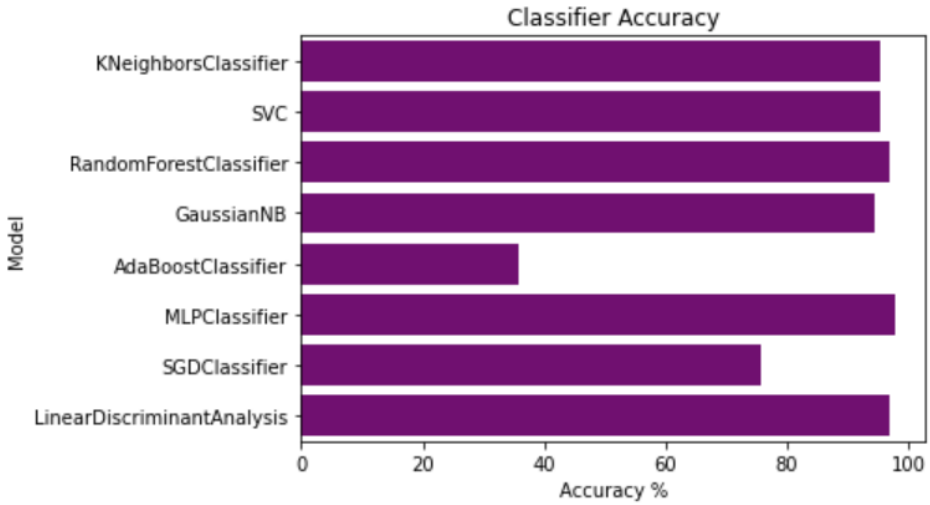


Figure 3.3: Visualisation des justesses de test.

La figure ci-dessus est résultat d'une implémentation sur le Notebook.Nous vous invitons à vous référer à notre Notebook remis avec le code POO.

Chapter 4

CONCLUSION

Durant ce projet on a pu arriver à faire une comparaison entre les différents modèles de classification en se basant principalement sur la bibliothèque sklearn ce qui nous a aider à approfondir et enrichir nos connaissances en Machine Learning et surtout pour les méthodes de classifications.

En guise de conclusion, la plupart de nos modèles donnent de bons résultats, avec souvent une justesse de 100% pour l'entraînement et globalement au-delà des 94% pour la validation. Tout de même, on note que les meilleurs performances selon la justesse de validation sont : Random Forest et LDA. En revanche, Adaboost a donné un sous apprentissage au quel nous pouvons remédier en modifiant la régularisation.

Dans une autre perspective, il sera intéressant d'ajouter d'autres méthodes pour exploiter davantage leurs résultats.Il pourrait être envisagée afin d'obtenir des modèles plus performants avec une combinaison de modèles de classification tout en exploitant des stratégies de bagging et boosting.

Bibliography

[1] :https://scikit-learn.org/stable/model_selection.html.

[2] :https://scikit-learn.org/stable/modules/preprocessing.html.

[3] : https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassif

[4] :https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.

[5] :https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC

[6] :https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison

[7] :https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[8] :https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.

[9] :https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.

[10] :https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.Linear

[11] :https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.h

[12] :https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClass

[13] :https://www.pierre-giraud.com/python-apprendre-programmer-cours/introduction-oriente-objet/.