UNIVERSITÉ DE
SHERBROOKE

Faculté de science

IFT712 - Projet de session

Auteur	*Email*	*Id*
AARABA ABDALLAH	aara2601@USherbrooke.ca	19 143 080
RAMI ABDELLAH	rama2104@USherbrooke.ca	19 143 062

Présenté à :
Pierre-Marc Jodoin

12 Décembre 2019

Table des matières

1 Introduction

Le présent rapport présente le travail fait dans le cadre du projet de session d'automne 2019 du cours IFT 712. Ce projet consiste à appliquer l'ensemble des techniques d'apprentissage automatique apprises tout au long du cours.

1.1 Présentation du problème

Dans ce projet on a comme objectif de choisir un ensemble de données étiquetée et réparties en deux ou plusieurs classes, et y appliquer six différents algorithmes de classification dont l'implémentation est fournie par la bibliothèque "Sklearn", puis calculer des métriques d'évaluations en utilisant des données de test pour pouvoir analyser les résultats et faire la comparaison entre les modèles.

1.2 Description des données

L'ensemble des données qu'on a choisi est celui de **"Leaf classification"**, dont on a extrait les données à partir du site web "Kaggle".
Cet ensemble de données est répartie en deux échantillons, le premier est étiqueté, et sert donc pour faire l'apprentissage de nos modèles. Le deuxième est non étiqueté, et va servir pour calculer l'erreur que fait l'algorithme sur des données jamais vu.
Les étiquettes de données sont réparties sur 99 catégories, qui représente 99 espèces de plantes différentes.
L'ensemble des données contiennent en somme 1548 entrée dont 37.5% est de l'ensemble de test. Chacune des 99 étiquettes se répète 16 fois parmi ces données.
Chaque entrée de l'ensemble de données contient une image, qui représente un spécimen de la feuille d'arbre de la catégorie de l'étiquette correspondante. Ainsi qu'une autre partie qui contient des données numériques, qui décrivent les informations concernant la forme, la marge et la texture de la feuille d'arbre, et chacune de ces trois caractéristiques est exprimé par 64 attributs, ce qui donne en totale 192 attributs numériques.

2 Pré-traitement des données

Pour le pré-traitement, on a décidé de créer en sortie quatre ensembles de données différents à partir des données brute, et qui vont servir à ce que chacun des 6 algorithmes, s'entraîne sur chacun des quatre ensembles et produit ainsi quatre modèles. Ces ensembles ont était construit comme suit :

1. **Ensemble des données composé des caractéristique numérique :**
   Dans ce premier ensemble on a pris tout simplement que la partie numérique de chaque entrée de l'ensemble de départ et on lui ait appliqué la normalisation, ce qui donne donc $64 * 3 = 192$ colonnes correspondantes à tout les attributs numériques de chaque données, en plus d'une colonne qui contient le numéro correspondant à l'étiquette de l'entrée pour les données de l'entraînement.
2. **Ensemble des données à base des images :**
   Pour cette ensemble on a générer pour chaque entrée de l'ensemble de départ, un vecteur de longueur $80 * 80 = 6400$ qui construit n'est construit qu'à partir de l'image correspondante à cette entrée.
   Pour ce faire, on commencée redimensionner toutes les images vers la taille $80 * 80$ est ce en utilisant la fonction **"resize"** de la bibliothèque **"cv2"** de Python, puis on a aplatit cette matrice pour obtenir le vecteur souhaité.
3. **Ensemble des données composé des caractéristiques et des images :**
   Dans cette ensemble on n'a fait que concaténer les deux vecteurs obtenus pour chaque entrée dans les deux ensembles précédents, et ainsi pour chaque donnée on obtient un vecteur d'entrée de longueur $192 + 6400 = 6594$ plus 1 colonne qui contient l'étiquette pour les données d'entraînement.
4. **Ensemble construit par réduction de d'intentionnalité :**
   Dans cette ensemble on a appliqué la méthode d'ACP sur les données du premier ensemble, celui composé des caractéristiques, au but de résumer toutes ces caractéristiques en un nombre donnée de dimensions, en espérant que ça va permettre au modèles de mieux séparer les classes.
   On comprend que le nombre de dimensions de sortie de la méthode est un hyper-paramètre, c'est pour

cela qu'on a implémenter pour chaque algorithme de classification une fonction de cross-validation permettant de trouver le meilleur nombre de dimensions des données transformées parmi l'intervalle allant de 2 jusqu'à 192. Pour faire la comparaison entre les modèles dans la fonction de cross-validation, on a utilisés deux méthodes. La première consiste à entraîner le modèle sur 80% des données d'entraînement est calculer l'erreur de validation sur les 20% qui reste et puis prendre le modèle qui minimise cette erreur. La deuxième méthode consiste à entraîner le modèle sur les données d'entraînement en totalité, et faire la comparaison en se basant sur l'erreur d'apprentissage.
On a finit par adopté la deuxième méthode, car c'est celle qui a donnée de meilleur résultats à la fin quand on teste le meilleur modèle produit sur les données de test.

3 Réalisation

3.1 Régression Logistique

Si on pose l'hypothèse disant que les données sont linéairement séparables, alors l'usage d'un modèle comme la régression logistique est raisonnable. Donc, comme premier modèle, on a utilisé la régression logistique pour justement savoir à quel point cette hypothèse est valable. Ainsi, on a utilisé le modèle du module sklearn.linear_model.LogisticRegression en citant en paramètre que l'algorithme d'optimisation du gradient est 'newton-cg' : LogisticRegression(solver='newton-cg', multi_class='multinomial'). Cet algorithme a été choisi vu qu'il est recommandé pour les problèmes de classification multi-classes avec une pénalité de régularisation en L2 (par défaut).

3.2 AdaBoost

Ici, on a utilisé l'algorithme AdaBoost qui fait la combinaison de plusieurs modèles linéaires simples fonctionnant à base de seuils(stumps). Pour l'implémentation de ce modèle, on a utilisé le modèle du sklearn sklearn.ensemble.AdaBoostClassifier comme suit : AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), algorithm="SAMME", n_estimators=n_estimators). Ce modèle utilise l'algorithme "SAMME" qui donne un poids élevé au modèles les plus crédibles parmi les modèles combinés.
Il faut noter que le nombre de modèles à combiner "n_estimators" est trouvé par une recherche par cross-validation dans un intervalle entre 200 et une valeur donnée en paramètre ayant par défaut une valeur de 5000. Le choix de cet intervalle est dû au fait qu'on veut optimiser cette recherche, donc on a tester sur des valeurs inférieurs à 200, et on a réalisé que ça donne de mauvais résultats au niveau de l'erreur d'apprentissage. Et au-delà de 5000, cette valeur ne change plus, ou ça donne des résultats moins bonnes que la valeur n_estimators=5000. Ceci est fait sur chacun des types de données d'entraînements cités en haut. Ainsi, pour le dernier type d'entraînement (données issues de l'ACP), on cherchera deux paramètres : le nombre de composants principales ainsi que le nombre de modèles à combiner.

3.3 Random Forrest

Dans ce modèle, on a choisi d'utiliser une forêt d'arbres de décision, qui fait le bagging de plusieurs modèles de forte capacité afin d'attribuer une classe à chaque donnée par le biais d'un vote majoritaire. Ce modèle a été réalisé à l'aide du module Sklearn.ensemble.RandomForestClassifier du Sklearn comme suit : RandomForestClassifier(n_estimators=n_estimators).
Ici, n_estimators est le paramètre qui décrit combien d'arbres sont combinés dans cette forêt. Ainsi, ce paramètre est recherché à l'aide d'une cross-validation qui teste différents valeurs de ce paramètre dans un intervalle entre 1 et 5000. Encore une fois, cette recherche est faite pour chacun des quartes types d'entraînement. On a utilisé ici le critère d'impureté "gini" qui est spécifié par défaut vu que son calcul est plus rapide que le calcule de l'entropie vu la complexité du calcul de la fonction du logarithme qui est impliqué dans ce dernier critère. En outre, on a laissé par défaut max_depth = None, comme ça chaque nœud est étendu jusqu'à ce que toutes les feuilles sont considérées d'être pures. On a laissé encore une fois par défaut le paramètre "bootstrap" = True, pour que chaque arbre de décision s'entraîne à l'aide d'un ensemble de données généré à chaque fois à l'aide du bootstrapping.

3.4 Machine à vecteurs de support

Un autre modèle intéressant à utiliser, surtout sur les données numériques, est le modèle construit par l'algorithme SVM, consistant à classer les données en maximisant la marge entre la frontière qui sépare les classes et les points de chacune de ces classes les plus proches de cette frontière, qu'on appelles des vecteurs de support. Ceci à pour but de permettre une meilleur généralisation.
L'entraînement avec la méthode SVM, se fait avec différent type de kernel, ce qui constitue une hyper-paramètre consistant a spécifier à l'algorithme quel kernel il faut utiliser. En plus, chacun des kernels a d'autres hyper paramètres en plus qui sont utiles pour ajouter une spécification à la fonction du kernel.
La fonction utilisés pour créer le modèle SVM est "SVC" du sous-package "svm" de sklearn. Cette fonction nous donne la possibilté de choisir parmi quatres fonctions de kernels disponibles, qui sont la fonction linéaire, la fonction "rbf", la fonction polynomial et finalement la fonction sigmoid.
Ainsi, on a décider de créer un modèle pour chacun des ces 4 fonctions et comparer les résultats, et pour les hyper-paramètres qui détermine des variables dans les fonctions des kernels, ceux-ci vont être choisis par cross-validation. C'est hyper-paramètres sont les suivants :

1. **rbf :** $\exp(-\gamma\|x - x'\|^2)$ et les hyper-paramètres sont :
   — γ spécifié par "gamma"
2. **Sigmoid :** $\tanh(\gamma\langle x, x'\rangle + r)$ et les hyper-paramètres sont :
   — γ spécifié par "gamma"
   — r spécifié par "coef0"
3. **polynomial :** $(\gamma\langle x, x'\rangle + r)^d$ et les hyper-paramètres sont :
   — γ spécifié par "gamma"
   — r spécifié par "coef0"
   — d spécifié par "degree"

3.5 Gaussian Naive Bayes

Pour ce modèle, on a utilisé la méthode de naive bayes, où on attribue une classe à une donnée par maximum à posteriori sur les classes : $\hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i \mid y)$ Or, la question qui se pose est comment calcule-t-on la distribution de vraisemblance pour chaque classe. Pour Résoudre ce problème, on a mis l'hypothèse disant que cette distribution est gaussienne pour chaque classe donnée. Cette hypothèse peut se justifier vu que les données sont normalisées ainsi qu'ils sont numériques et continues. Ainsi, la distribution de vraisemblance pour chaque classe s'écrit sous la forme suivante :
$P(x_i \mid y) = \frac{1}{\sqrt{(2\pi)^n|\Sigma_y|}} \exp\left(-\frac{1}{2}(x_i - m_y)^T {\Sigma_y}^{-1}(x_i - m_y)\right)$
La distribution a priori d'une est calculé par la proportion de données de cette classe par rapport à toutes les données. Ceci est implémenté à l'aide de sklearn.naive_bayes.GaussianNB de la bibliothèque Sklearn.

3.6 Réseaux de neurones

Pour ce dernier algorithme, on a créer un modèle de réseau de neurones pour chacun des quatre ensembles décrit dans le pré-traitement des données, ce qui donne donc diffèrents modèles. Le nombre des couches ainsi que le nombre de neurones dans chaque couche va être décidé par une fonction de cross-validation qui teste différentes combinaisons de nombre de couches allant de 1 à 3, avec différente combinaisons de nombre de neurones pour chaque couche allant de 40 à 180 par saut de 10. La comparaison entre modèle dans la fonction de cross validation se fait par la loss de l'entraînement, car dans le cas où on fait l'entraînement en utilisant une partie des données pour validation on obtient des résultats plus mauvais.
Pour la fonction d'activation, on a choisit la fonction "relu", et comme algorithme d'optimisation on a choisit l'algorithme "adam".
Ainsi

4 Résultats

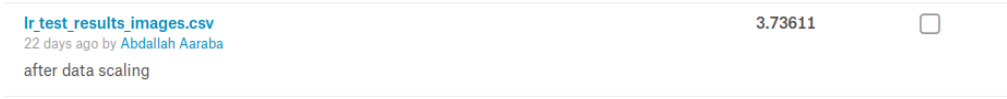
Dans ce qui suit on présente les différents erreurs de tests des six méthodes utilisées, calculés par la cross-entropy chez Kaggle lorsqu'on soumet les prédictions sur les données de test. Ceci est fait en attribuant à chaque donnée de test un vecteur de 99 dimensions, où chaque dimension représente la probabilité que cette donnée appartient à la classe associée à cette dimension. Les classes sont ordonné en ordre alphabétique.

4.1 Régression logistique

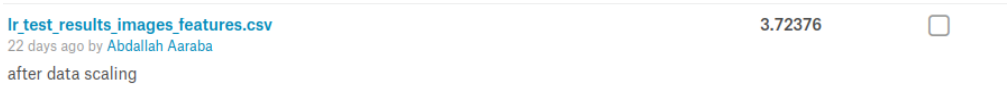
En dessous, les résultats de tests pour chacun des quatre modèles de la méthode de la régression logistique :

1. L'entraînement avec les caractéristiques seules donne :

2. L'entraînement avec les vecteurs des images donne :



3. L'entraînement avec les vecteurs des images concaténés avec les caractéristiques donne :

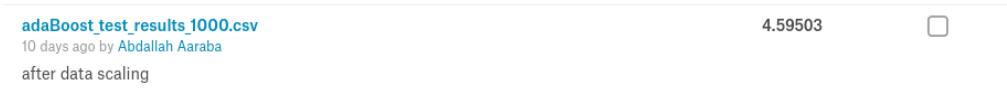


4. L'entraînement en applicant l'ACP sur les caractéristiques avec le nombre de composant = 158 généré par la cross-validation donne :

4.2 AdaBoost

En dessous, les résultats de tests pour chacun des quatre modèles de la méthode d'AdaBoost. Il faut noter que le nombre d'estimateurs utilisé dans l'implémentation des quatre modèles vaut 1000. Ce nombre a été généré par la cross-validation :

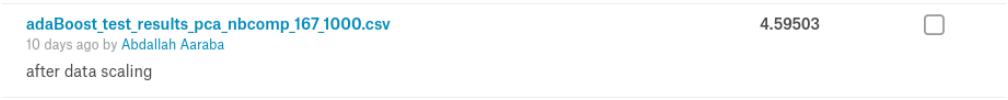
1. L'entraînement avec les caractéristiques seules donne :



2. L'entraînement avec les vecteurs des images donne :

3. L'entraînement avec les vecteurs des images concaténés avec les caractéristiques donne :

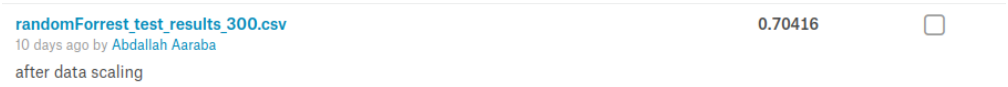
4. L'entraînement en applicant l'ACP sur les caractéristiques avec le nombre de composant = 167 généré par la cross-validation donne :



4.3 Random Forrest

En dessous, les résultats de tests pour chacun des quatre modèles de la méthode de Random forrest. Il faut noter que le nombre d'arbres combinés dans chacun des quatres modèles vaut 300, vu que ça donne de bonnes résultats par rapport aux autres nombre. Encore une fois, Ce nombre a été généré par la cross-validation :

1. L'entraînement avec les caractéristiques seules donne :



2. L'entraînement avec les vecteurs des images donne :

3. L'entraînement avec les vecteurs des images concaténés avec les caractéristiques donne :

4. L'entraînement en applicant l'ACP sur les caractéristiques avec le nombre de composant = 167 généré par la cross-validation donne :

4.4 Gaussian Naive Bayes

En dessous, les résultats de tests pour chacun des quatre modèles de la méthode de Gaussian Naive Bayes :

1. L'entraînement avec les caractéristiques seules donne :

2. L'entraînement avec les vecteurs des images donne :

3. L'entraînement avec les vecteurs des images concaténés avec les caractéristiques donne :

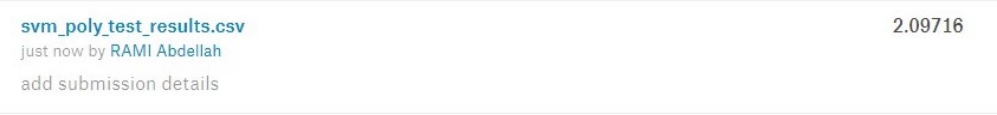
4. L'entraînement en applicant l'ACP sur les caractéristiques avec le nombre de composant = 19 généré par la cross-validation donne :

4.5 Machine à vecteurs de support

En dessous, les résultats de tests pour chacun des quatre modèles de la méthode de Machine à vecteurs de support :

1. L'entraînement avec la fonction de kernel linéaire :

2. L'entraînement avec la fonction de kernel polynomial :



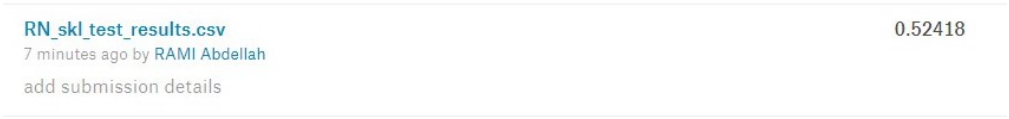
3. L'entraînement avec la fonction de kernel sigmoid :

4. L'entraînement avec la fonction de kernel rbf :

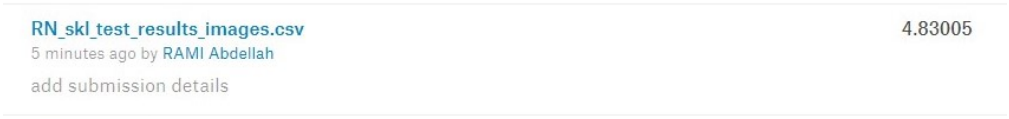
4.6 Réseaux de neurones

En dessous, les résultats de tests pour chacun des quatre modèles de la méthode de Réseaux de neurones :

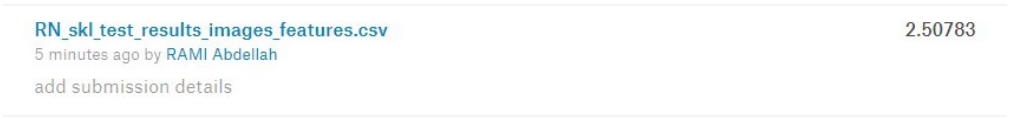
1. L'entraînement avec les caractéristiques seules donne :



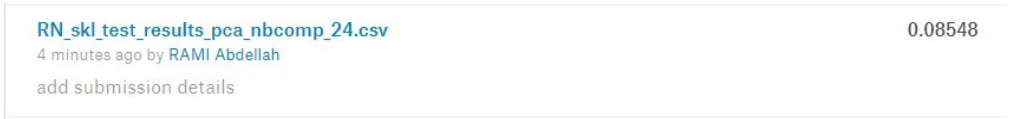
2. L'entraînement avec les vecteurs des images donne :



3. L'entraînement avec les vecteurs des images concaténés avec les caractéristiques donne :



4. L'entraînement en applicant l'ACP sur les caractéristiques avec le nombre de composant = 24 généré par la cross-validation donne :



5 Analyses et comparaisons

1. Entraînement avec les caractéristiques seules :
   Ci-dessous, une figure qui résume les erreurs de test de chacun des six modèles lorsqu'ils sont entraînés par les caractéristiques :

Régression logistique	0.05250
AdaBoost	4.59503
Random Forrest	0.70416
Gaussian Naive Bayes	34.1898
Réseau de neurones	0.52418

Tableau 1: Loss des 5 méthodes appliqués sur les caractéristiques

   Interprétation : On voit d'une part que l'entraînement avec le données de caractéristiques ne marche pas bien pour le modèle Gaussian Naive Bayes. Ainsi, on peut facilement refuter l'hypthèse disant que la distribution de vraisemblance est gaussienne pour ce type de données. D'autre part on voit que la régression logistique a fait l'erreur minimale pour ce type de données.

2. Entraînement avec les images seules :
   Ci-dessous, une figure qui résume les erreurs de test de chacun des six modèles lorsqu'ils sont entraînés par les images :

Régression logistique	3.73611
AdaBoost	4.59503
Random Forrest	1.75993
Gaussian Naive Bayes	19.07191
Réseau de neurones	4.83005

Tableau 2: Loss des 5 méthodes appliqués sur les images

Interprétation : Encore une fois, l'entraînement avec les images ne marche toujours pas bien pour le modèle Gaussian Naive Bayes. Donc, l'hypthèse de normalité des données est toujours refutée pour ce type de données. Or, cette fois, c'est le Random Forrest qui a commis l'erreur minimale.

3. Entraînement avec les images et les features :
   Ci-dessous, une figure qui résume les erreurs de test de chacun des six modèles lorsqu'ils sont entraînés par les images et les caractéristiques concaténés :

Régression logistique	3.72376
AdaBoost	4.59504
Random Forrest	0.79464
Gaussian Naive Bayes	19.01377
Réseau de neurones	2.50783

Tableau 3: Loss des 5 méthodes appliqués sur les images et caractéristiques combinée

Interprétation : On constate que même si on entraine les modèles par les images ainsi que les caractéristiques, ça n'améliore pas beaucoup les résultats. Pour ce type de données, l'hypothèse de normalité de données est toujours refutée pour le modèle GNB. Encore une fois, c'est le Random Forrest qui a donné le bon résultat pour ce type de données.

4. Entraînement avec l'ACP :
   Ci-dessous, une figure qui résume les erreurs de test de chacun des six modèles lorsqu'ils sont entraînés par le résultat de l'application de l'ACP sur les caractéristiques :

Régression logistique	0.03096
AdaBoost	4.59503
Random Forrest	1.19674
Gaussian Naive Bayes	0.49516
Réseau de neurones	0.08548

Tableau 4: Loss des 5 méthodes appliqués sur les caractéristiques de l'ACP

Interprétation : Cette fois-ci, on constate que l'entraînement en applicant l'ACP sur les caractéristiques marche très bien pour le modèle Gaussian Naive Bayes. Ainsi, l'hypthèse de normalité des données peut être accéptée dans ce cas. De nouveau, la régression a donné un bon résultat pour ce type de données, voir le meilleur parmis tous les types des données.

5. Analyse des résultats de la méthode SVM :
   Puisque la méthode SVM n'était pas entraîné avec les 4 ensembles d'entraînement décrits dans le pré traitement, donc on ne peut la comparer avec les autres modèles. Aulieu, on va comparer entre les résultats de la méthodes en utilisant chacun des 4 kernels.

Kernel linéaire	2.39146
Kernel polynomial	2.09718
Kernel sigmoid	1.6796
Kernel rbf Bayes	2.09716

Tableau 5: Les loss des 4 modèles de la méthode SVM

Interprétation : Il est clair que pour les 4 modèles la loss est assez élevée, et donc al méthode SVM n'as pas donnée de bon résultats sur cette ensemble de données.
D'autres part, on voit le kernel "sigmoid" a donner le meilleur résultat, et ceci peut revenir à la complexité de sa fonction. Puis on voit bien que le kernel polynomial a donnée un meilleur résultat que le kernel linéaire, ce qu'on peut expliquer par le fait que la fonction polynomiale est une extension de la fonciton linéaire.

6. Comparaison entre les six méthodes :
Dans cette dernière partie, on va compare les meilleurs résultats donnée par les 4 modèles pour chacune des 6 méthodes.

Régression logistique	0.03096
AdaBoost	4.59503
Random Forrest	1.19674
Gaussian Naive Bayes	0.49516
Réseau de neurones	0.08548
svm	1.6796

Tableau 6: Les loss des 4 modèles de la méthode SVM

Interprétation : Le meilleur résultat à était donné par la méthode régression logistique, puis on voit que la méthode du Naive Bayes et les réseau de neurones ont aussi donnés de bon résultats.

6 Conclusion

Pour conclure, on a appliqué six méthodes de classification, à savoir : la Régression Linéaire, AdaBoost(Boosting), Random Forrest(Bagging), Gaussian Naive Bayes, Support Vector Machines ainsi qu'un Réseau de Neurones. On a entraîné chacun de ces modèles sur quatre types de données différents afin de se trouver avec la meilleur combinaison de modèle plus le type de données donnant le meilleur résultat de test. La plus part des hyperparamètres impliqués dans les modèles ont été recherché à travers une cross-validation. On a fini par une comparaison entre les différentes combinaisons qui a donné comme conclusion que le meilleur modèle est celui fait grâce à la régression logistique.