Rapport de projet

IFT 712 - Techniques d'apprentissages

Auteurs :	
Ikram Mekkid	19 143 008
Sahar Tahir	19 145 088
Aurélien Vauthier	19 126 456

Table des matières

Présentation du projet

Présentation générale

Pour ce projet, nous avons choisi d'utiliser la base de donnée *Leaf-classifier* du site Kaggle. Cette base de données est composée en particulier d'un fichier *train.csv* fournissant 990 feuilles décrites à travers 192 caractéristiques et un dossier d'images binaire. Chaque feuille est ainsi associé à une image.

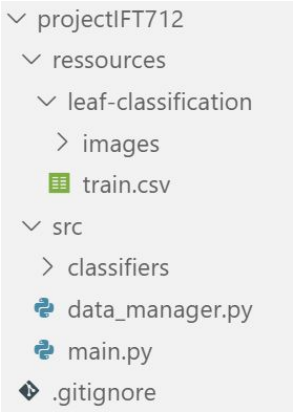
Gestion de projet

Nous avons utilisé Github comme gestionnaire de versions et Trello afin de gérer les tâches à réaliser. Notre board sur Trello est accessible à l'adresse suivante : https://trello.com/invite/b/uoamnEhL/5c0c625bd51c34e516ee80513bf239ac/project-ift712

Démarche scientifique

Design du projet

La structure de notre projet prend la forme suivante :



Il se compose de 2 dossiers principaux et d'un fichier *gitignore* :

- *ressources* : dossier qui contient toutes les ressources (données et images) qu'on utilise au sein de notre projet.
- *src* : le dossier source qui contient le code du projet.

Dans le dossier *src*, on trouve un dossier *classifier* et les deux fichiers *data_manager.py* et *main.py*, dont la description est la suivante:

main.py

Il s'agit du fichier principal de notre projet, il permet d'entraîner et de montrer les statistiques de tous les classifieurs. Lors de la création du *DataManager* il est possible de passer en argument l'attribut booléen *pca* afin de choisir si l'on souhaite appliquer l'ACP sur les données. Les différentes classes de classifieurs pourront chacune prendre un booléen *useImageData* afin de choisir si les données des images doivent être utilisés. Lors de la méthode *train* il est aussi possible de définir l'attribut *verbose* à True pour obtenir régulièrement des informations sur l'état de la cross-validation.

data_manager.py

La classe *DataManager* a pour rôle principal d'extraire les données à partir du fichier *train.csv*. Elle divise les données en données d'entraînement et données de test. La classe *DataManager* extrait aussi les caractéristiques des images de chacun des types de données.

Si l'utilisateur le souhaite, afin de réorganiser les données et de diminuer leur dimensionnalité, on applique une analyse en composantes principales (ACP).

classifiers

Ce dossier contient la classe abstraite Classifier définissant les méthodes *train*, *predict*, *error*, *accuracy*, *loss* et *show_stats*. La méthode *train* permet comme son nom l'indique d'entraîner le classifieur en utilisant les données d'entraînement obtenu à partir du *DataMananger* fourni. *predict* permet de faire la prédiction de données. *error*, *accuracy* et *loss* nous permettent de générer des statistiques afin de comparer nos modèles. En particulier, la précision et la perte sont utilisées dans *show_stats* afin d'afficher les valeurs obtenues pour les données d'entraînement et de test.

La classe *ParametricClassifier* étend la classe *Classifier* dans le but de représenter la classe mère de tous les classifieurs possédant des hyper-paramètres et afin de gérer la cross-validation. Nous y avons donc redéfinie la méthode *train* afin d'exécuter au préalable une cross-validation. Cela est réalisé grâce à la classe *GridSearchCV*, de sklearn, et aux attributs *model* et *param_grid*. Ces attributs ne sont pas définie dans la classe *ParametricClassifier* et doivent donc être définie dans les classe filles afin d'y spécifier respectivement le classifieur de base et la liste des paramètres et de leurs valeurs à tester lors de la cross-validation.

Enfin, nous y retrouvons aussi six classes héritant de *ParametricClassifier*. Ces classes implémentent chacune un classifieur de la librairie sklearn. Nous avons choisi d'utiliser les classifieurs suivants : Logistic Regression, SVM, Random forest, AdaBoost, Neural network, Linear discriminant analysis.

Traitement des données

Le traitement des données se fait principalement en deux grandes étapes:

1. traitement des données issues du fichier 'train.csv'
2. traitement des images et extraction de certaines caractéristiques

Données chiffrées

La méthode privée *_extractBasicData* est responsable de l'extraction des données numériques du fichier csv. Selon la valeur de l'attribut *pca*, l'analyse des composantes principales sera exécutée ou non. Nous avons testé plusieurs valeurs, pour déterminer le pourcentage de variance conservé optimal par rapport au nuage original. Voici les résultats obtenus :

Variance conservé	75%	79%	82%	85%	86%	87%	88%	90%	92%
Nombre de caractéristiques	15	18	20	24	25	26	28	32	37

Nous avons choisis de prendre 0.85 comme pourcentage de variance ce qui diminue le nombre des caractéristiques de 192 à 24 (soit une réduction de 87,5%) en éliminant seulement 15% de la variance.

L'utilisation du *Shuffle Split* fournit des indices de train / test pour diviser les données en ensembles d'entraînement et de test. Ainsi, les données originales sont mélangées puis divisées en données d'entraînement *_X_data_train* et données de test *_X_data_test*. On génère aussi les cibles *_y_train* et *_y_test* qui contiennent les espèces de chaque données encodées par la classe *LabelEncoder.*

Pour pouvoir faire l'extraction des caractéristiques des images, nous avons besoin des ids des données pour cela on les génère aussi dans *_id_img_train, _id_img_test.*

Les méthodes suivantes permettent d'obtenir les différentes données souhaitées :

getBasicTrainData()	données d'entraînement
getBasicTrainData()	données de test
getTrainTargets()	cible des données d'entraînement
getTestTargets()	cible des données de test

Traitement des images

Pour chaque image nous avons choisi d'extraire les données suivantes:

indice	Nom	Description
black_pxl%	pourcentage des pixels noir	On supprime le cadre noir qui entour la feuille et on calcule le pourcentage des pixels noir.
white_pxl%	pourcentages des pixels blancs	On supprime le cadre noir qui entour la feuille et on calcule le pourcentage des pixels blanc.
ratio_W/L	rapport de la largeur de la feuille sur sa longueur	On supprime le cadre noir qui entour la feuille et on calcule le rapport entre la largeur de la feuille sur sa longueur.
nb_peak	nombre de sommets de la feuille	On extrait le contour de la feuille, puis on calcule le nombre de sommets de la feuille.
ellipse_eccentricity	excentricité de l'ellipse	Excentricité de l'ellipse qui correspond le mieux au contour de la feuille.
ellipse_deviation	angle de déviation de l'ellipse	L'angle auquel l'ellipse est dirigé.
line_gradient	gradient de la droite	On approche le contour en ligne droite, puis on calcule le gradient de la droite.
line_y0	image de l'abscisse 0	L'image de l'abscisse 0 selon l'équation de la droite

La méthode *_extractImagesCaracteristics* prend en paramètres une liste des ids des images puis extrait toutes leurs caractéristiques. Pour ce faire, elle utilise plusieurs méthodes dont la description est illustrée dans le tableau suivant :

_first_left_t *_first_right_t* *_first_top_l* *_first_bottom_l*	Ces 4 méthodes calculent les indices des 4 pixels blancs caractérisant la feuille : -le plus à gauche -le plus à droite -le plus haut -le plus bas
_rm_frame	Supprime le cadre noir qui entoure la feuille dans l'image. Cette méthode utilise les 4 méthodes précédentes.
_blackWhite	Calcule le pourcentage des pixels noir et le pourcentage des pixels blancs.
_ratio_width_length	Calcule le rapport largeur / longueur.
_Contour_Features	Cette méthode calcule premièrement le contour de la feuille dans l'image, puis extrait les caractéristiques: *nb_peak*, *ellipse_eccentricity*, *ellipse_deviation*, *line_gradient*, *line_y0*

Pour calculer les caractéristiques des données d'entraînement ainsi que des données de test, on utilise la méthode *_extractImageData* qui applique *_extractImagesCaracteristics* sur *_id_img_train* et *_id_img_test*.

Les méthodes suivantes permettent de récupérer les différentes données :

getImageTrainData()	Caractéristiques des images des données d'entraînement
getImageTestData()	Caractéristiques des images des données de test
getALLTrainData()	Les données d'entraînement combinées avec les caractéristiques de leur image
getALLTestData()	Les données de test combinées avec les caractéristiques de leur image

Les classifieurs

On a utilisé les 6 classifieurs suivants de la librairie sklearn :

Logistic Regression

La régression logistique est un modèle de régression linéaire mais au lieu d'optimiser une fonction linéaire, la régression logistique cherche à optimiser une fonction de perte plus complexe "sigmoid". Pour un meilleur apprentissage nous avons fait une recherche du meilleur hyper-paramètre pour l'attribut du terme de régularisation.

SVM

La classe *SvmClassifier* représente une machine à vecteur de support pouvant utiliser les kernels rbf, sigmoidal et polynomial. Nous avons pu remarquer à travers différentes cross-validations que le kernel le plus optimal pour notre jeu de données était le polynomial. Nous avons aussi joué avec les hyper-paramètres suivants :

- *C* : paramètre de régularisation
- *gamma* : coefficient multiplicateur du noyau
- *coef0* : terme indépendant (i.e. constante) du noyau rbf et polynomial
- *degree* : le degré du noyau polynomial

Random forest

Une forêt d'arbres décisionnels est un classifieur combinant plusieurs arbres de décision. Dans le cadre de la classification de nos données et pour mieux entraîner notre modèle nous avons appliqué la cross-validation sur les paramètres suivants :

- *max_depth* : profondeur maximale d'un arbre de décision
- *n_estimators* : nombre d'arbres de décision

AdaBoost

Afin de mettre en pratique les techniques de boosting, nous avons choisi d'utiliser la classe implémentant l'algorithme AdaBoost-SAMME sur des arbres décisionnels (aussi appelés, *stump decision tree*). Ici, nous avons utilisé la cross-validation sur les paramètres suivants :

- *base_estimator* : la classe du modèle boosté. En particulier, nous avons testé différentes valeurs pour le paramètres *max_depth* des arbres décisionnels.
- *n_estimators* : le nombre maximum de modèles combinés.
- *learning_rate* : le learning rate, réduisant la contributions de chaque modèle.

Neural network

Pour effectuer une classification par réseaux de neurones on a appliqué le *MLPClassifier* de sklearn, en effet le Perceptron MultiCouches (PMC) est un des réseaux de neurones les plus utilisés actuellement en apprentissage supervisé. Dans notre cas, pour

améliorer les résultats de notre modèle, nous avons tenté d'effectuer une cross-validation en optimisant les paramètres suivants :

- hidden_layer_sizes : le nombre de couches cachées
- learning_rate_init : le taux d'apprentissage initial
- activation : la fonction d'activation {'identity', 'logistic', 'tanh', 'relu'}
- solver : {'lbfgs', 'sgd', 'adam'}

Linear discriminant analysis

L'analyse discriminante linéaire est une techniques d'analyse discriminante prédictive généralisant l'analyse discriminante de Fisher. Elle a pour objectif de classifier linéairement des entités à partir de combinaisons linéaires des caractéristiques initiales. Dans ce classifieur, nous pouvons utiliser différents solveur et différentes valeurs pour *shrinkage*. Cependant, le solveur *eigen* nécessite que la matrice des données soit inversible (ce qui n'est pas le cas lorsqu'on utilise les données des images) et le solveur *svd* impose de ne pas avoir de shrinkage.

Analyse de résultats

Logistic Regression

Comme on peut le voir sur le tableau des résultats suivant, et comme on pouvait l'imaginer, un classifieur linéaire peut difficilement classifier nos données qui ne sont assurément pas linéairement séparables.

Hyperparamètres	penalty	accuracy (test)	loss (test)
Données numériques	'l2'	56.0606%	4.1876
Données numériques et images	'l2'	57.5758%	2.9392
Données numériques (ACP) et images	'l2'	51.5152%	3.0418

SVM

Comme nous pouvons le remarquer sur les exemples de résultats obtenus ci-dessous, le noyau polynomial, sur les données numériques seules, est bien plus performant que le noyau 'rbf' sur les autres jeux de données. Nous n'avons cependant pas réussi à utiliser le noyau polynomial sur les deux derniers jeux de données car l'entraînement de ce dernier était trop long voir infini. Malgrés différentes tentatives pour changer les jeux de paramètres de la cross-validation, nous avons été contraint de n'utiliser que les noyaux rbf et sigmoid. Or ces noyaux souffrent de sur apprentissage (en particulier sans l'utilisation de l'ACP) et ce, malgré de larges champs de valeurs testé pour le paramètres de régularisation C.

Hyperparamètres	Kernel	C	gamma	coef0	degree	accuracy (test)	loss (test)
Données numériques	'poly'	0.0001	0.0001	17.5	9	95.9596%	2.3066
Données numériques et images	'rbf'	2e-137	0.01	0	None	17.6768%	4.5963
Données numériques (ACP) et images	'rbf'	743	2.6e-05	0	None	28.2828%	3.2960

Random forest

D'apres les resultats ci-dessous, on remarque alors que les FAD donnent de très bons résultats pour toutes sorte de données, malgrés une diminution de l'efficacité avec l'ACP. Cela confirme qu'une combinaison de modèles (dans ce cas : les arbres de décisions via le

bagging) peut toujours être une bonne idée. On en déduit par conséquent, qu'il s'agit d'un bon classifieur pour notre jeu de données.

Hyperparamètres	max_depth	n_estimators	accuracy (test)	loss (test)
Données numériques	60	99	98.4848%	0.7310
Données numériques et images	70	88	97.9798%	0.7192
Données numériques (ACP) et images	50	89	92.9293%	0.8768

AdaBoost

Nous pouvons voir ci-dessous des exemples de résultats de cross-validation pour l'algorithme AdaBoost. On remarque ici que les meilleurs sont obtenus sans ACP et avec une légère amélioration lorsque l'on dispose des informations sur les images.

Hyperparamètres	base_ estimator	n_estimators	learning_rate	accuracy (test)	loss (test)
Données numériques	max_ depth = 13	88	0.016681	97.9798%	0.1572
Données numériques et images	max_ depth = 13	84	0.027825	98.4848%	0.1946
Données numériques (ACP) et images	max_ depth = 13	86	0.1	92.4242%	0.5111

Neural network

Comme nous pouvons le constater d'après les résultats ci-dessous, le perceptron multi-couche a beaucoup plus de mal avec les données numériques associés aux images. En effet, dûe à de long temps de convergence, nous n'avons pas réussi à effectué de cross-validation sur les hyper-paramètres. Nous avons remarqué que pour un nombre maximum d'itération 10000, le réseau ne converge toujours pas mais nous pouvons atteindre une précision de l'ordre des 75% en test avec toutes les données sans ACP. Nous pouvons donc en conclure que le PCM donne de bons résultats lorsqu'il est entraîné de manière prolongé avec une haute valeur de max_iter. Cependant, pour notre cas d'utilisation, le PCM n'est pas un bon classifieur.

Hyperparamètres	hidden_layer_sizes	learning_rate_init	activation	solver	accuracy (test)	loss (test)
Données numériques	(104,19)	0.0522	identity	adam	87.8788%	0.2983
Données numériques et images	par défaut (100,)	1.0	identity	adam	54.5455%	15.6994
Données numériques (ACP) et images	par défaut (100,)	1.0	identity	adam	30.3030%	2.6166

Linear discriminant analysis

Voici des exemples de cross-validation, comme nous pouvons le remarquer, les meilleurs résultats sont obtenues lorsque nous ne réalisons pas d'ACP. Nous pouvons imaginer que cela s'explique par le fonctionnement du classifieur qui réalise, lors de son entraînement, un travail similaire à l'ACP. Ainsi, cette algorithme serait plus performant avec l'ensemble des caractéristiques plutôt qu'avec des caractéristiques appauvri en termes d'informations sur les feuilles.

Hyperparamètres	solver	shrinkage	accuracy (test)	loss (test)
Données numériques	'lsqr'	3.0888e-06	98.4848%	0.0652
Données numériques et images	'lsqr'	7.1968e-12	97.9798%	0.2298
Données numériques (ACP) et images	'lsqr'	1e-12	94.9495%	0.2990

Conclusion

En conclusion, nous pouvons noter que les meilleurs résultats de classification ont été obtenus par les combinaisons de modèles Random Forest et AdaBoost, ainsi qu'avec l'analyse discriminante linéaire. De plus, contrairement à ce que l'on pourrait penser, l'ACP est rarement bénéfique aux algorithmes de classifications lorsque l'on regarde les performances. Cependant, nous avons remarqué que l'ACP avait tendance à accélérer la vitesse d'entraînement, ce qui peut ainsi parfois contrebalancer la perte de performance.

Bibliographie

- https://www.kaggle.com/c/leaf-classification
- https://scikit-learn.org/stable/
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html
- https://pillow.readthedocs.io/en/stable/reference/Image.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html