L2 MI/ Mini-projet Challenge Areal - Grp Orbiter
Identifiant codalab : orbiter@chalearn.org
Membres de l'équipe : Pinto Florian, Bourbonnot Marc, Tifouche Keissa,
Slimani Adlene, Jules Bourges et Thierry Habinshuti
URL du challenge : https://codalab.lri.fr/competitions/399 (preprocessed)
https://codalab.lri.fr/competitions/379
Repo Github du projet : https://github.com/grpOrbiter?tab=repositories
Vidéo youtube : https://www.youtube.com/watch?v=PhNkvf0F9gM
Diaporama :
https://github.com/grpOrbiter/prepro_orbiter/blob/master/Pr%C3%A9sentation1.pptx
Soumission de code n°8806.

Contexte et description du problème :

De tout temps l'homme a cherché à se situer dans l'espace, à connaître et à comprendre l'univers qui l'entoure. Les dernières avancées technologiques lui ont permit d'envoyer des outils dans l'espaces et ainsi d'obtenir des images satellites de la Terre et de l'Univers.

Une image satellite, ou image satellitaire, est une prise de vue transmise d'un satellite artificiel en orbite. Elles permettent d'obtenir différentes informations comme la surveillance des pays ennemis pour les militaires (fonction première de la création d'un satellite d'observation), prévisions météorologiques (par exemple les satellites Météosat), ou tout simplement pour les recherches sur l'Univers. Certains satellites sont capables d'une précision telle que cela peut devenir un problème, notamment en termes de vie privée ou de secret d'État.

Notre projet s'agit du challenge Areal qui consiste justement à classifier des images prises par satellite en différentes catégories en utilisant les méthodes de l'apprentissage Supervisé. Plus clairement, à déterminer si les images sont des océans, déserts, forêts… Pour cela, nous avons à notre disposition deux jeux de données de 13 classes (700 images par classes). Les images brutes sont en 128*128 pixels, et chaque pixel est en RGB; c'est à dire, 3 canaux. Ce qui nous donne en général, 49152 canaux par image (128*128*3).

Pour évaluer la fiabilité et la performance de notre modèle, nous avons utilisé la métrique qui donne un score entre 0 et 1, ce dernier étant le meilleur score. Ce calcul correspond au nombre de bons résultats sur le nombre total d'exemples.

Les images brutes et pré-traitées se différencient en 2 projets (Deux parties indépendantes) et deux niveaux de complexité différents. Les données brutes sont souvent plus complexes et nécessitant des pré-traitements spécifiques pour les rendre manipulables par les algorithmes, peuvent servir de sources pour un problème de modélisation.

En ce qui concerne notre groupe, nous travaillons sur le premier projet qui met à notre disposition les images prétraitées: de taille 32*32 pixels, donc 3072 canaux. S'il nous reste du temps, nous aimerions aussi réaliser la deuxième partie qui consiste un traitement spécial sur les données brutes.

Nous avons divisé notre travail à fournir en 3 parties, chacune traitée par un binôme. Le premier binôme, s'occupe de la partie preprocessing; c'est à dire la transformation des images en données simples pour être utilisées par le classifier. Pour cela nous avons utilisés le preprocessing PCA, les détails sur cette méthode sont bien présents dans la première partie du projet. Le deuxième groupe, quant à lui s'occupe du "classifier", qui analyse les données prétraitées mises à notre disposition et prédit la classe de chacune d'entre elle selon leurs origines. Pour cela, il va d'abord falloir lui fournir un jeu de données d'exemples

afin qu'il puisse s'entraîner et s'améliorer, d'où le mot **apprentissage**. Dans cette partie nous avons testés plusieurs classifieurs telles que le perceptron multiclasse et SVC et nous avons utilisés gridsearch afin de faire varier les hyperparamètres. On a commencé à tester des hyperparamètres d'une manière aléatoire parce qu'on n'avait aucune idée du comportement de nos données vis-à-vis de nos méthodes de classification. Enfin, le dernier groupe s'occupe de la partie visualisation; c'est-à-dire, de l'affichage graphique afin de déterminer et mieux visualiser les données fournies et les résultats obtenus avec clarté et précision. Pour obtenir des graphes nous avons utilisés tout simplement la bibliothèque matplotlib qui met à notre disposition plusieurs graphiques de visualisation avec les codes qui les fournissent, et nous avons choisi les plus faciles à obtenir avec nos propres données et résultats mais aussi les graphiques qui sont simples à analyser comme la matrice de confusion qui utilise des couleurs évaluées sur une échelle.

Pour réaliser le projet du début jusqu'à la fin, nous avons utilisé des bibliothèques Python mises à notre disposition afin de nous guider sur des choses qu'on ne connaissait pas auparavant et nous aider à avancer plus vite pour obtenir certains codes de méthodes utilises dans certaines parties de notre travail et qu'on ne serait pas forcément capable de faire à notre niveau.

Description des classes :

1) Preprocessing : Jules Bourges et Thierry Habinshuti

Dans un premier lieu, nous avons pensé à réaliser cette partie avant de passer à la classification dans le but de renDre la dimension de nos données plus inférieure. Nous nous sommes aidés de SelectKBest existant dans la bibliothèque « Sklearn » qui nous a proposé plusieurs méthodes, mais notre choix s'est porté sur la méthode Principal component analysis (PCA) (ref.2). Cette méthode consiste à transformer des variables liées entre elles (dites « corrélées » en statistique) en nouvelles variables décorrélées les unes des autres. Ces nouvelles variables sont nommées « composantes principales », ou axes principaux. Elle permet au praticien de réduire le nombre de variables et de rendre l'information moins redondante.

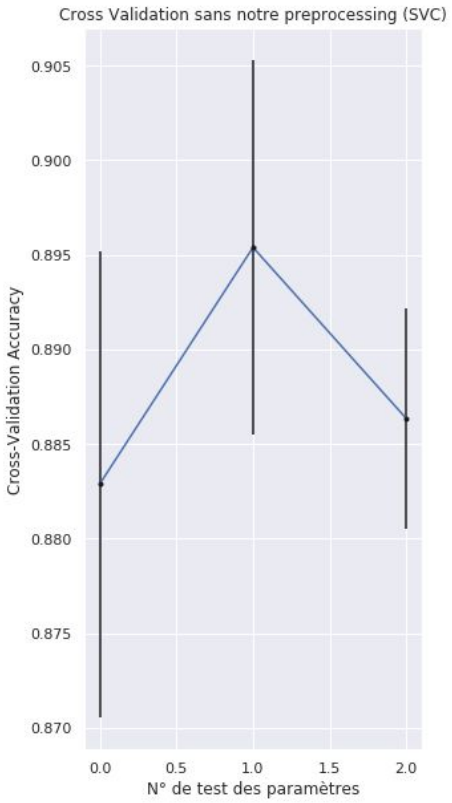
Dans le cas d'une image très simple, les pixels sont représentés dans un plan et considérés comme une variable aléatoire à deux dimensions. La PCA va déterminer les deux axes qui expliquent le mieux la dispersion de l'objet, interprété comme un nuage de points. Elle va aussi les ordonner par inertie expliquée, le second axe étant perpendiculaire au premier.

L'algorithme Principal Component Analysis (PCA) utilise la fonction pca.fit_transform() de sklearn.decomposition qui ajuste le modèle X en lui appliquant une réduction de dimension. Nous avons aussi fait variées les hyperparamètres à l'aide de gridsearch mais nous n'avons pas notés de grande différence dans les résultats. Grâce à l'algorithme du PCA nous espérions avoir un meilleur score lors de la validation mais ce n'est pas le cas. C'est à cause du fait que les images que nous utilisons sont déjà prétraitées ce qui est donc normale d'avoir un résultat plus bas que la moyenne. Nous

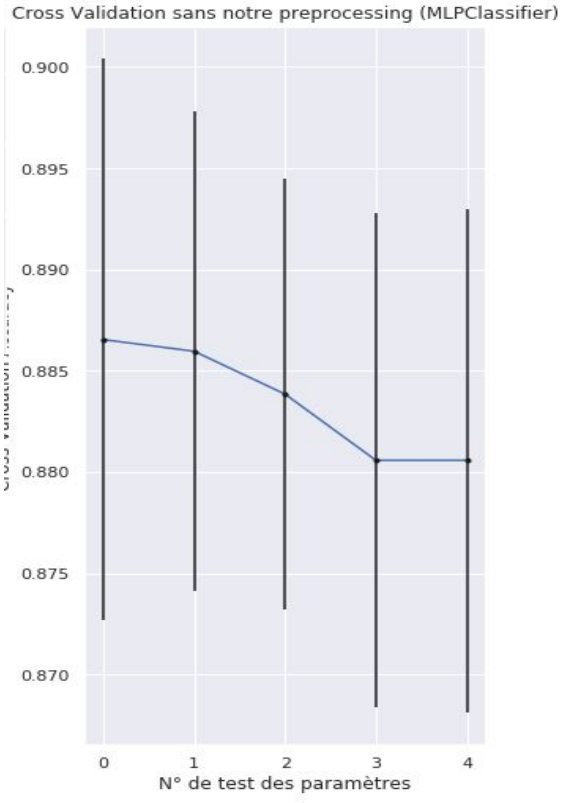
aurions dû utiliser le PCA sur les données brutes mais malheureusement celles ci n'étaient pas accessible à la fac.

```
def transforme(X, i):
    pca = PCA(n_components=i)
    X = pca.fit_transform(X)
    return X
```

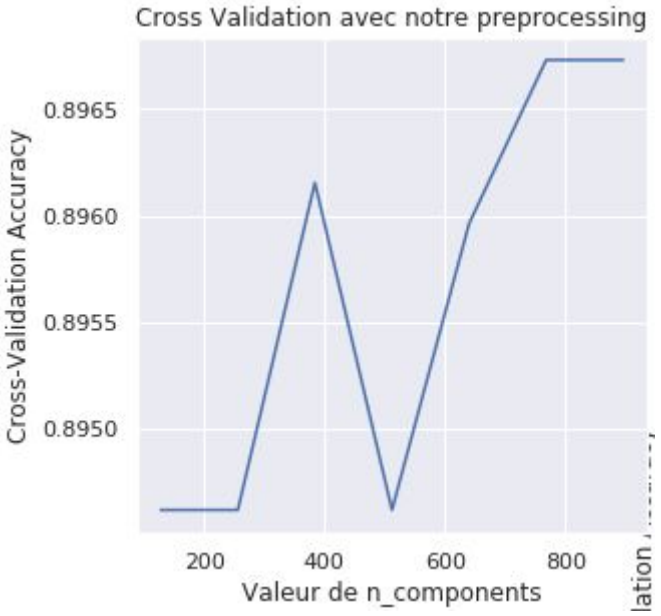
Nous avons réussi à visualiser la Cross Validation avec et sans notre méthode de préprocessing PCA. Il s'agit des graphiques qui mesurent la fiabilité de Cross validation en fonctions des hyperparamètres.



figure(1): Cross-validation sans préprocessing en utilisant SVC.



figure(2): Cross-validation sans préprocessing en utilisant MLP.



figure(3): Cross-validation avec la présence de notre préprocessing.

2) Classifieur : Tifouche Keissa et Slimani Adlene

Le but ce cette partie Classifieur est de pouvoir classifier les données prétraitées (à qui un traitement a déjà été effectué) en 13 classes contenant des images. L'image ci-dessous résume les statistiques exactes fournies dans ce projet concernant nos données.

Dataset	Nombre d'exemples	Nombre de features	Variables catégorielles	Données manquantes	Nombre d'exemples par classes
Training	5200	1024	Non	Non	400
Validation	1950	1024	Non	Non	150
Test	1950	1024	Non	Non	150

figure(4): Ce tableau représente les statistiques sur les données fournies dans le projet.

Pour commencer, nous avons comparé plusieurs algorithmes possibles sur les données d'entraînement en utilisant la bibliothèque "sklearn". A travers cette dernière, nous avons déjà déterminé quelques méthodes algorithmiques utiles pour effectuer notre tâche: la validation croisée du k-fold consiste à n'utiliser qu'une partie des données pour apprendre et à se servir des autres données pour tester le résultat et les méthodes naïves de Bayes ou même la méthode de OneRule Classifier vu durant le TP2. Ainsi, notre choix se portera sur l'algorithme le plus efficace avec les meilleurs résultats de fiabilité (Accuracy), c'est-à-dire, plus le pourcentage de fiabilité est grand, plus l'algorithme est meilleur.

Nous avons vu également le perceptron multi-classes qui utilise un algorithme d'apprentissage similaire à celui pour le cas binaire. Tous les vecteurs de pondération sont d'abord initialisés à zéro, puis plusieurs itérations sont effectuées sur les données

d'apprentissage, avec ajustement des vecteurs de pondération chaque fois qu'une donnée est mal classée. L'algorithme revient donc à apprendre *m* (où *m*=13) classifieurs binaires, le premier différenciant les données de la classe 1 de toutes les autres, le second celles de la classe 2 de toutes les autres, etc. Cela donne le pseudo-code suivant (réf. 1) :

Algorithme : Perceptron multi-classe

Entrée : une liste de données d'apprentissage $(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)$ où $y^i \in \{c_1, c_2, \ldots, c_m\}$, le nombre d'itérations N

Sortie : les vecteurs de pondération $w_{c_1}, w_{c_2}, \ldots, w_{c_m}$

Initialiser les vecteurs w_{c_k} à 0 pour k de 1 à m

Pour *itération* allant de 1 à N **faire**

  Pour chaque exemple (x^i, y^i) **faire**

    Calculer la prédiction $\hat{y}^i = argmax_{c_k} \langle w_{c_k}, x^i \rangle$

    Si $\hat{y}^i \neq y^i$ **alors**

      Augmenter le poids de la classe correcte : $w_{y^i} = w_{y^i} + x^i$

      Diminuer le poids de la classe prédite : $w_{\hat{y}^i} = w_{\hat{y}^i} - x^i$

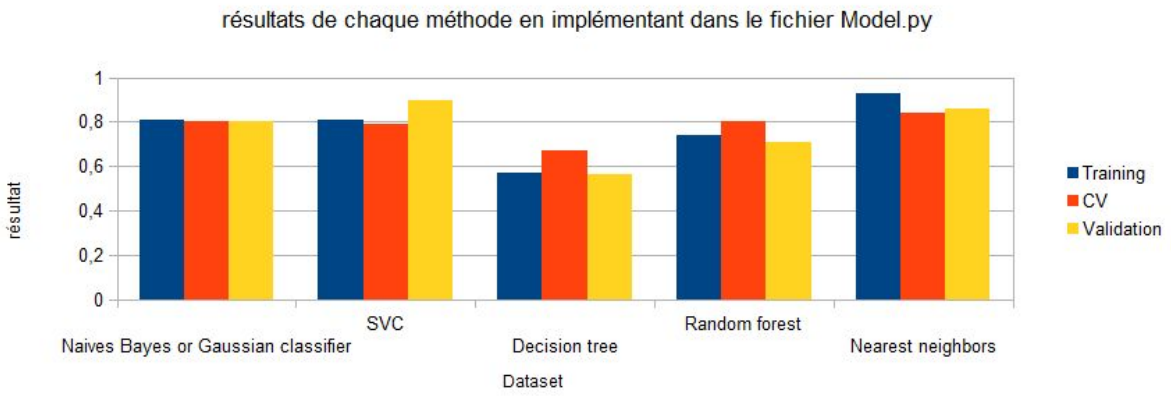
Retourner la liste $[w_{c_1}, w_{c_2}, \ldots, w_{c_m}]$

figure(5): Pseudo-code du perceptron multiclasse (MLP).

Pour tester ce code, nous l'avons implémenté dans le fichier Model.py se trouvant dans le dossier téléchargé du projet, en remplaçant la méthode Decision Tree (méthode déjà existante) par l'algorithme fourmi (Perceptron multi-classe) ou meme le morceau de code existant sur la bibliothèque *Sklearn*. Apres plusieurs essais d'implémentation d'autres méthodes comme Naives Bayes, Gaussian Classifier et Nearest neighbors, les résultats sont implémentés dans le tableau ci-dessous: (ref. 3)

Dataset	Naives Bayes or Gaussian classifier	SVC	Decision tree	Random forest	Nearest neighbors
Training	0.81	0.8106	0.57	0.74	0.9267
CV	0.80	0.79	0.67	0.80	0.84
Validation	0.80	0.8974	0.56	0.71	0.8569

figure(6): Tableau résumant les résultats des méthodes de classification



figure(7): Histogramme des résultats des méthodes de classification

On remarque que le Glaussian classifier, nearest neighbors et SVC nous donnent des résultats satisfaisants. Ainsi que Decision tree et Random forest qui ont un score plus élevé en CV quand Validation on peut en déduire qu'il on fait du sur-apprentissage.
Evidemment, nous avons choisi de garder SVC qui nous a donné un résultat qu'on juge bon et qui vaut 0.8974, par contre au niveau de Training et Cross Validation, Nearest neighbors est meilleur. Malheureusement, nous n'avons pas eu assez de temps pour approfondir nos analyses et nos essais sur ce dernier qui nous semble très intéressant.

Cependant, ces différents classifier nécessitent de nombreux paramètres. Ainsi, nous avons utilisé GridSearchCV. Cette fonction de ScikitLearn, permet de déterminer les meilleurs paramètres que peut avoir un classifier en fonction des données. Cela nous permettra d'améliorer nos résultats car nous aurons les paramètres les plus optimaux. Voici un exemple d'appel à cette fonction :

```
grid = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring='accuracy')
```

Ici, "SVC()" correspond au classifier dont les paramètres vont être testés, "tuned_parameters" correspond aux différentes valeurs que peuvent prendre certains (ou la totalité) des paramètres qui sont définis par :

```
tuned_parameters = {'kernel':('linear', 'rbf'),'C':[1, 10, 100], 'gamma': [1e-3, 1e-4]}
```

et les derniers paramètres correspondent au critère de sélection, dans notre cas une cross-validation.
Finalement, nous avons entraîné le modèle avec les différents paramètres pour que la recherche nous donne les paramètres les plus adaptés à notre cas, grâce à :

```
grid.fit(X_train, Y_train)
```

ce qui nous donne accès au meilleur paramètre et score du classifier. Il nous reste donc qu'à appliquer le classifier à nos données.

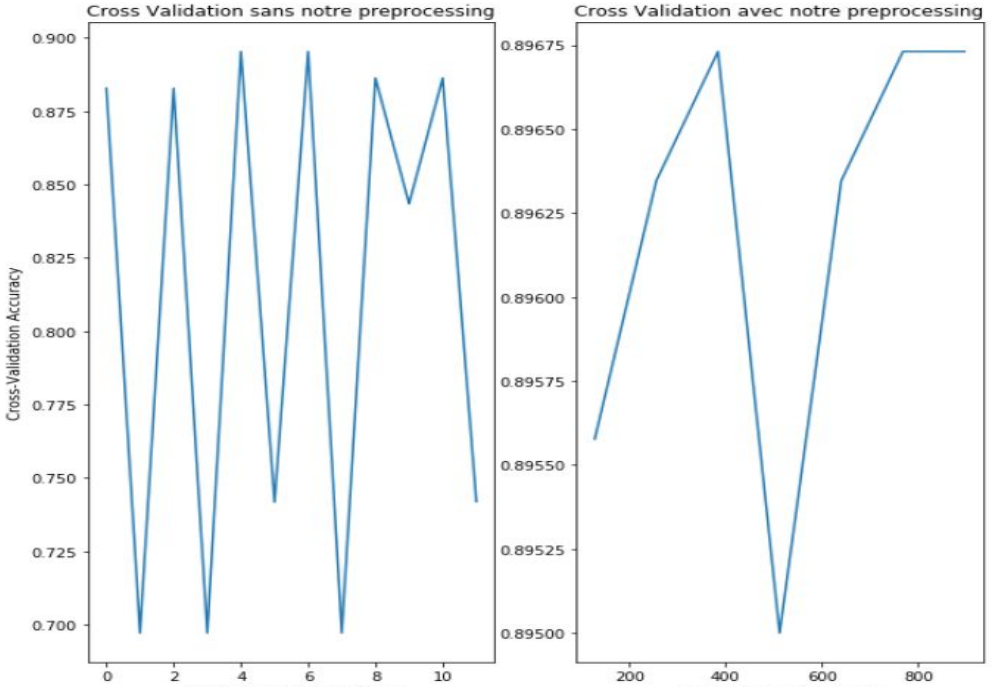
Interface graphique : Marc Bourbonnot et Florian Pinto

L'objectif de la partie graphique du projet est de pouvoir représenter les informations que nous possédons concernant le projets de manière plus équivoque, ainsi que les résultats obtenus à partir de ces informations, en leur effectuant de différentes transformations en utilisant des méthodes de classification. Ces méthodes sont disponibles dans les bibliothèques standard vues durant les TP précédents. En effet, le but est de transformer des informations en graphes faciles à visualiser au lieu de les traiter sous forme de chiffres.

En ce qui concerne la visualisation, nous avons utilisé principalement la bibliothèque Matplotlib, qui permet de faire des graphes, ainsi que les affichages basique que nous propose python. Ainsi, le code (ci-dessous) nous a permit de donner les graphiques suivant :

```
plt.figure(figsize=(10, 10))
ax = plt.subplot(121)
plt.plot(range(0, len(grid_mean_scores)), grid_mean_scores)
plt.xlabel('N° de test des paramètres')
plt.ylabel('Cross-Validation Accuracy')

plt.title('Cross Validation sans notre preprocessing')
plt.show
plt.subplot(122)
plt.plot(range(128, 1024, 128), valeurs_prepro_GSCV)
plt.xlabel('Valeur de n_components')
#plt.ylabel('Cross-Validation Accuracy')
plt.title('Cross Validation avec notre preprocessing')
plt.show
```



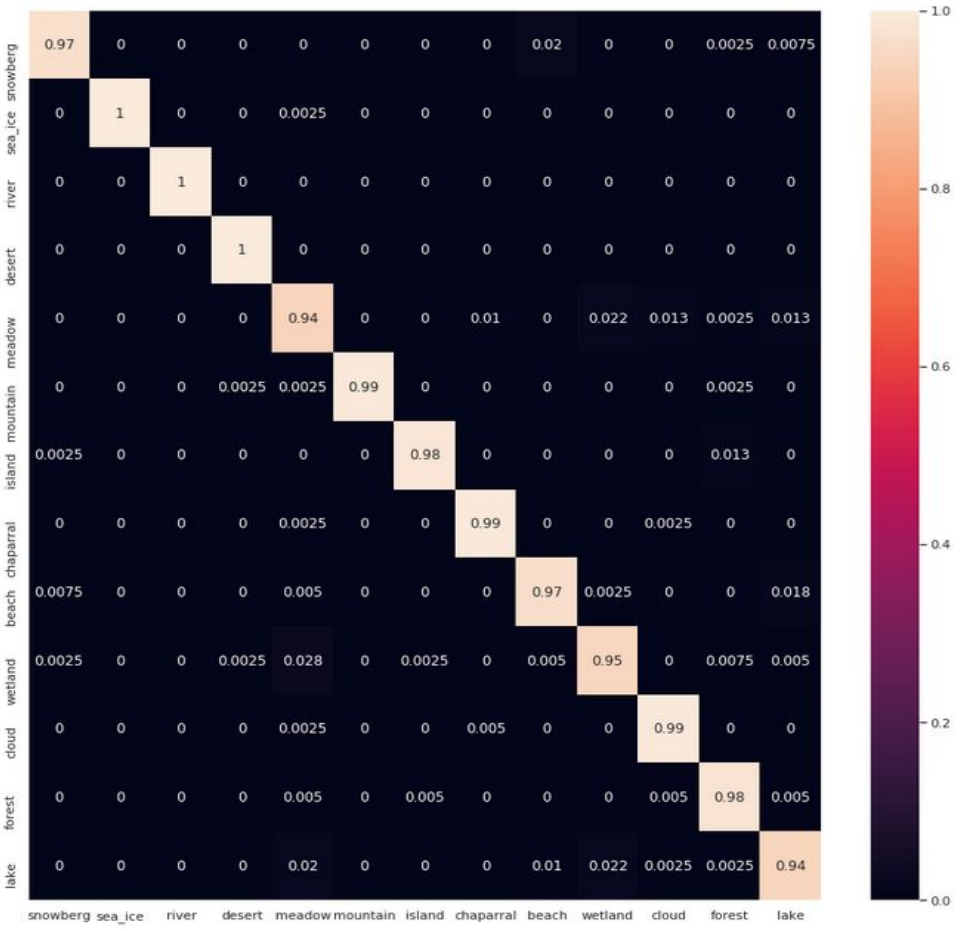
figure(8): Graphiques représentant Cross Validation sans et avec la méthode du préprocessing

Ici, ce code nous permet de faire un affichage de nos données de cross-validation. Le graphique de gauche représente les différents paramètres possibles pour le classifier en fonction du score obtenu en cross-validation (correspondant à la partie haute du code). On peut voir que les valeurs sont très proches, mais que nous allons pouvoir trouver des paramètres qui vont nous donner un meilleur résultat. Le graphique de droite, correspondant à la partie basse des deux blocs de code, représente l'évolution du score de cross-validation en fonction des différentes valeurs du paramètre "n_components" de notre fonction de preprocessing. De même, on peut remarquer qu'une valeur sort du lot et ce sera celle-ci qu'on utilisera.

```
print("Pour n_components = {}, on obtient un score de {}.".format(i, round(res[2], 3)))
```

La ligne de code ci-dessus, quand à elle, correspond à un affichage simple avec python. Celle-ci permet de se rendre compte de l'évolution de l'exécution du code (pas très long donc pas beaucoup à afficher) et elle permet aussi de donner des valeurs, qui permettent de vérifier le graphique.

Dans la partie visualisation, l'une des méthodes les plus importantes pour mieux visualiser les données est la matrice de confusion heatmap, extraite dans la bibliothèque "Seaborn" de visualisation de données Python basée sur matplotlib. Le principe est de tracer les données rectangulaires sous forme de matrice codée en couleur; c'est-à-dire un carré par valeur et une couleur qui dépend de cette valeur. L'image ci-dessous présente la matrice de confusion de nos datas après la classification, le plus bas résultat qu'on a trouvé était de 0.94, ce qui montre la bonne fiabilité de notre classifier. On est tombé quelques fois aussi sur 1, ce qui est parfait! (ref. 4)



figure(9): Matrice de confusion de nos données en fonction des 13 classes.

Quelques précisions :

1. Cross validation :

La *validation croisée* désigne le processus qui permet tester la précision prédictive d'un modèle dans un échantillon test par rapport à la précision prédictive de l'échantillon d'apprentissage à partir duquel le modèle a été développé. Elle est fondé sur une technique d'échantillonnage

1) On divise l'échantillon en original en K échantillon
2) On sélectionne un des échantillons comme ensemble de validation et les autres échantillons constitueront l'ensemble d'apprentissage.
3) On calcule le score de performance, et on répète l'opération pour que chaque sous-échantillon ait été utilisé exactement une fois comme ensemble de validation.
4) On calcule le score de performance en faisant une moyenne de tous les autres scores.

2. Sur-apprentissage:

En statistique, le sur-apprentissage ou sur-ajustement (en anglais « overfitting ») est une analyse statistique qui correspond trop étroitement ou exactement à un ensemble particulier de données.
Ainsi, cette analyse peut ne pas correspondre à des données supplémentaires et ne pas prévoir de manière fiable les observations futures.

3. La métrique utilisée dans ce projet:

La métrique permet d'évaluer la performances de notre modèle. Ce calcul correspond au **nombres de bon résultats sur le nombre d'exemples**. Des valeurs élevées signifient que notre modèle à une bonne fiabilitée. Ce score est compris entre 0 et 1! Le code ci-dessous montre comment on peut réaliser la fonction "accurancy" utilisée dans notre projet.

```
1 import numpy as np
2 import scipy as sp
3
4 def accuracy(solution, prediction):
5         error = 0
6         for sol, pred in zip(solution, prediction):
7                 if sol != pred:
8                         error += 1
9         return 1 - (error / len(solution))
```

figure(10): Code la fonction "accurancy"

4. Naives Bayes:

Le classifieur bayesien naïf est une méthode probabiliste simple d'apprentissage supervisé basé sur le Théorème de Bayes. Il suppose que l'existence d'une caractéristique pour une classe, est indépendante de l'existence d'autres caractéristiques. Un fruit peut être considéré comme une pomme s'il est rouge, arrondi, et fait une dizaine de centimètres. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que le fruit est une pomme en considérant indépendamment ces caractéristiques de couleur, de forme et de taille.

5. Support Vector Classification:

SVC est une méthode de classification. L'implémentation est basée sur libsvm. La complexité temporelle de l'ajustement est plus que quadratique par rapport au nombre d'échantillons, ce qui rend difficile la mise à l'échelle d'un jeu de données comprenant plus de deux cent mille échantillons.
La prise en charge multiclass est gérée selon un schéma un contre un.
La formulation mathématique sur la manière dont les gamma, le coef0 et le degré s'affectent est basée sur les fonctions de Kernel (noyau).

6. Decicion Tree:

Les arbres de décision (DT) est une méthode d'apprentissage supervisé non paramétrique utilisée pour la classification et la régression. L'objectif est de créer un modèle qui prédit la valeur d'une variable cible en apprenant des règles de décision simples déduites des entités de données. Les avantages de cette méthode sont:

1) Simple à comprendre et à interpréter. Les arbres peuvent être visualisés.
2) Capable de traiter à la fois des données numériques et catégoriques
3) Possible de valider un modèle à l'aide de tests statistiques. Cela permet de rendre compte de la fiabilité du modèle
4) Nécessite peu de préparation de données. D'autres techniques nécessitent souvent une normalisation des données, des variables nominales doivent être créées et des valeurs vierges doivent être supprimées. Notez cependant que ce module ne prend pas en charge les valeurs manquantes.

7. Random Forest Classifier:

Une forêt aléatoire est un méta-estimateur qui adapte un certain nombre de classificateurs d'arborescence de décision à différents sous-échantillons de l'ensemble de données et utilise la moyenne pour améliorer la précision prédictive et contrôler le sur-ajustement. La taille du sous-échantillon est toujours identique à la taille de l'échantillon d'origine, mais les échantillons sont dessinés avec remplacement si bootstrap = True (par défaut).

Conclusion:

A travers ce projet Areal qui consiste à classifier des images capturées par un satellite en plusieurs classes selon leurs critères, nous avons mis à l'épreuve les différentes méthodes de l'apprentissage supervisé permettant de valider (ou d'infirmer) la valeur de l'apprentissage. Comme ces images sont sous formes de données prétraitées auxquelles un traitement d'image a déjà été exécuté (Partie 1), nous avons directement commencé la classification.

Nous avons essayé de résoudre la problématique de ce projet en essayant plusieurs méthodes de classification et en variant aussi leur paramètres grâce à gridsearch. Nous étions séduits par la fiabilité du *classifier SVC,* qui nous a donné un résultat satisfaisant (0.8974). Cette fiabilité a été calculé en sommant le nombre de bonnes réponses par classe et diviser le résultat sur le nombre total d'exemples.

Une autre étape importante dans l'avancement du projet est la partie visualisation. Cette dernière nous a permit de mieux visualiser nos données et les transformations faites sur les exemples. En utilisant la matrice de corrélation, des histogrammes, des tableaux descriptifs… nous avons mieux géré l'analyse de nos données.

La classification supervisée est l'une des tâches les plus importantes en fouille de données (Data Mining), elle-même constituant une des étapes essentielle d'un projet d'extraction de connaissances à partir de données.

Ce projet nous a permis d'appréhender le travail en équipe, la rigueur nécessaire ainsi que le respect des dates butoires indispensable pour ce genre de travail.

Références :

(1) : TP4 de Perceptron - (M2IS IN) Université Marseille. Rémi Eyraud. 2018.
(2) : https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
(3) : https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
(4) : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html