

Challenge Areal - groupe Sputnik (grp.Sputnik)

Groupe Administratif 3 et 4

Membres : VERRIEZ CHASTANG Mathilde,

DORDAIN Jean-Camille, HERBERA Mathis,

ZENAIDI Lydia, DAHASSE Rayane, HUART Kevin

URL du challenge: <https://codalab.lri.fr/competitions/399> et <https://codalab.lri.fr/competitions/379>

Numéro de soumission du code : 7650

URL du diapo :

https://github.com/grpSputnik/Sputnik/blob/master/Diapo_Video_Mini_Projet-FINAL-DIAPPO.pdf

Repo GitHub du projet : <https://github.com/grpSputnik/Sputnik>

Repo Youtube du projet: <https://www.youtube.com/watch?v=rp-SxUgZ-p4&feature=youtu.be>

L2 MI – Mini Projet

INTRODUCTION:

Background

Depuis l'antiquité, l'Homme a toujours souhaité cartographier la planète afin de se repérer, se déplacer et bâtir. Depuis le XXe siècle, la cartographie a évolué en devenant plus juste grâce aux nouveaux moyens mis en place (photographie, dessin assisté d'ordinateur, image satellite, ...). La nouvelle étape logique semble être la reconnaissance automatique du terrain.

Les applications possibles sont grandes : déplacement autonome d'un robot/drone, étude du terrain, ... etc.

Material and methods

Le but de notre projet est de classer des images de paysages naturelles en 13 catégories différentes. Le set de données brut comporte 9100 images satellites de taille 128*128 qu'il faudra regrouper en 700 images par classe. Les classes sont une sorte d'environnement naturel ou paysage tel que: plage, nuage, lac, forêt...

Notre challenge est séparé en deux parties(ou deux niveaux de difficultés). La première consiste à se servir de données pré traitées: plusieurs variables ont été regroupées en une seule, ce sont des données simplifiées en vecteur de 1024 features.

La deuxième partie consiste à se servir de données brutes que l'on devra nous même pré traiter afin de pouvoir ensuite les classer.

Des exemples d'applications quotidiens seraient l'évaluation et l'estimation des quantités de récoltes, des déploiements militaires...

Pour réaliser le projet, nous nous sommes séparés en 3 groupes: l'un s'occupant du processing des données brutes, l'autre des prédictions et le dernier de la visualisation.

Le preprocessing est une étape importante, celle-ci va se charger de transformer des données brutes en données compréhensibles pour le classifieur et le rendre beaucoup plus performant.

La prédiction a pour but de classer les données préprocessées en utilisant des méthodes de classification de la bibliothèque sklearn. Le travail effectué a été dans la modification et l'optimisation du model.py, validation sur codalab (meilleur score ;) ainsi que fournir les résultats à la partie visualisation.

La visualisation a pour but de représenter les résultats afin de les présenter, et d'en faire ressortir le sens. Le travail effectué va de la réalisation des graphes présents dans ce rapport, ainsi que la représentation des images brute et celle preprocess.

Results

Meilleur score de validation sur codalab, en utilisant le classifieur SVM avec optimisation nous obtenons un score de 0.899 (meilleur que celui des Master 2 ;)

Meilleur score de validation sur codalab avec preprocessing sur données preprocess : 0.45.

Description rapide des classes :

Processing(DORDAIN Jean-Camille & HERBERA Mathis)

Le preprocessing est une étape importante, celle-ci va se charger de transformer des données brutes en données compréhensible pour le classifieur et le rendre beaucoup plus performant.

En effet, des données brutes (par exemple des images) peuvent souvent être incomplètes et susceptibles de contenir beaucoup d'erreurs. Il ne serait donc pas envisageable d'utiliser des algorithmes de machine learning pour de telles données.

Pour revenir au projet Areal, nous avons 9100 images brutes regroupant images d'entraînement, validation et de test. Chaque image brute est représentée par une matrice et chaque donnée dans cette matrice possède des features, une feature pouvant prendre plusieurs formes.

L'idée ici serait de réduire le nombre de données par image en ne prenant que les données les plus pertinentes par rapport aux autres, pour cela, nous allons utiliser un algorithme de feature selection qui s'occupera donc de retirer toutes les features n'atteignant pas un certain seuil de variance(exemple voir figure 1).

Une autre méthode serait d'utiliser une méthode récursive d'élimination de features(appelé RFE).(ref [5])

Celle-ci va récursivement choisir des features en prenant à chaque fois des features de plus en plus petites. Pour ce faire, un estimateur est entraîné en s'aidant de toutes les features disponible, puis l'importance de chaque feature va être obtenu avec : "feature_importances_", nous donnant l'importance pour chacune d'elles (voir exemple figure 2) . Enfin, les moins importantes sont retirées. On répète cette procédure autant de fois que nécessaire jusqu'à arriver au nombre de features désirées.

Cette méthode est très efficace mais peut prendre beaucoup plus de temps que l'autre méthode. En effet, plus il y a de features, plus le temps sera long parce que pour chaque features, toutes les autres features seront appelées. Nous allons donc plus nous intéresser sur la première méthode.

Ensuite, il est nécessaire d'utiliser l'algorithme Principal Component analysis(PCA)([3]-[4]). Celle-ci va s'occuper de modifier les features en de nouvelles features, tout ça en réduisant la dimension de ces dernières, avec quelques pertes d'informations.

La grande différence entre ces deux algorithmes est que la première va juste se charger de sélectionner les meilleures features par rapport aux autres sans les modifier, alors que le PCA va lui, modifier les features en des dimensions plus petites.

C'est pourquoi de bons algorithmes de preprocessing sont importants pour espérer avoir les meilleurs résultats possibles et ce, en un meilleur temps possible. Nous avons ici un ensemble de 6 vecteurs de dimension 3 représentés sur chaque ligne.

L'idée ici est de retirer des features(colonne) qui sont à 0 ou à 1 en fonction du taux d'apparition de ces 0 ou 1.

On va donc parcourir toutes les colonnes et retirer celles qui ont les variances les plus basses à définir.(par exemple 0.10). La variance étant définie par :

$$\text{Var}[X] = p(1-p).$$

La première colonne est celle avec la variance la plus basse (environ 0.14) avec 5 0 et un seul 1. Nous allons donc la retirer et de ce fait retirer une dimension pour les 3 vecteurs.

Pseudo-code Preprocessing : [1] et [2]

<p><u>Méthode avec variance basse:</u></p> <pre>def transform(self, X, y=None): self = VarianceThreshold(threshold=(0.05)) X = sel.fit_transform(X)</pre> <p><u>Méthode récursive d'élimination :</u></p> <pre>rfe = RFE(model, 40) fit = rfe.fit(X, Y) print("Num Features: %d") % fit.n_features_ print("Selected Features: %s") % fit.support_ print("Feature Ranking: %s") % fit.ranking_</pre> <p>"Selection des donnees avec les algorithmes PCA"</p> <pre>pca = PCA(n_components=self.n_pca) X=pca.fit_transform(X) scaler=StandardScaler() X=scaler.fit_transform(X) return X</pre> <pre>def fit_transform(self, X, y): return self.fit(X,y).transform(X)</pre>	<p>La méthode transform va retirer les features impertinentes en utilisant la variance treshold, et les retirer si la variance est inférieure à 0,05.</p> <p>Cette méthode va donc prendre en paramètre le nombre de meilleures features que l'on veut garder(ici en exemple : 40). Les print qui suivent vont s'occuper d'afficher le nombre de features gardées et le rang pour chacune d'elles. Sachant que cette méthode sera moins bien que l'autre en raison du grand nombre de features.</p> <p>Puis elle va standardiser les données pour qu'elles soient utilisable.</p> <p>On utilise le PCA pour modifier les images avec des features différentes.</p> <p>La méthode fit_transform va sélectionner les features les plus importantes.</p>
---	---

Après avoir fait la description des algorithmes étudiés et pseudo-code, nous avons décidé de ne pas tout utiliser, et d'utiliser que certains algorithmes dans un ordre bien précis pour avoir un meilleur résultat possible.

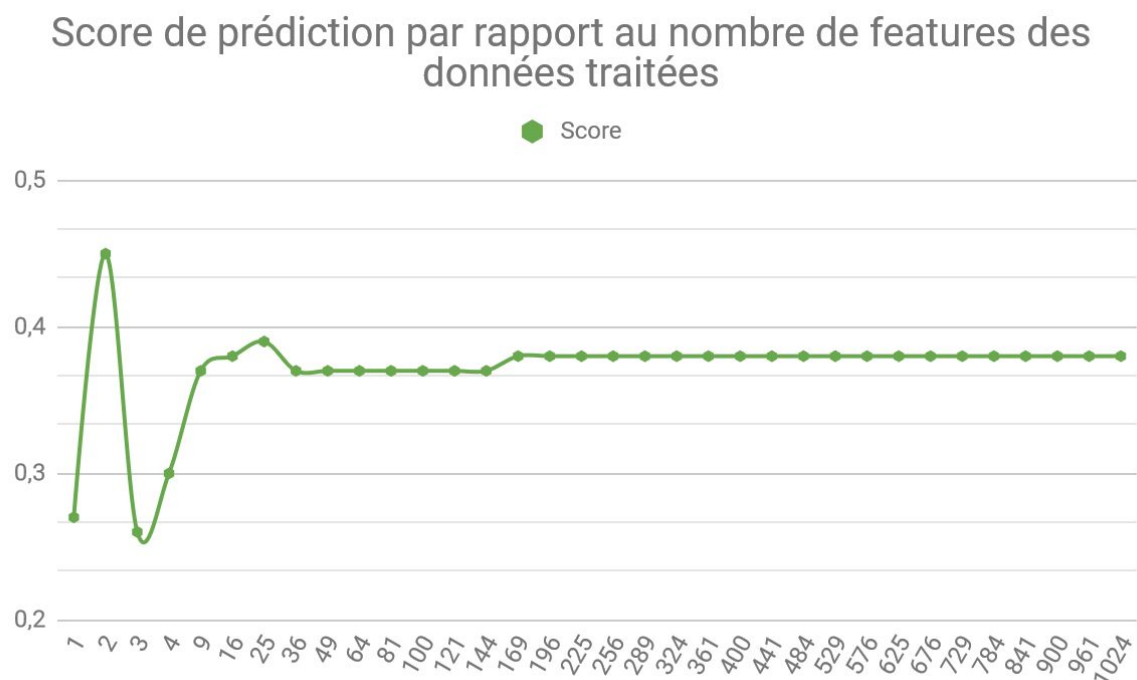
En ayant fait des tests sur des données déjà preprocess, nous avons sélectionné les best features en fonction de la variance (de 0 et de 1), puis on a fini sur un PCA (Principal Component Analysis).

L'analyse en composantes principales (PCA), est une méthode de la famille de l'analyse des données et plus généralement de la statistique multivariée, qui consiste à transformer des variables liées entre elles en nouvelles variables décorrélées les unes des autres. Ces nouvelles variables sont nommées axes principaux. Elle permet au praticien de réduire le nombre de variables et de rendre l'information moins redondante. Il s'agit d'une approche à la fois géométrique et statistique. Lorsqu'on veut compresser un ensemble de N variables aléatoires, les n premiers axes de l'analyse en composantes principales sont un meilleur choix, du point de vue de l'inertie ou de la variance. Exemple : dans le cas d'une image, les pixels sont représentés dans un plan et considérés comme une variable aléatoire à deux dimensions. Le 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. (source : wikipédia).

L'algorithme de PCA s'occupe de faire une combinaison linéaire, et fait une somme pondérée. Il ne prend pas les classes Y en paramètre mais seulement les données, c'est un algorithme non supervisé. Le PCA est un technique permettant de trouver les directions des variances maximales entre les features.

Avec cette méthode, nous avons atteint un score de 0.45 sur codalab avec le paramètre `n_components = 2`.

Nous avons essayé de n'utiliser que le PCA pour voir si le score était similaire pour les données déjà préprocéssées, à savoir laisser en paramètre le nombre de features que l'on souhaitait à 1024, mais nous nous sommes rendu compte que le score était beaucoup moins bon. En essayant avec différents paramètres, nous avons eu les résultats suivants :



Le score étant bas pour 1024 features en paramètre ne devrait pas et devrait renvoyer le même score car il devrait y avoir une complète rotation des données. Mais l'algorithme de PCA modifie les valeurs des features avant de faire la rotation, d'où le problème du score très bas.

Avec le nombre de features fixé à 2 en paramètre du PCA, nous avons un résultat de 0.45 sur codalab. Le résultat est étonnant car en prenant seulement 2 features par image, on arrive à un meilleur score que pour les 1024 features.

Nous avons aussi décidé d'utiliser un feature selection en fonction de la variance avant d'utiliser le PCA. Comme les données sont déjà préprocessées, la variancethreshold ne change en rien le score, tout simplement parce que les données ont une variance trop élevée et ne sont pas retirées.

L'algorithme de feature selection est alors une étape très importante pour que le PCA puisse être performant.

Une autre méthode intéressante pour réduire le nombre de features autre que PCA (principal component analysis) est le LDA (linear discriminant analysis). Contrairement au PCA, le LDA est un algorithme supervisé prenant en paramètre les data, mais aussi le label Y associé aux data. Le LDA va se charger de trouver un sous espace de feature permettant de mieux séparer les différentes classes, c'est pourquoi il a besoin du label.

La grande différence entre les deux est que le PCA est utilisé pour extraire uniquement les meilleures features en se basant sur la variance, on essaye de trouver la meilleure projection possible où les variations sont maximisées.

Le LDA, va, quant à lui être utilisé pour la classification et va projeter les données avec les labels, pour améliorer le résultat.

Afin de bien comprendre la différence entre les deux, il faut faire des graphiques pour chacun de ces algorithmes.

Visualisation(Verriez Chastang & Huart Kevin)

Le but de la partie visualisation est de montrer les résultats par des représentations graphiques.

Afin de comprendre l'utilité de l'étape de preprocessing, voici quelques images avant et après cette étape :


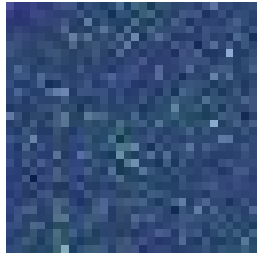

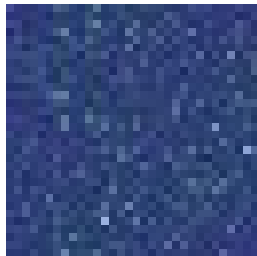
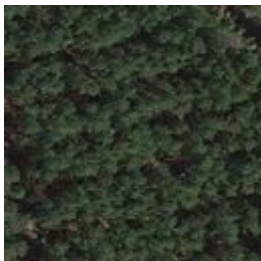
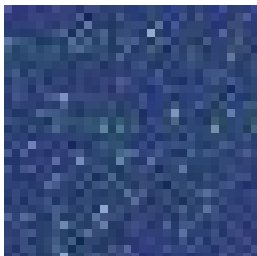

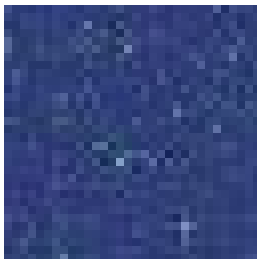
Nom	Avant preprocess	Après preprocess
Island		
Chaparral		
Forest		
Snowberg		

Figure -B-

On peut observer dans la Figure -B- que les données à traiter lors de la partie prédiction sont beaucoup moins nombreuses, ce qui permet d'être beaucoup plus efficace (et ne pas prendre un an pour faire tourner l'ensemble d'images d'entraînement !).

Les autres graphiques ont été réalisés grâce à Google Sheet, car il était plus simple de passer par un tableur.

Prediction(Lydia ZENAIDI, Rayane DAHASSE)

Le processing étant une étape cruciale dans la prédiction des classes pour les images pré-traitées, il a été important pour nous de trouver le meilleur algorithme de classification dans un premier temps sans amélioration des algorithmes testés ce qui se conclut par le choix du classifieur KNearestNeighbors qui donnait le meilleur score (ref [a][b][c][d][e]), puis recommencer cette étape en améliorant chacune des méthodes en faisant varier les hyper-paramètres.

La variation des hyper-paramètres se faisant de manière non optimale (à la main), nous avons cherché une manière plus automatisée de faire varier les hyper-paramètres dans un champ plus large valeurs, pour cela nous avons trouvé deux solutions : le GridSearch et le RandomizedSearch qui sont deux méthodes prenant toutes deux en paramètres une liste d'hyper-paramètres ainsi que l'intervalle de valeurs associées à chacun, cela prenait certes plus de temps que manuellement (on a failli craquer) mais au moins on était sûr d'obtenir le meilleur résultat possible tout en récupérant les hyper-paramètres utilisés. Nous nous sommes donc retrouvé avec un autre classifieur qui donne le plus haut score de classification qui est SVM.

Le code ayant permis de trouver ce résultat est le suivant :

```
params = {'C': [6,7,8,9,10], 'kernel': ['linear','rbf'], 'degree':[4,5,6]}  
clf = svm.SVC() (ref [f][g])  
self.classifier = RandomizedSearchCV(clf, params)
```

puis pour récupérer les meilleurs hyper-paramètres trouvés on ajoute après le FIT :

```
print("Best Hyper Parameters:\n", self.classifier.best_params_)
```

Nous vous proposons donc un diagramme représentant la variation des scores de chaque méthode avant optimisation et après optimisation :

Résultats de Cross-Validation des différentes méthodes avant et après optimisation

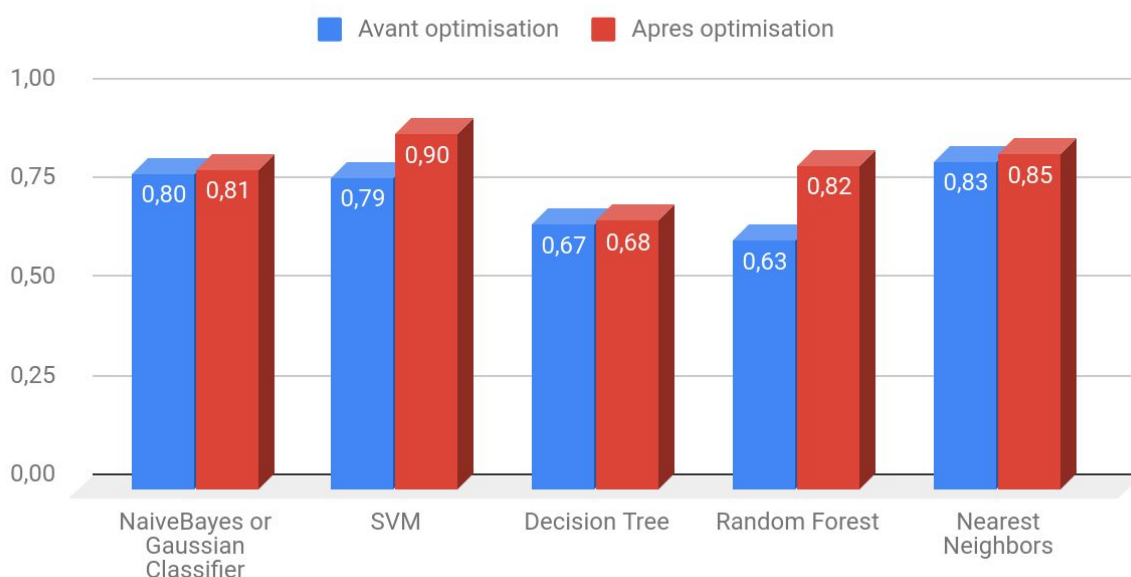


Figure -C-

Comme vous pouvez le voir dans la Figure -C- avant optimisation des hyper-paramètres, comme vous pouvez le constater nous avons obtenu un score de cross-validation plus élevé en utilisant l'algorithme de Nearest Neighbors, or après optimisation des hyper-paramètres vous pouvez remarquer que le classifieur qui obtient le meilleur score de cross-validation devient alors SVM, c'est pour cela que nous l'avons gardé dans notre model.py!

Bonus :

- Définition de la cross-validation : La validation croisée ou cross-validation est, en apprentissage automatique, une méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage. Utilité : supposons posséder un modèle statistique avec un ou plusieurs paramètres inconnus, et un ensemble de données d'apprentissage sur lequel on peut entraîner le modèle. Le processus d'apprentissage optimise les paramètres du modèle afin que celui-ci corresponde aux données le mieux possible. Si on prend ensuite un échantillon de validation indépendant issu de la même population d'entraînement, il s'avérera en général que le modèle ne réagit pas aussi bien à la validation que durant l'entraînement : on parle parfois de surapprentissage. La validation croisée est un moyen de prédire l'efficacité d'un modèle sur un ensemble de validation hypothétique lorsqu'un ensemble de validation indépendant et explicite n'est pas disponible. Technique utilisée : il existe au moins trois variantes. La première est «testset validation» : on divise l'échantillon de taille n en deux sous-échantillons, le premier dit d'apprentissage et le second dit de test. Le modèle est bâti sur l'échantillon d'apprentissage et validé sur l'échantillon de test. L'erreur est estimée en calculant un test, une mesure ou un score de performance du modèle sur l'échantillon de test, par exemple l'erreur quadratique moyenne. La deuxième méthode est «k-fold cross-validation» : on divise l'échantillon original en k échantillons, puis on sélectionne un des k échantillons comme ensemble de validation et les $k-1$ autres échantillons constitueront l'ensemble d'apprentissage. On calcule comme dans la première méthode le score de performance, puis on répète l'opération en sélectionnant un autre échantillon de validation parmi les $k-1$ échantillons qui n'ont pas encore été utilisés pour la validation du modèle. L'opération se répète ainsi k fois pour qu'en fin de compte chaque sous-échantillon ait été utilisé exactement une fois comme ensemble de validation. La moyenne des k erreurs quadratiques moyennes est enfin calculée pour estimer l'erreur de prédiction. La 3ème méthode est «leave-one-out cross-validation» (LOOCV) : cas particulier de la deuxième méthode où $k=n$, c'est-à-dire que l'on apprend sur $n-1$ observations puis on valide le modèle sur la n ème observation et l'on répète cette opération n fois. (source : wikipédia).
- Explication du sur-apprentissage : en statistique, le surapprentissage 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 ou ne pas prévoir de manière fiable les observations futures. Un modèle surajusté est un modèle statistique qui contient plus de paramètres que ne peuvent le justifier les données. Le problème existe aussi en apprentissage automatique. Il est en général provoqué par un mauvais dimensionnement de la structure utilisée pour classifier. De par sa trop grande capacité à capturer des informations, une structure dans une situation de sur-apprentissage aura de la peine à généraliser les caractéristiques des données. Elle se comporte alors comme une table contenant tous les échantillons utilisés lors de l'apprentissage et perd ses pouvoirs de prédiction sur de nouveaux échantillons. (source : wikipédia).
- Définition métrique:

la métrique évalue la précision du modèle elle est calculé par le ratio de bonnes réponses (images bien classés par classes) sur le nombre totale d'exemples (nombres d'images) ce calcule produit un score entre 0 et 1, 0 étant le pire et 1 le meilleur. une prédiction de 1/13 s'apparente à une prédiction aléatoire du fait des 13 classes de notre jeu de données. le score acceptable est donc supérieur à 1/13

Scoring metric :

```
def accuracy(solution, prediction):
    error = 0
    for sol, pred in zip(solution, prediction):
        if sol != pred:
            error += 1
    return 1 - (error / len(solution))
```

- **Table 1: Statistics of the data**

Dataset	Num. Examples	Num. Variables/features	Has categorical variables?	Has missing data?	Num. Examples in each class
Training	5200	1024	No	No	400 per class
Validation	1950	1024	No	No	-
Test	1950	1024	No	No	-

- **Table 2: Optimisation results**

Method	NaiveBayes or Gaussian Classifier	SVM	Decision Tree	Random Forest	Nearest Neighbors
Training	0.82	0.97	0.83	1.00	0.89
CV	0.81	0.90	0.68	0.82	0.85

Validation score for SVM : 0.899

references prédiction :

- [a]https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
- [b]<https://scikit-learn.org/stable/modules/svm.html>
- [c]<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
- [d]<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>
- [e]<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>
- [f]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- [g]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

références preprocessing :

- [1]https://github.com/CobaltL2/L2_hiva2/blob/master/L2_hiva/my_code_29_04_17_final/myPreprocessing_unfinished.py
- [2]<https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1ca9>
- [3]<https://www.quora.com/What-is-the-difference-between-principal-component-analysis-PCA-and-feature-selection-in-machine-learning-Is-PCA-a-means-of-feature-selection>
- [4]https://scikit-learn.org/stable/modules/feature_selection.html
- [5]<https://machinelearningmastery.com/feature-selection-machine-learning-python/>
- [6]<https://www.quora.com/What-is-the-difference-between-LDA-and-PCA-for-dimension-reduction>