TP 9 - IA Algorithme KNN et K-Mean

Dans cette activité, nous nous plaçons dans un contexte souvent rencontré dans les cas d'application de l'I.A. Nous souhaitons construire un programme permettant d'associer une entité (un individu) à groupe. Ceci est par exemple rencontré lorsqu'on souhaite faire du ciblage dans des groupes de population (publicité, campagnes d'information...). L'une des difficultés est que nous ne connaissons pas à priori les différents groupes. La première étape consiste donc à déterminer les groupes existants dans la population étudiée. Il faut ensuite déterminer à quel groupe appartient l'entité à classifier. L'activité développée ici suit ce schéma.

I Clustering et Algorithme K-mean

Le jeu de données fourni contient les caractéristiques dimensionnelles du bec de pinsons des iles Galápagos dits 'pinsons de Darwin'. Ces pinçons sont emblématiques des travaux de recherche de Charles Darwin sur l'évolution des espèces. Au cours de son exploration des îles Galapagos, Charles Darwin a prélevé plusieurs spécimens de pinsons sur des îles différentes. Après études, il s'est avéré que tous ces spécimens étaient d'espèces dérivant d'une même souche, mais dont la morphologie a évolué par sélection naturelle pour s'adapter aux sources de nourritures différentes disponibles sur chaque île. La forme du bec notamment a évolué, permettant à certains une adaptation à la chair des cactus, pour d'autres à l'ouverture de graines, ou encore à la chasse aux insectes.

I.1 Importation des données

Le script fourni contient une première partie permettant l'importation des données.

Manipulation

Exécutez le script afin d'importer les données, puis utilisez l'explorateur de variable pour observer la structure de la variable **donnees**.

I.2 Librairie Scikit.learn

Au cours du TP, nous utiliserons le module scikit learn. Scikit learn est un module très complet comprenant de nombreux algorithmes d'apprentissage machine, mais aussi des jeux de données, et des outils n'analyse des résultats, de mise en forme des données...

A la suite du script, dans la partie « Clustering scikit learn – Premier essai », vous trouverez le code suivant :

```
from sklearn.cluster import KMeans

KM=KMeans(n_clusters=4, random_state=170) #instancie un objet informatique implémentant la méthode K-Mean

Labels_predits = KM.fit_predict(donnees)  #Utilise l'objet K-Mean pour la détermination des clusters


les données contenues dans la variable donnees
```

Manipulation

Décommentez cette partie du code.

Vous pouvez observer que la fonction KMeans prend deux arguments :

— *n_clusters* : le nombre de groupes à construire, fixé arbitrairement à 4 pour cette première application.

— *random_state* : un nombre entier nécessaire pour l'initialisation pseudo-aléatoire des centroïdes. Utiliser deux fois le même nombre permet d'obtenir deux initialisations pseudo-aléatoires identiques.

Manipulation

Exécutez le code de la partie « Clustering scikit learn – Premier essai », puis observez le résultat renvoyé par la méthode *fit_predict*. A quoi correspond-il ?

I.3 Nombre optimal de clusters

Nous ne savons pas encore combien d'espèces sont pinsons sont représentées dans notre jeu de données. Pour le déterminer, nous allons déterminer le nombre K optimal de clusters minimisant la distance intra-classe.

Dans la partie « nombre optimal de clusters » du script python fourni, vous trouvez les lignes de code suivantes :

```
KM=KMeans(n_clusters=K, random_state=170)
Labels_predits = KM.fit_predict(donnees)
Inertia=KM.inertia_
```

KM est l'instatiation d'un objet informatique, dont l'attribut inertia_ correspond à l'indicateur basé sur la variance de la distance intra-classe vu en cours. Plus inertia_ est faible, plus la somme des distances point-centroïde est faible.

Nous souhaitons tracer l'évolution de l'attribut inertia_ en fonction du nombre de cluster K utilisé pour le calcul.

Manipulation

Initialser une liste de valeurs de K pour lesquelles vous souhaitez calculer inertia_. Proposer un code python appliquant la méthode KMeans (utilisez les lignes de code précédente) et calculant inertia_ pour chacune des valeurs de K choisies. Votre programme produira une liste des valeurs de inertia_.
A la suite du script, toujours dans la partie « nombre optimal de clusters », tapez les lignes de codes permettant d'afficher l'évolution de la valeur de inertia_ en fonction de du nombre K de clusters.
En utilisant le graphique affiché, déterminer le nombre d'espèces présentes dans nos données.

I.4 Affichage des clusters

Dans le script fourni, la partie « Affichage des clusters » permet d'afficher les clusters.

Manipulation

Dans la partie « Affichage des clusters », complétez la définition de K en indiquant la valeur du nombre de clusters choisie. Exécutez le code de sorte à afficher les clusters. L'affichage obtenu permet-il de dire que le nombre K a bien été choisi ? Si ce n'est pas le cas, reprendre le calcul en adaptant la valeur de K.

I.5 Validation du regroupement (clustering)

Les activités précédentes vous ont permis de définir un certain nombre de clusters. Pour vérifier que les regroupements effectués sont corrects, on peut comparer les classes obtenues par prédiction aux classes effectives. Le fichier finch_beaks_1975.csv contenant les données contient aussi l'espèces de chaque pinson.

Dans la partie « Validation du clustering » du script, vous trouverez les lignes de codes permettant de d'obtenir la liste labels. Dans cette liste, à l'indice i, vous trouverez la classe du i-eme individu. La structure est donc la même que dans la liste Labels_predits.

On peut alors déterminer le taux d'erreur de notre clustering

Manipulation

A la suite du code de la partie « Validation du clustering », tapez les lignes de codes permettant de calculer le taux d'erreur de clustering obtenu.

II Classification et algorithme des K plus proches voisins (K-nearest neighboors KNN)

II.1 Les données étudiées dans cette application

On reprend les données du §I. Les classes qui constituent ces données sont au nombre de deux. En effet les espèces dont nous avons les données sont :

— 'fortis' : suite à la sècheresse de 1973 , a adpaté son bec pour les graines à coque dure, bec court et robuste,

— 'scandens' : bec long et robuste. Idéal pour récupérer les fruits et les graines entre les épines de cactus.

Dans notre programme, ces espèces seront représentées par les 'classes' ou 'labels' :

— 0 pour 'fortis'

— 1 pour 'scandens'

Dans le script fourni, il y a quelques lignes permettant de séparer le jeu de données en deux parties :

— Une partie destinée à l'apprentissage,

— Une partie destinée aux tests

On aura donc à notre disposition les listes *donnees_apprentissage* et *labels_apprentissage*, et les listes *donnees_test* et *labels_test*.

II.2 Algorithme KNN avec scikit learn

Le module scikit learn fournit un ensemble d'éléments nécessaires à l'applcation de l'algorithme KNN. L'implémentation est différente de ce qui a été vu en cours. Le code ci-dessous, également présent dans script dans la partie « KNN avec scikit learn », présente son utilisation.

```
1 from sklearn import neighbors
2 objetKNN = neighbors.KNeighborsClassifier(n_neighbors=7) #crée un objet informatique implémentant l'algorithme KNN
3 objetKNN.fit(donnees_entrainement, labels_entrainement)
4 label_predit = objetKNN.predict(donnees_test       )          #Applique l'algorithme KNN pour la classification des données test
```

Manipulation

Exécuter le code présent dans script dans la partie « KNN avec scikit learn ». Observez le résultat, et vérifiez s'il est correct. Effectuer de nouveaux essais pour d'autres données test, vérifiez à chaque fois si le résultat obtenu est correct.

II.3 Validation par matrice de confusion

Afin de vérifier la qualité de notre programme de classification, nous souhaitons tester ses performances sur chacune des données du jeu *donnees_test* en comparant les labels prédits aux labels réels contenus dans la liste *labels_test*. Pour cela, nous pouvons utiliser une fonction du module scikit-learn permettant d'obtenir la matrice de confusion.

Manipulation

- A la suite du script, dans la « matrice de confusion », importez la fonction $confusion_matrix$ depuis le module *sklearn.metrics*.

La fonction $confusion\_matrix$ prend en argument :

— La liste des labels réels,

— La liste des labels prédits,

Et renvoie la matrice de confusion correpondante.

Manipulation

A la suite du script, dans la « matrice de confusion », utilisez la fonction $confusion\_matrix$, et affichez la matrice de confusion.
Justifiez le résultat obtenu, en faisant le lien notamment avec la disposition des données observez lors de leur affichage graphique .

II.4 Adaptation du code à un second jeu de données

On se propose de réutiliser notre programme sur un autre jeu de données. Ces données correspondent à des photos en niveaux de gris de visages de 40 personnes différentes. Les images, au format 64X64 ont été « aplaties » en concaténant les liste des lignes de pixels. L'importation des données est déjà codée dans le script $TP\_IA\_olivetti.py$.

II.4.1 Importation des données

Manipulation

Ouvrez le script *TP_IA_olivetti.py*, exécutez le code de la partie « importation des données »

Les listes contenant les données et les labels portent toujours les nom donnees et labels. Ainsi il sera simple de réutiliser le code déjà écrit.

Manipulation

A la suite de *TP_IA_olivetti.py*, utilisez le code permettant de d'afficher l'évolution de l'attribut *inertia_* en fonction du nombre de clusters. On pourra utiliser un nombre de clusters appartenant à l'ensemble [20, 40, 60, 80].

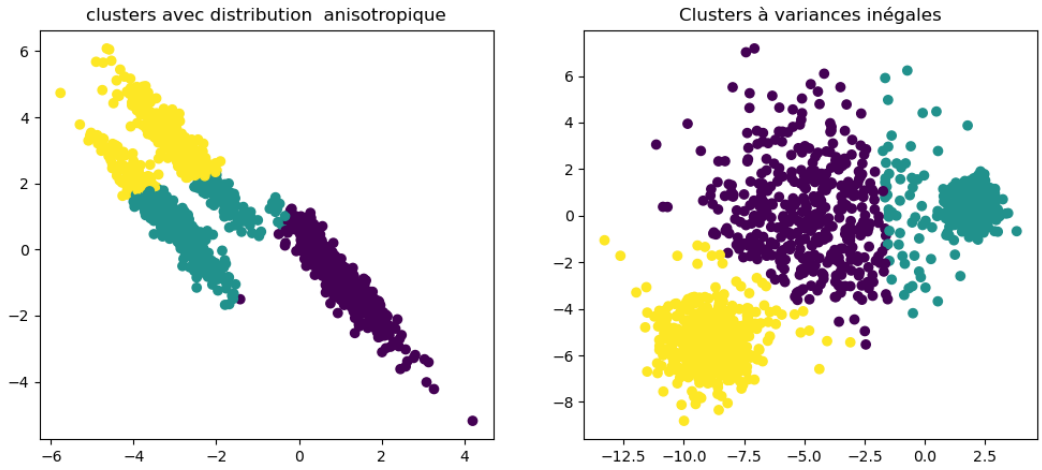
Sur le graphique obtenu, on voit qu'il est plus difficile d'appliquer la méthode Elbow sur ce jeu de données. En effet cette fois-ci il n'apparaît pas clairement de cassure sur la courbe obtenue. Cela est dû au fait que les classes sont proches les unes des autres, et les points au sein d'une classe sont disparates. Si on pouvait, comme précédemment, afficher un graphique de points utilisant les valeurs des données comme coordonnées, on aurait plus de difficulté à distinguer les différentes classes.

Les digits utilisés pour construire le jeu de données sont les chiffres 0 à 9. Pour affiche l'image correspondante à la données i, on peut utiliser la commande $plt.imshow(donnees[i].reshape((8,8)), cmap =$ "$gray$"). Dans le script *TP_IA_olivetti.py*, à la fin de la partie « importation des données », cette commande est déjà tapée, il suffit de la décommenter pour l'utiliser. Vous pouvez modifier la valeur de i pour afficher d'autres digits.

II.4.2 Clustering K-Means – Validation du clustering

A la suite de *TP_IA_olivetti.py*, dans la partie « Validation du clustering », effectuez un clustering avec le bon nombre de classes, puis comparez les labels prédits aux vrais labels. Normalement les données sont regroupées par groupe de 10 données, chaque groupe constituant une classe. Que pensez-vous du Clustering effectué par notre programme ?

L'algorithme de clustering K-Mean est très sensible au type de données. La référence du module scikit learn indique plusieurs type de distribution de données qui peuvent poser problème à l'algorithme K-Mean, comme indiqué sur la figure suivante. Il existe d'autres algorithmes de classification qui ont de meilleures performances sur des jeux de données aux distributions plus complexes. On peut citer notamment les algorithmes DB-SCAN, et partitionnement spectral.



II.4.3 Classification KNN - Prédiction et validation par matrice de confusion

Nous avons vu précédemment notre nouveau jeu de données a posé des difficultés à l'algorithme K-mean. Qu'en est-il pour l'algorithme de classification KNN ?

Manipulation

A la suite de *TP_IA_olivetti.py*, dans la partie « 4 - d) KNN avec scikit learn et validation par matrice de confusion», séparez labels et données en *donnees_apprentissage* et *labels_apprentissage*, et *donnees_test* et *labels_test*.
A la suite de *TP_IA_olivetti.py*, dans la partie 4 - d) , effectuez une prédiction sur les données test, puis faites calculer la matrice de confusion. Le résultat obtenu est-il satisfaisant ?

Le jeu de données utilisé contient peut d'observations pour chaque classe, en effet il n'y a que dix photos pour chaque visage. Le jeu de données étant scindé en deux parties, l'une pour l'apprentissage, l'autre pour les tests, dans in rapport 70% / 30%, cela fait peu de données, à la fois pour l'apprentissage et pour les tests.

II.4.4 Classification KNN - Validation croisée

Pour améliorer l'utilisation des données, on peut utiliser la validation croisée. Le principe consiste en un découpage des données en N sous-ensembles. L'un de ces sous-ensembles est utilisé comme données test, les N-1 autres sont utilisés comme données d'entrainement. Ensuite un effectue une permutation circulaires, les donnée ayant servi de test sont utilisées comme données d'entrainements, et l'un des N-1 paquets ayant servi pour l'entrainement est maintenant utilisé pour les tests. Cela est répété jusqu'à ce que chacun des N paquets de données aient tous été utilisés comme données de test. Il y a donc N permutations. A chaque permutation, on effectue un prédiction du label des données test, ce qui permet de comparer au vrai résultat et d'estimer la performance de l'algorithme.

Le module scikit-learn propose une classe effectuant cette opération : *GridSearchCV*.

La justesse de l'algorithme KNN dépend notamment du nombre K de voisins utilisé. La classe *GridSearchCV* permet également de tester différentes valeurs de K, et d'indiquer la valeur optimale du nombre K de voisins.

Le code de la partie « 4 - e) Validation croisée » du scripts *TP_IA_olivetti.py* contient le code suivant :

```
1  from sklearn import model_selection
2
3  from sklearn import neighbors
4
5  L_K = [1, 2, 3, 4, 5, 6, 7, 8, 9]
6
7  knnclass_cv = model_selection.GridSearchCV(neighbors.KNeighborsClassifier(),
8  {'n_neighbors': L_K}, cv=5,scoring='accuracy')knnclass_cv.fit(donnees, labels)
9
10 print("meilleure valeur de k : ", knnclass_cv.best_params_['n_neighbors'])
11
12 print("exactitude", knnclass_cv.best_score_)
```

La fonction GridSearchCV prend en argument :

— neighbors.KNeighborsClassifier() la fonction de classification

— un dictionnaire définissant les argument du la fonction de classification à régler pour optimiser la classification. Dans notre cas le seul paramètre est le nombre de voisins n_neighbors, et on fournit la liste des valeurs à tester L_K,

— cv est le nombre de paquets pour la répartition des données en sous-ensembles pour les permutations en données de test ou d'apprentissage,

— scoring est le paramètre à maximiser pour l'optimisation. Ici nous avons choisi 'accuracy', qui correspond à l'exactitude de la classification.

Manipulation

Exécutez cette partie du code, et prononcez-vous sur la qualité de notre programme de classification.