TP 1 - Introduction aux bibliothèques Python

Introduction

Le but de ce premier TP est de vous familiariser avec les bibliothèques Python de fouille données : scikit-learn (outils et algorithmes d'apprentissage automatique), numpy (pour les manipulations de nombres), scipy (pour les calculs scientifiques), pandas (pour manipuler les ensembles de données) et matplotlib (pour la visualisation).

1 Analyse descriptive des données

L'ensemble de données « Fruits » que vous allez étudier a été créé par Iain Murray de l'université d'Édimbourg. Il a été construit à partir de l'observation de quelques dizaines d'oranges, de mandarines, de citrons et de pommes de différentes variétés. Les mesures de chacun de ces fruits ont été enregistrées dans une table ; c'est un ensemble de taille réduite, établi à des fins pédagogiques.

Téléchargez le fichier `fruit_data_with_colors.txt` du site web du cours et mettez-le dans le répertoire de votre choix.

Vous pourrez vous appuyer sur le script `tp1.py` à compléter au fur et à mesure des questions.

1. Utilisez pandas pour lire le fichier et le convertir en *data frame*. Combien y a-t-il d'attributs ? Que représentent-ils ? Quel est leur type ? Quelle est la classe à prédire ? Combien y a-t-il d'instances dans le fichier ? Les classes sont-elles équilibrées quant au nombre d'instances ?
2. Examinez les données de manière graphique à l'aide d'un diagramme de dispersions (fonction plotting.scatter_matrix() de pandas) et d'histogrammes montrant la répartition selon un attribut et la classe.

2 Prétraitement

Suivant les méthodes de fouille de données considérées, il peut être utile d'appliquer différents prétraitements. On s'intéresse ici à deux grandes classes de filtres : la discrétisation et la normalisation.

1. Appliquez à partir des données originelles deux variantes de discrétisation : une première reposant sur des intervalles de même taille (*equal-intervaled* avec la fonction cut() de pandas) ou de mêmes effectifs (*equal-sized bins* avec qcut()). En affichant les historiques correspondant à chacune des deux configurations à l'aide de plot.bar(), comparez les histogrammes obtenus pour l'attribut « mass ».
2. Reprenez les données originelles. Quel est l'effet du filtre de de normalisation preprocessing.MinMaxScaler de scikit-learn ? Dans quel cas d'utilisation est-il préférable d'appeler la fonction transform() de cet objet plutôt que fit_transform() ?

3 Clustering [1]

Le clustering peut être vu comme une méthode non supervisée qui cherche à regrouper les instances qui se ressemblent, le plus souvent après avoir fixé au préalable un certain nombre de groupes (ou *clusters*) à construire.

Plusieurs algorithmes de clustering sont disponibles dans scikit-learn. Dans cette séance, deux méthodes de clustering seront testées : l'une classique est basée sur l'algorithme de partionnement des données *K-means* et est désignée par KMeans, l'autre est une méthode hiérarchique ascendante produisant un arbre donnant la proximité entre les instances et est appelée AgglomerativeClustering (pour ce dernier modèle, nous utiliserons en réalité des méthodes de bibliothèque scipy puisque nous nous contenterons d'un affichage).

1. Ouvrez le fichier de données en appliquant un filtre de normalisation. Appliquez KMeans sur ces données en fixant successivement le nombre de *clusters* à 3, 4 et 5. Examinez manuellement les résultats du clustering en traçant les instances dans une espace en 3 dimensions correspondant aux attributs « mass », « width » et « color_score ». Quelle(s) classe(s) représente chacun des *clusters* ?
2. Pour fixer le nombre de clusters, on propose d'utiliser la métrique R^2 calculée suivant les variances intra-classe et inter-classe. Cette métrique permet d'avoir une évaluation objective de l'homogénéité des groupes construits artificiellement, sans avoir étiqueté manuellement chaque instance. Un module R_square_clustering.py effectuant le calcul de R^2 vous est fourni sur le site du cours. Dessinez le graphe montrant l'évolution de cette métrique en faisant varier le nombre de *clusters* entre 2 et 10. Comment varie R^2 ? Quel est le nombre de clusters que vous retiendriez ?
3. Il est possible d'obtenir une évaluation quantitative plus complète du clustering lorsque l'on dispose des classes à obtenir dans les données (c'est un cas peu réaliste de l'apprentissage non supervisé mais ici nous sommes dans la situation luxueuse où chaque instance est étiquetée). Un module purity.py disponible sur le site du cours détermine un score de pureté en attribuant chaque cluster à la classe majoritaire. Utilisez-le pour tracer un graphe montrant l'évolution de ce score en fonction du nombre de *clusters* variant entre 2 et 10. Quel est le nombre de groupes que vous retiendriez suivant ce critère ?
4. Remplacez l'utilisation du calcul des k-moyennes par une classification ascendante hiérarchique en établissant le dendogramme des données normalisées à l'aide de la méthode de Ward et de la bibliothèque scipy.cluster.hierarchy. Commentez l'arbre obtenu et proposez une partition en sélectionnant une hauteur appropriée de coupure.

4 Classement

Un des intérêts de scikit-learn est d'incorporer un grand nombre d'algorithmes d'apprentissage automatique permettant de prédire la classe des individus. On se propose ici de tester quelques méthodes simples de classement selon deux protocoles d'évaluation.

1. Utilisez dans premier temps un découpage des données en $\frac{3}{4}$ entraînement et $\frac{1}{4}$ test à l'aide de la fonction train_test_split de scikit-learn. Comparez les performances des méthodes suivantes de classement :
   — classifieur élémentaire associant chaque instance à la classe majoritaire (DummyClassifier),

1. Appelé aussi « classification » dans le cours.

— méthode bayésienne naïve (GaussianNB),
— arbres de décision (DecisionTreeClassifier,
— la forêt d'arbres décisionnels (RandomForestClassifier)
— méthode de régression logistique (LogisticRegression).
Quel est le taux de classification obtenu pour chaque méthode ? Quelle est la classe pour laquelle la prédiction fait le plus d'erreurs ? Dans le cas de l'arbre de décision, visualisez avec la fonction tree.plot_tree et commentez l'arbre construit par le modèle.

2. Utilisez maintenant une validation croisée en 5 strates à l'aide de la fonction cross_val_score. Parmi les cinq méthodes précédemment testées, quelle est la plus performante ? Le classement des méthodes est-il le même que lors du protocole d'évaluation précédent ?
3. Les paramètres des méhodes de classement ont été jusqu'à présent laissés par défaut, mais ne conviennent pas nécessairement au type de données étudiées. Pour améliorer les performances du classifieur basé sur la régression logistique, utilisez l'optimiseur `GridSearchCV` réalisant une recherche par grille par validation croisée. Faites varier les paramètres C et penalty. Quel est le meilleur score obtenu parmi l'ensemble des valeurs de paramètres testées ? Quelles sont les meilleurs valeurs constatées sur le jeu de données étudiées ? Utilisez dans ce qui suit ces valeurs comme paramètres pour le modèle LogisticRegression [2].

5 Classement et discrétisation

On examine maintenant l'effet de la discrétisation sur la construction de modèles de classement.

1. Discrétisez les attributs numériques à l'aide d'intervalles de mêmes tailles. Appliquez les mêmes méthodes de classement que précédemment. Obtenez-vous les mêmes performances qu'avant la discrétisation ?
2. Mêmes questions que précédemment en utilisant cette fois-ci une discrétisation à l'aide d'intervalles de mêmes effectifs.

6 ACP et sélection de variables

L'analyse en composantes principales (ACP) peut être utilisée en méthode de prétraitements pour réduire le nombre de variables, ce qui est souvent très utile quand le nombre de dimensions des données est très grand (par exemple plusieurs dizaines). Dans cette section, on se propose d'étudier l'ACP sur les données et de comparer son influence sur les performances de classement par rapport à une sélection arbitraire ou par élimination récursive des variables.

1. Ouvrez le fichier de données en appliquant un filtre de normalisation puis une ACP. Quelles sont les valeurs propres des facteurs de l'ACP ? Dessinez le graphe montrant l'évolution des valeurs propres en fonction de leur rang. Quelles variables originelles sont corrélées avec chacun des nouveaux facteurs ? Tracez les instances dans le plan principal constitué des deux premiers facteurs. Les classes des fruits sont-ils bien séparés dans ce nouvel espace ?
2. Calculez les performances en validation croisée des quatres méthodes de classement utilisées dans la section 4 sur les données, en ne considérant non plus les quatre attributs initiaux mais les deux facteurs principaux de l'ACP. Obtenez-vous les mêmes performances qu'en utilisant deux attributs du fichier original normalisé (par exemple les deux premiers attributs) ?

2. NB : Il y a ici une erreur dans le protocole expérimental, puisque les paramètres devraient être optimisés sur un ensemble de développement, avant de comparer les méthodes sur un autre ensemble de test...

3. Dans la question précédente, les deux attributs ont été choisis au hasard. scikit-learn met à disposition plusieurs méthodes de sélection de variables. Utilisez le sélectionneur SelectKBest avec un critère bsé sur l'information mutuelle, pour ne retenir que les deux entrées. Quels sont les deux attributs jugés les plus importants par cette méthode ? Calculez les performances en validation croisée des quatre classifieurs utilisés sur les deux meilleures variables retenues et comparez les résultats précédents calculés sur les deux premiers facteurs de l'ACP.