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.

1. Vérifiez que

```
pip install -U scikit-learn  sklearn
pip install -U pandas
pip install -U graphviz

```

2. 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 ?

```
# read input text and put data inside a data frame
fruits = pd.read_table('fruit_data_with_colors.txt')
print(fruits.head())

# print nb of instances and features
print(fruits.shape)

# print feature types
print(fruits.dtypes)

# print balance between classes
print(fruits.groupby('fruit_name').size())

```

3. 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.

```
pd.plotting.scatter_matrix(X, c = y, marker = 'o', s=40, hist_kwds={'bins':15}, figsize=(9,9), cmap = cmap)

for attr in feature_names:
  pd.DataFrame({k: v for k, v in fruits.groupby('fruit_name')[attr]}).plot.hist(stacked=True)
  plt.suptitle(attr)
  plt.savefig('fruits_histogram_'+attr)

```

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 ».

```
pd.cut(fruits[attr],10).value_counts(sort=False).plot.bar()
plt.suptitle('Histogram for '+attr+' discretized with equal-intervaled bins')

```

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

1. Appelé aussi « classification » dans le cours.

les résultats du clustering en traçant les graphes comme suit. Quelle(s) classe(s) représente chacun des *clusters* ?

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans

# Plot clusters
lst_kmeans = [KMeans(n_clusters=n) for n in range(3,6)]
titles = [str(x)+' clusters' for x in range(3,6)]
fignum = 1
for kmeans in lst_kmeans:
    fig = plt.figure(fignum, figsize=(8, 6))
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
    kmeans.fit(X_norm)
    labels = kmeans.labels_
    ax.scatter(X['mass'], X['width'], X['color_score'],
               c=labels.astype(np.float), edgecolor='k')

    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel('mass')
    ax.set_ylabel('width')
    ax.set_zlabel('color_score')
    ax.set_title(titles[fignum - 1])
    ax.dist = 12
    plt.savefig('k-means_'+str(2+fignum)+'_clusters')
    fignum = fignum + 1
    plt.close(fig)

# Plot the ground truth
fig = plt.figure(fignum, figsize=(8, 6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
for label in fruits['fruit_name'].unique():
    ax.text3D(fruits.loc[fruits['fruit_name']==label].mass.mean(),
              fruits.loc[fruits['fruit_name']==label].width.mean(),
              fruits.loc[fruits['fruit_name']==label].color_score.mean(),
              label,
              horizontalalignment='center',
              bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))
ax.scatter(X['mass'], X['width'], X['color_score'], c=y, edgecolor='k')
ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('mass')
ax.set_ylabel('width')
ax.set_zlabel('color_score')
ax.set_title('Ground Truth')
ax.dist = 12
plt.savefig('k-means_ground_truth')
plt.close(fig)
plt.show()

```

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.

```
lst_k=range(2,11)
lst_rsq = []
for k in lst_k:
    est=KMeans(n_clusters=k)
    est.fit(X_norm)
    lst_rsq.append(r_square(X_norm, est.cluster_centers_,est.labels_,k))

fig = plt.figure()
plt.plot(lst_k, lst_rsq, 'bx-')
plt.xlabel('k')
plt.ylabel('RSQ')
plt.title('The Elbow Method showing the optimal k')

```

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 comme suit.

```
from scipy.cluster.hierarchy import dendrogram, linkage

lst_labels = map(lambda pair: pair[0]+str(pair[1]), zip(fruits['fruit_name'].values, fruits.index))
linkage_matrix = linkage(X_norm, 'ward')
fig = plt.figure()
dendrogram(
    linkage_matrix,
    color_threshold=0,
    labels=lst_labels,
)
plt.title('Hierarchical Clustering Dendrogram (Ward)')
plt.xlabel('sample index')
plt.ylabel('distance')
plt.tight_layout()

```

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),
   — méthode bayésienne naïve (GaussianNB),
   — arbres de décision (DecisionTreeClassifier,
   — 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 et commentez l'arbre construit par le modèle.

```
for clf,name_clf in zip(lst_classif,lst_classif_names):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy of '+name_clf+' classifier on training set: {:.2f}'
          .format(clf.score(X_train, y_train)))
    print('Accuracy of '+name_clf+' classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
    print(confusion_matrix(y_test, y_pred))

# print decision tree
import graphviz
dot_data = tree.export_graphviz(dectree, out_file=None,
                                feature_names=feature_names,
                                class_names=fruits['fruit_name'].unique(),
                                filled =True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("fruits")

```

2. Utilisez maintenant une validation croisée en 5 strates à l'aide de la fonction cross_val_score. Parmi les quatre méthodes précédemment testées, quelle est la plus performante ? Le classement des méthodes est-il le même que lors du protocole d'évaluation précédent ?

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 ?

```
prefix = 'eqintervaled_bins_'
for attr in feature_names:
    fruits[prefix+attr]=pd.cut(fruits[attr],10)
```

```
    # use pd.concat to join the new columns with your original dataframe
    fruits =pd.concat([fruits,pd.get_dummies(fruits[prefix+attr],prefix=prefix+attr)],axis=1)
    # now drop the original column (you don't need it anymore)
    fruits.drop(prefix+attr,axis=1, inplace=True)

feature_names_bins = filter(lambda x: x.startswith(prefix) and x.endswith(']'), list(fruits))
X_discret = fruits[feature_names_bins]
```

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 ?

```
from sklearn.decomposition import PCA

acp = PCA(svd_solver='full')
coord = acp.fit_transform(X_norm)
# explained variance scores
print(acp.explained_variance_ratio_)
# plot eigen values
n = np.size(X_norm, 0)
p = np.size(X_norm, 1)
eigval = float(n-1)/n*acp.explained_variance_
fig = plt.figure()
plt.plot(np.arange(1,p+1),eigval)
plt.title("Scree plot")
plt.ylabel("Eigen values")
plt.xlabel("Factor number")
plt.savefig('acp_eigen_values')
plt.close(fig)

# print eigen vectors
print(acp.components_)
# lines: factors
# columns: variables

# print correlations between factors and original variables
sqrt_eigval = np.sqrt(eigval)
corvar = np.zeros((p,p))
for k in range(p):
    corvar[:, k] = acp.components_[k,:] * sqrt_eigval[k]
```

```
print(corvar)
# lines:  variables
# columns: factors

# plot instances on the first plan (first 2 factors)
fig, axes = plt.subplots(figsize=(12,12))
axes.set_xlim(-1,1)
axes.set_ylim(-1,1)
for i in range(n):
    plt.annotate(y.values[i],(coord[i,0], coord[i,1]))
plt.plot([-1,1],[0,0], color='silver', linestyle='-',linewidth=1)
plt.plot([0,0],[-1,1], color='silver', linestyle='-',linewidth=1)
plt.savefig('acp_instances_1st_plan')
plt.close(fig)
```

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) ?
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 comme suit RFECV pour réaliser une élimination récursive des variables à l'aide d'un estimateur évaluant l'importance de chaque attribut. Quels sont les deux attributs jugés les plus importants par cette méthode ?

```
from sklearn.feature_selection import RFECV
from sklearn.svm import SVR
estimator = SVR(kernel="linear")
selector = RFECV(estimator=estimator, cv=5)
selector.fit(X_norm, y)
print("Optimal number of features: %d" % selector.n_features_)
print(selector.ranking_)
```

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.