PCA, ICA, NMF

TRAVAUX PRATIQUES

- SÉPARATION DE SOURCES AUDIO PAR ICA -

On se propose dans cette première partie d'utiliser l'ICA pour séparer les deux sources audio (monophoniques) $s_k(n)$, $k \in \{1, 2\}$, qui composent un mélange stéréophonique $x_l(n)$, $l \in \{1, 2\}$, dit *linéaire instantané*, c'est-à-dire obtenu par combinaisons linéaires des sources monophoniques suivant :

$$x_l(n) = \sum_{k=1}^{2} a_{l,k} s_k(n) \; ;$$

les coefficients $a_{l,k}$ étant les gains de mixage, autrement dit les intensités avec lesquelles les sources contribuent à chaque mélange.

1. Étudier le script `ica_audio.py` et l'utiliser pour générer le mélange stéréo. Prendre le temps d'écouter les différents signaux.
2. Utiliser l'ICA pour extraire les sources originales à partir du mélange. On exploitera pour cela `sklearn.decomposition.FastICA`.
3. Créer les fichiers audio pour chaque source extraite et les écouter.

```
# ica_audio.py

# Author: Slim Essid

import pylab as pl
import numpy as np
import wave
from sklearn import decomposition

# -- Read audio data -----------------------------------------------------

def wavread(filename):
    wf = wave.open(filename)
    s = np.fromstring(wf.readframes(wf.getnframes()), dtype=np.int16)
    wf.close()
    return s

def wavwrite(filename, s, nchannels=1):
    wf = wave.open(filename,'wb')
    wf.setnchannels(nchannels)
    wf.setsampwidth(2)
    wf.setframerate(32000)
    s = s/max(abs(s))*2**14 # normalise arbitrarily
    wf.writeframes(s.astype(np.int16).tostring())
```

```
    wf.close()

s1 = wavread('./snd/es02.wav')
s2 = wavread('./snd/si01.wav')

# -- Create stereo mix ----------------------------------------------------

nsmp = min(len(s1), len(s2))
s1 = s1[:nsmp]
s2 = s2[:nsmp]
S = np.c_[s1, s2]
# Mixing matrix
A = np.array([ [0.6, 0.5],
               [0.4, 0.5]
               ])
# Do the mix
X = S.dot(A)
# Write mix file
x = np.zeros(2*nsmp)
x[::2] = X[:,0]
x[1::2] = X[:,1]
wavwrite('./snd/mix.wav', x, nchannels=2)

# -- ICA source separation ------------------------------------------------
...

# Visualise sources
...

# Write audio sources
...
```

- EXTRACTION DE CARACTÉRISTIQUES DE VISAGES -

On étudie ici l'utilisation des techniques de décomposition par PCA et NMF pour l'extraction de caractéristiques utiles à la reconnaissance automatique de visages.

1. Étudier et tester le script `pca_nmf_faces.py`. Analyser le type de décomposition obtenu par NMF en comparaison avec celui obtenu par PCA.
2. Modifier le script pour réaliser une évaluation des performances d'un système de reconnaissance automatique de visages utilisant les caractéristiques extraites par PCA, comparées à celles obtenues par un système exploitant les caractéristiques extraites par NMF. On pourra utiliser la LDA pour la classification. On effectuera l'évaluation par validation croisée. On observera l'évolution des scores en faisant varier le nombre de composantes utilisé dans les décompositions.

```
# pca_nmf_faces.py

# Authors: Vlad Niculae, Alexandre Gramfort, Slim Essid
# License: BSD

from time import time
from numpy.random import RandomState
```

```
import pylab as pl
import numpy as np

from sklearn.datasets import fetch_olivetti_faces
from sklearn import decomposition

# -- Prepare data and define utility functions ----------------------------

n_row, n_col = 2, 5
n_components = n_row * n_col
image_shape = (64, 64)
rng = RandomState(0)

# Load faces data
dataset = fetch_olivetti_faces(shuffle=True, random_state=rng)
faces = dataset.data

n_samples, n_features = faces.shape

# global centering
faces_centered = faces - faces.mean(axis=0, dtype=np.float64)

print "Dataset consists of %d faces" % n_samples

def plot_gallery(title, images):
    pl.figure(figsize=(2. * n_col, 2.26 * n_row))
    pl.suptitle(title, size=16)
    for i, comp in enumerate(images):
        pl.subplot(n_row, n_col, i + 1)

        comp = comp.reshape(image_shape)
        vmax = comp.max()
        vmin = comp.min()
        dmy = np.nonzero(comp<0)
        if len(dmy[0])>0:
            yz, xz = dmy
        comp[comp<0] = 0

        pl.imshow(comp, cmap=pl.cm.gray, vmax=vmax, vmin=vmin)
        #print "vmax: %f, vmin: %f" % (vmax, vmin)
        #print comp

        if len(dmy[0])>0:
            pl.plot( xz, yz, 'r,', hold=True)
            print len(dmy[0]), "negative-valued pixels"

        pl.xticks(())
        pl.yticks(())

    pl.subplots_adjust(0.01, 0.05, 0.99, 0.93, 0.04, 0.)

# Plot a sample of the input data
plot_gallery("First centered Olivetti faces", faces_centered[:n_components])

# -- Decomposition methods ------------------------------------------------
```

```
# List of the different estimators and whether to center the data

estimators = [
    ('pca', 'Eigenfaces - PCA',
     decomposition.PCA(n_components=n_components, whiten=True),
     True),

    ('nmf', 'Non-negative components - NMF',
     decomposition.NMF(n_components=n_components, init=0, tol=1e-6,
                       max_iter=2000),
     False)
]

# -- Transform and classify ----------------------------------------------------

labels = dataset.target
X = faces
X_ = faces_centered

for shortname, name, estimator, center in estimators:
    #if shortname != 'nmf': continue
    print "Extracting the top %d %s..." % (n_components, name)
    t0 = time()

    data = X
    if center:
        data = X_

    data = estimator.fit_transform(data)

    train_time = (time() - t0)
    print "done in %0.3fs" % train_time

    components_ = estimator.components_

    plot_gallery('%s - Train time %.1fs' % (name, train_time),
                 components_[:n_components])
```

- EXTRACTION DE THÈMES À PARTIR DE TEXTES -

Il s'agit dans cette partie de tester l'utilisation de la NMF pour l'extraction de thèmes à partir d'un corpus de textes ; l'idée principale étant d'interpréter chaque composante NMF extraite comme étant associée à un thème.

Étudier et tester le script suivant (introduit sur `http://scikit-learn.org/stable/auto_examples/applications/topics_extraction_with_nmf.html`):

```
# Author: Olivier Grisel <olivier.grisel@ensta.org>
# License: Simplified BSD

from time import time
from sklearn.feature_extraction import text
from sklearn import decomposition
from sklearn import datasets
```

```
n_samples = 1000
n_features = 1000
n_topics = 10
n_top_words = 20

# Load the 20 newsgroups dataset and vectorize it using the most common word
# frequency with TF-IDF weighting (without top 5% stop words)

t0 = time()
print "Loading dataset and extracting TF-IDF features..."
dataset = datasets.fetch_20newsgroups(shuffle=True, random_state=1)

vectorizer = text.CountVectorizer(max_df=0.95, max_features=n_features)
counts = vectorizer.fit_transform(dataset.data[:n_samples])
tfidf = text.TfidfTransformer().fit_transform(counts)
print "done in %0.3fs." % (time() - t0)

# Fit the NMF model
print "Fitting the NMF model on with n_samples=%d and n_features=%d..." % (
    n_samples, n_features)
nmf = decomposition.NMF(n_components=n_topics).fit(tfidf)
print "done in %0.3fs." % (time() - t0)

# Inverse the vectorizer vocabulary to be able
feature_names = vectorizer.get_feature_names()

for topic_idx, topic in enumerate(nmf.components_):
    print "Topic #%d:" % topic_idx
    print " ".join([feature_names[i]
                    for i in topic.argsort()[:-n_top_words - 1:-1]])
    print
```

Si possible appliquer la même technique aux données de votre projet.