Separación de imágenes con técnicas de PCA e ICA

Pablo de Castro

January 2, 2015

1 Introducción

El objetivo de esta práctica es la aplicación de las técnicas de Análisis por Componentes Principales (*PCA, del inglés*) y de Análisis por Componentes Independientes (*ICA, del inglés*), para separas tres imágenes que se encuetran mezcladas para extraer las componentes y la matríz de mezcla.

2 Fundamento Teórico

2.1 Problema Propuesto

Se han provisto tres imágenes de 512×512 píxeles cada una conteniendo una mezcla lineal de tres imágenes originales. Como se mostrará en la siguiente sección, dos de las imágenes originales pueden distinguirse claramente (una mujer joven y un gato). Sin embargo, la tercera imágen no es claramente identificable por lo que está oculta.

Las imágenes a utilizar no contienen ningún componente de ruido. El modelo de mezcla utilizado se puede describir como:

$$\mathbf{Y} = \mathbf{AX}$$

en donde $\mathbf{Y}$ son las tres imágenes mezcladas (dimensión 3×512^2), $\mathbf{X}$ las tres imágenes originales (dimensión 3×512^2) y $\mathbf{A}$ es la matrix de mezcla (dimensión 3×3). Se desconoce tanto la matriz de mezcla $\mathbf{A}$ como las componentes originales $\mathbf{X}$. Este problema, en general, carece de solución determinada. Sin embargo, en este trabajo se utilizaran las técnicas de Análisis de Componentes Principales y Análisis por Componentes Independientes para estimar ambos elementos bajo ciertas condiciones impuestas. La aplicación de estos métodos también tiene como objetivo la identificación de la tercera imagen original que se encuentra oculta.

2.2 Análisis de Componentes Principales

El Análisis de Componentes Principales (PCA, del inglés), es un prodedimiento estadístico para transformar un conjunto de observaciones mediante una transformación ortogonal a otro conjunto de componentes que esten descorrelacionadas, denominadas componentes principales.

Esta tranformación se define de tal modo que cada componente tenga la máxima varianza posible bajo la condición de que la transfromación se ortogonal a las componentes anteriores. El procedimiento permite, de forma no paramétrica, la compresión o representación en un sistema de coordenadas que explique mejor la varianza de los datos.

Supóngase que se tiene una matríz de datos $\mathbf{Y}$ (con dimensiones (nº componentes) $\times$ (nº muestras)), el objetivo es encontrar un tranformación lineal ortonormal $\mathbf{P}$ que genere una nueva representación de los datos $\mathbf{Z}$ en los que las componentes esten descorrelacionadas:

$$\mathbf{Z} = \mathbf{PY}$$

La correlación entre las componentes se puede evaluar con el valor de los elementos no diagonales de la matriz de covarianza $\mathbf{C_Y}$, que para la matríz de datos $\mathbf{Y}$ una vez centralizada (restando la media de cada componente) se define (sin tener en cuenta los factores constantes) como:

$$\mathbf{C_Y} = \mathbf{Y}\mathbf{Y^T}$$

El objetivo es que en la representación $\mathbf{Z}$ las componentes esten descorrelacionadas, es decir que los elementos no diagonales de la matríz sean nulos [1]. La matrix de covarianza $\mathbf{C_Z}$ para matríz de datos $\mathbf{Z}$ se define como:

$$\mathbf{C_Z} = \mathbf{Z}\mathbf{Z^T} = \mathbf{P}\mathbf{Y}(\mathbf{P}\mathbf{Y})^T = \mathbf{P}\mathbf{Y}\mathbf{Y^T}\mathbf{P}^T = \mathbf{P}\mathbf{C_Y}\mathbf{P}^T$$

Dado que la matríz $\mathbf{P}$ es ortogonal, su tranversa es igual a su inversa $\mathbf{P^T} = \mathbf{P^{-1}}$, por lo que $\mathbf{P}$ es la matríz que diagonaliza la matríz de covarianza de los datos $\mathbf{Y}$. En resumen, el problema de encontrar los componentes principales se reduce a la búsqueda de los autovectores de $\mathbf{C_Y}$, para lo cuál se pueden utiliza diversos métodos (i.e. EVD y SVD) . La primera componente correspondera con el autovector del autovalor mayor en valor absoluto, la segunda componente al segundo mayor y así sucesivamente.

Este método es útil para la compresión y representación de datos, ya que permite cambiar a una base en la que las componentes estén descorrelacionadas mediante una tranformación lineal. Por lo tanto, cuando se aplique sobre los datos de este problema se espera que cada componente principal corresponda aproximadamente a una imágen inicial.

2.3 Análisis de Componentes Independientes

El Análisis por Componentes Independientes (*ICA, del inglés*) es un método computacional para separar componentes estadísticamente independientes presentes en un conjunto de datos. Este proceso también se conoce como separación de fuentes independientes y es una posible solución para el problema propuesto en esta práctica.

Para la aplicación este procedimiento, se supone que las fuentes (i.e. componentes de $\mathbf{X}$) son estadísticamente independientes, que tienen distribuciones no gaussianas (aunque una sola componente si que puede ser gaussiana) y que a matriz de mezcla $\mathbf{A}$ es invertible. Bajo dichas condiciones el sistema es identificable y sus componentes independientes pueden determinarse salvo permutaciones y factores multiplicativos.

Existen diversos métodos para la obtención de las componentes principales, incluyendo la maximización de la no-gaussianidad (i.e. kurtosis o negentropía), máxima verosimilitud, mínima información nula, métodos tensoriales y de decorrelación no lineal. Muchos de los métodos mencionados requiren la aplicación de PCA como un primer paso para la preparación de los datos, que se denomina blanqueado (*withening*). El objetivo último de todos los métodos es la busqueda de una matríz de componentes $\mathbf{W}$, que es la inversa de la matriz de mezcla $\mathbf{A}$, de modo que las componentes de $\mathbf{X}$ sean estadísticamente independientes.

3 Desarrollo

3.1 Software Utilizado

El análisis de las imágenes en esta práctica se ha realizado en el entorno de computación interactivo *IPython Notebook* [2]. Este mismo documento puede ser visualizado o descargado en formato digital en http://nbviewer.ipython.org/github/pablodecm/SepFuentes/blob/master/SepFuentes.ipynb.

Se han utilizado las bibliotecas de software *open source*: *numpy* [**?**] para la manipulación de arrays y matrices, *matplotlib* [3] para la visualización de imágenes y *scikit-learn* [4] para las implementaciones de PCA e ICA utilizadas.

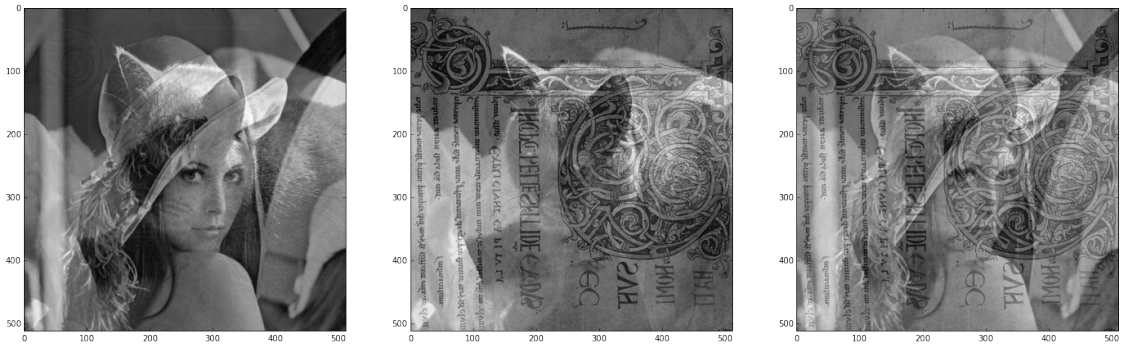
A continuación, se importan todos los modulos necesarios de la bibliotecas que serán de utilidad en los apartados posteriores.

```
In [1]: %matplotlib inline
        import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn import decomposition
```

3.2 Visualización de las Imágenes Provistas

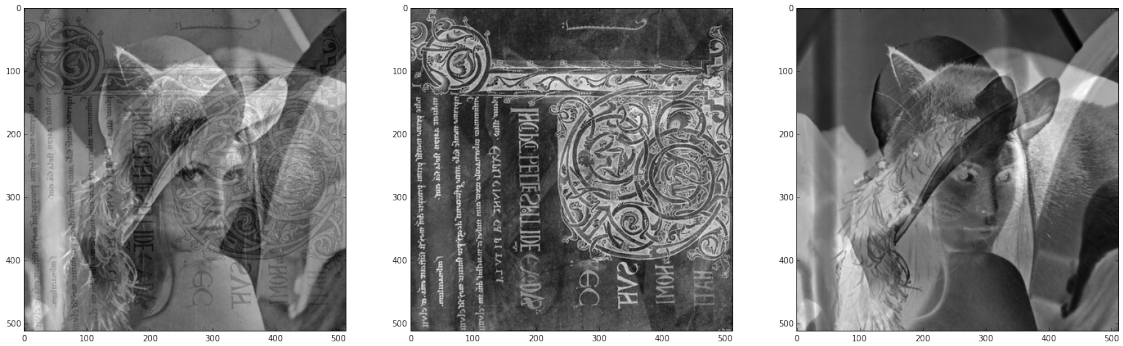
```
In [2]: img_1 = np.genfromtxt('imagen_mezclada_uno.dat', dtype='float64')
        img_2 = np.genfromtxt('imagen_mexclada_dos.dat', dtype='float64')
        img_3 = np.genfromtxt('imagen_mexclada_tres.dat', dtype='float64')

        f, (ax1,ax2,ax3) = plt.subplots(1,3)
        f.set_size_inches((24,40))
        ax1.imshow(img_1, cmap=plt.cm.gray)
        ax2.imshow(img_2, cmap=plt.cm.gray)
        ax3.imshow(img_3, cmap=plt.cm.gray)

Out[2]: <matplotlib.image.AxesImage at 0x107e78c50>
```



```
In [3]: # each image is flattened and all three are stacked together
        X = np.vstack((img_1.flatten(),img_2.flatten(),img_3.flatten()))
        X.shape

Out[3]: (3, 262144)

In [4]: pca = decomposition.PCA(n_components=3)
        img_pca= pca.fit_transform(X.T)
        f, (ax1,ax2,ax3) = plt.subplots(1,3)

        print X.T.shape
        print "Explained variance:\n", pca.explained_variance_
        print "Explained variance ratio:\n", pca.explained_variance_ratio_
        print "Inverse of Mixing Matrix:\n", pca.components_
        print "Mixing Matrix:\n", np.linalg.inv(pca.components_)
        f.set_size_inches((24,40))
        ax1.imshow(img_pca[:,0].reshape((512,512)), cmap=plt.cm.gray)
        ax2.imshow(img_pca[:,1].reshape((512,512)), cmap=plt.cm.gray)
        ax3.imshow(img_pca[:,2].reshape((512,512)), cmap=plt.cm.gray)

(262144, 3)
Explained variance:
[ 5.72141859  0.90953297  0.189357  ]
Explained variance ratio:
```

```
[ 0.83887973  0.13335657  0.0277637 ]
Inverse of Mixing Matrix:
[[ 0.65909172  0.14263253  0.73841321]
 [ 0.74315636 -0.27417719 -0.61036505]
 [ 0.11539815  0.95104302 -0.28670619]]
Mixing Matrix:
[[ 0.65909172  0.74315636  0.11539815]
 [ 0.14263253 -0.27417719  0.95104302]
 [ 0.73841321 -0.61036505 -0.28670619]]
```

Out[4]: <matplotlib.image.AxesImage at 0x10809b5d0>



In [5]:
```
np.random.seed(20)

ica = decomposition.FastICA(n_components=3)
img_ica= ica.fit_transform(X.T)

print "Mixing Matrix:\n",ica.mixing_
print "Unmixing Matrix:\n",ica.components_

f, (ax1,ax2,ax3) = plt.subplots(1,3)
f.set_size_inches((24,40))
ax1.imshow(img_ica[:,0].reshape((512,512)), cmap=plt.cm.gray)
ax2.imshow(img_ica[:,1].reshape((512,512)), cmap=plt.cm.gray)
ax3.imshow(img_ica[:,2].reshape((512,512)), cmap=plt.cm.gray)
```

```
Mixing Matrix:
[[-567.29852888  675.60652116  -74.83530837]
 [-241.26704961  -37.39529646  183.64472666]
 [-419.33633585  630.17696537  581.14653985]]
Unmixing Matrix:
[[-0.00103173 -0.00330088  0.00091023]
 [ 0.00047438 -0.00271003  0.00091747]
 [-0.00125886  0.00055686  0.00138266]]
```

Out[5]: <matplotlib.image.AxesImage at 0x10a809ad0>

```
In [6]: np.dot(X,X.T)
        np.dot(img_pca.T,img_pca)

Out[6]: array([[  1.49983556e+06,  -1.52587631e-09,  -3.71935371e-10],
               [ -1.52587631e-09,   2.38428612e+05,   2.27581509e-10],
               [ -3.71935371e-10,   2.27581509e-10,   4.96388021e+04]])

In [7]: X

Out[7]: array([[-0.20441684, -0.21622352, -0.22803021, ..., -1.6040118 ,
                -1.5789903 , -1.4360781 ],
               [-0.55223228, -0.50717805, -0.46212381, ..., -0.2389776 ,
                -0.20706421, -0.19486738],
               [-0.04551583,  0.09989691,  0.24530965, ..., -0.96726079,
                -0.82407212, -0.73492105]])

In []:
```

References

[1] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[2] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.