Raul E. Lopez Briega (https://relopezbriega.github.io/)

Matemáticas, análisis de datos y python

- Atom (/atom.xml)

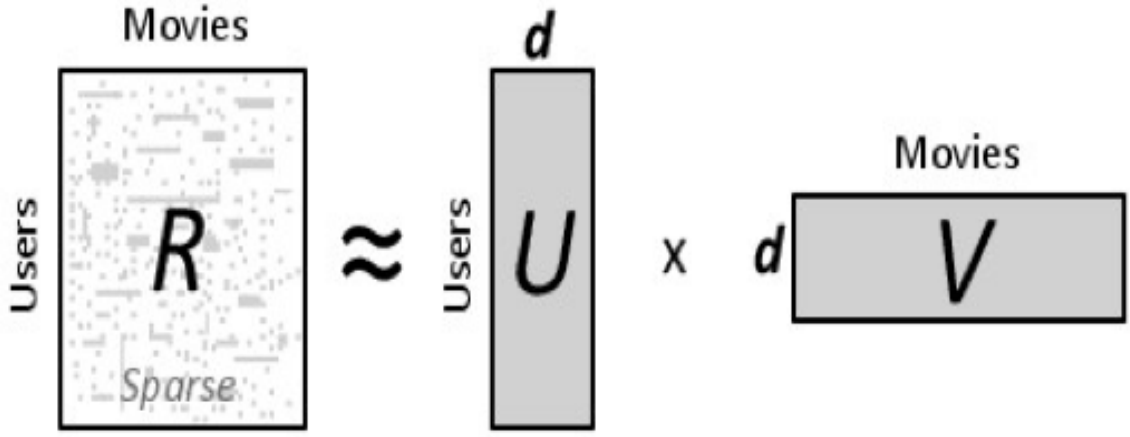
Search

» Atom

- About (/pages/acerca-de-mi.html)
- Home Page (/index.html)
- Archives (/archives.html)
- Mi otro Blog (http://relopezbriega.com.ar/)
- IAAR Book (https://iaarbook.github.io/)
- 2048 (/2048/)
- Contacto (/contacto/)

Factorización de Matrices con Python

Tue 13 September 2016



Introducción

Cuando trabajamos en problemas de Machine Learning (https://relopezbriega.github.io/tag/machine-learning.html), muchas veces nos vamos a encontrar con enormes conjuntos de datos (https://es.wikipedia.org/wiki/Conjunto_de_datos), con cientos o miles de características o *features*. Una forma simple de reducir las dimensiones de estas características es aplicar alguna técnica de **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)**. La **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** tiene enormes aplicaciones en todo tipo de problemas relacionados a la inteligencia artificial (https://es.wikipedia.org/wiki/Inteligencia_artificial), ya que la reducción de dimensionalidad (https://en.wikipedia.org/wiki/Dimensionality_reduction) es la esencia de la *cognición (https://es.wikipedia.org/wiki/Cognici%C3%B3n)*.

Asimismo, la **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** es también un tema unificador dentro del álgebra lineal numérica (https://relopezbriega.github.io/tag/algebra.html). Una amplia variedad de algoritmos (https://es.wikipedia.org/wiki/Algoritmo) se han desarrollado a lo largo de muchas décadas, proporcionando una plataforma numérica para operaciones de matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) tales como, la resolución de sistemas de ecuaciones lineales (https://es.wikipedia.org/wiki/Sistema_de_ecuaciones_lineales), la descomposición espectral

(https://es.wikipedia.org/wiki/Teorema_de_descomposici%C3%B3n_espectral), y la identificación de subespacios vectoriales (https://es.wikipedia.org/wiki/Subespacio_vectorial). Algunos de estos algoritmos también han demostrado ser de utilidad en problemas de análisis estadístico de datos (https://relopezbriega.github.io/tag/estadistica.html), como es el caso de la descomposición en valores singulares (https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares) o SVD (https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares), por sus siglas en inglés, que es la base del análisis de componentes principales (https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales) o PCA (https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales), que es una técnica muy utilizada para reducir el tamaño de los datos. Muchas investigaciones actuales en Machine Learning (https://relopezbriega.github.io/tag/machine-learning.html) han centrados sus esfuerzos en el uso de la **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** para mejorar el rendimiento de los sistemas de aprendizaje. Principalmente en el estudio de la factorización de matrices no negativas (NMF) (https://en.wikipedia.org/wiki/Non-negative_matrix_factorization), la cual se centra en el análisis de matrices de datos (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) cuyos elementos son positivos (no negativos), una ocurrencia muy común en los conjuntos de datos (https://es.wikipedia.org/wiki/Conjunto_de_datos) derivados de textos e imágenes.

¿Qué es la factorización de matrices?

En matemáticas (https://es.wikipedia.org/wiki/Matem%C3%A1ticas), la factorización (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n) es una técnica que consiste en la descomposición de una expresión matemática (que puede ser un número, una matriz (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)), un tensor (https://es.wikipedia.org/wiki/C%C3%A1lculo_tensorial), etc.) en forma de producto. Existen distintos métodos de factorización, dependiendo de los objetos matemáticos estudiados; el objetivo es simplificar una expresión o reescribirla en términos de «bloques fundamentales», que reciben el nombre de *factores*. Así por ejemplo, el número 6 se puede descomponer en el producto de 3 y 2. Si extendemos este concepto al mundo de las matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)), entonces podemos decir que la **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** consiste en encontrar dos o más matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) de manera tal que cuando se multipliquen nos devuelvan la matriz (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) original. Por ejemplo:

$$\begin{pmatrix} 3 & 4 & 5 \\ 6 & 8 & 10 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$$

Los métodos de **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** han ganado popularidad últimamente al haber sido aplicados con éxito en sistemas de recomendación (https://es.wikipedia.org/wiki/Sistema_de_recomendaci%C3%B3n) para descubrir las características latentes que subyacen a las interacciones entre dos tipos de entidades, como por ejemplo usuarios y películas. El algoritmo que ganó el desafío de Netflix (https://netflixprize.com/) fue un sistema basado en métodos de **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)**.

Factorización de matrices en sistemas de ecuaciones lineales

Dos de las **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** más utilizadas y que tal vez mucha gente las haya escuchado nombrar alguna vez son la factorización LU (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_LU) y la factorización QR (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_QR); las cuales se utilizan a menudo para resolver sistemas de ecuaciones lineales (https://es.wikipedia.org/wiki/Sistema_de_ecuaciones_lineales).

Factorización LU

$$A = LU$$

En álgebra lineal (https://relopezbriega.github.io/tag/algebra.html), la factorización o descomposición LU (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_LU) (del inglés Lower-Upper) es una forma de factorización (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n) de una matriz (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) como el producto de una matriz triangular

(https://es.wikipedia.org/wiki/Matriz_triangular) inferior y una superior. La factorización LU (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_LU) expresa el método de Gauss (https://es.wikipedia.org/wiki/M%C3%A9todo_de_Gauss) en forma matricial. Así por ejemplo, tenemos que $PA = LU$ donde P es una matriz de permutación (https://es.wikipedia.org/wiki/Matriz_de_permutaci%C3%B3n). Una condición suficiente para que exista la factorización es que la matriz A sea una matriz no singular (https://es.wikipedia.org/wiki/Matriz_no_singular).

En Python (https://www.python.org/) podemos encontrar la descomposición LU (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_LU) con la ayuda de SciPy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.lu_factor.html) de la siguiente forma:

In [1]:

Ver Código

In [2]:

```
# Ejemplo factorización LU
A = np.array([[7, 3, -1, 2]
             ,[3, 8, 1, -4]
             ,[-1, 1, 4, -1]
             ,[2, -4, -1, 6]])
P, L, U = la.lu(A)
```

In [3]:

```
# Matriz A
A
```

Out[3]:

```
array([[ 7,  3, -1,  2],
       [ 3,  8,  1, -4],
       [-1,  1,  4, -1],
       [ 2, -4, -1,  6]])
```

In [4]:

```
# Matriz de permutación
P
```

Out[4]:

```
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

In [5]:

```
# Matriz triangular inferior
L
```

Out[5]:

```
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.42857143,  1.        ,  0.        ,  0.        ],
       [-0.14285714,  0.21276596,  1.        ,  0.        ],
       [ 0.28571429, -0.72340426,  0.08982036,  1.        ]])
```

In [6]:

```
# Matriz triangular superior
U
```

Out[6]:

```
array([[ 7.        ,  3.        , -1.        ,  2.        ],
       [ 0.        ,  6.71428571,  1.42857143, -4.85714286],
       [ 0.        ,  0.        ,  3.55319149,  0.31914894],
       [ 0.        ,  0.        ,  0.        ,  1.88622754]])
```

In [7]:

```
# A = LU
L @ U
```

Out[7]:

```
array([[ 7.,  3., -1.,  2.],
       [ 3.,  8.,  1., -4.],
       [-1.,  1.,  4., -1.],
       [ 2., -4., -1.,  6.]])
```

Factorización QR

$A = QR$

La descomposición o factorización QR (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_QR) consiste en la descomposición de una matriz (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) como producto de una matriz ortogonal (https://es.wikipedia.org/wiki/Matriz_ortogonal) ($Q^T \cdot Q = I$) por una matriz triangular superior (https://es.wikipedia.org/wiki/Matriz_triangular). la factorización QR (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_QR) es ampliamente utilizada en las finanzas cuantitativas como base para la solución del problema de los mínimos cuadrados lineales (https://es.wikipedia.org/wiki/M%C3%ADnimos_cuadrados_ordinarios), que a su vez se utiliza para el análisis de regresión estadística (https://es.wikipedia.org/wiki/Regresi%C3%B3n_lineal).

En Python (https://www.python.org/) podemos encontrar la descomposición QR (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_LU) con la ayuda de SciPy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.lu_factor.html) de la siguiente forma:

In [8]:

```
# Ejemplo factorización QR
A = np.array([[12, -51, 4],
              [6, 167, -68],
              [-4, 24, -41]])
Q, R = la.qr(A)
```

In [9]:

```
# Matriz A
A
```

Out[9]:

```
array([[ 12, -51,   4],
       [  6, 167, -68],
       [ -4,  24, -41]])
```

In [10]:

```
# Matriz ortogonal Q
Q
```

Out[10]:

```
array([[-0.85714286,  0.39428571,  0.33142857],
       [-0.42857143, -0.90285714, -0.03428571],
       [ 0.28571429, -0.17142857,  0.94285714]])
```

In [11]:

```
# Matriz triangular superior R
R
```

Out[11]:

```
array([[ -14.,  -21.,   14.],
       [   0., -175.,   70.],
       [   0.,    0.,  -35.]])
```

In [12]:

```
# A = QR
Q @ R
```

Out[12]:

```
array([[  12.,  -51.,    4.],
       [   6.,  167.,  -68.],
       [  -4.,   24.,  -41.]])
```

Matrices dispersas y no negativas

En muchas ocasiones cuando trabajamos con matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) para representar el mundo físico, nos vamos a encontrar con que muchas de estas matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) están dominadas por mayoría de elementos que son cero. Este tipo de matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) son llamadas matrices dispersas (https://es.wikipedia.org/wiki/Matriz_dispersa), es decir, matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) de gran tamaño en que la mayor parte de sus elementos son cero. Como sería ineficiente almacenar en la memoria de la computadora todos los elementos en cero, en las matrices dispersas (https://es.wikipedia.org/wiki/Matriz_dispersa) solo vamos a almacenar los valores que no son cero y alguna información adicional acerca de su ubicación.

Asimismo muchos datos del mundo real son no negativos y los componentes ocultos correspondientes tienen un sentido físico solamente cuando son no negativos. En la práctica, ambas características, ser dispersas y no negativas son a menudo deseable o necesario cuando los componentes subyacentes tienen una interpretación física. Por ejemplo, en el procesamiento de imágenes (https://en.wikipedia.org/wiki/Image_processing) y visión artificial (https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial), las variables involucradas y los parámetros pueden corresponder a píxeles, y la factorización de matrices dispersas y no negativas (https://en.wikipedia.org/wiki/Non-negative_matrix_factorization) está relacionada con la extracción de las partes más relevantes de las imágenes. La representación *dispersa* de los datos por un número limitado de componentes es un problema importante en la investigación. En Machine Learning (https://relopezbriega.github.io/tag/machine-learning.html), las matrices dispersas (https://es.wikipedia.org/wiki/Matriz_dispersa) está estrechamente relacionada con la selección de atributos (https://relopezbriega.github.io/blog/2016/04/15/ejemplo-de-machine-learning-con-python-seleccion-de-atributos/) y ciertas generalizaciones en algoritmos de aprendizaje; mientras que la *no negatividad* se relaciona a las distribuciones de probabilidad (https://relopezbriega.github.io/blog/2016/06/29/distribuciones-de-probabilidad-con-python/).

En Python (https://www.python.org/), podemos representar a las matrices dispersas (https://es.wikipedia.org/wiki/Matriz_dispersa) con la ayuda de scipy.sparse (https://docs.scipy.org/doc/scipy/reference/sparse.html).

In [13]:

```
# Ejemplo matriz dispersa con scipy
A = sp.diags([1, -2, 1], [1, 0, -1], shape=[10, 10], format='csc')
A
```

Out[13]:

```
<10x10 sparse matrix of type '<class 'numpy.float64'>'
        with 28 stored elements in Compressed Sparse Column format>
```

In [14]:

```
A.data
```

Out[14]:

```
array([-2.,  1.,  1., -2.,  1.,  1., -2.,  1.,  1., -2.,  1.,  1., -2.,
        1.,  1., -2.,  1.,  1., -2.,  1.,  1., -2.,  1.,
        1., -2.])
```

In [15]:

```
A.indices
```

Out[15]:

```
array([0, 1, 0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6, 5, 6, 7, 6, 7, 8,
       7, 8, 9, 8, 9], dtype=int32)
```

In [16]:

```
A.indptr
```

Out[16]:

```
array([ 0,  2,  5,  8, 11, 14, 17, 20, 23, 26, 28], dtype=int32)
```

In [17]:

```
A.todense()
```

Out[17]:

```
matrix([[-2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 1., -2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  1., -2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  1., -2.,  1.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  1., -2.,  1.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  1., -2.,  1.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  1., -2.,  1.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  1., -2.,  1.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -2.,  1.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -2.]])
```

La descomposición en valores singulares (SVD) y el análisis de componentes principales (PCA)

El SVD (https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares) y el PCA (https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales) son herramientas ampliamente utilizadas, por ejemplo, en el análisis de imágenes médicas para la reducción de dimensionalidad (https://en.wikipedia.org/wiki/Dimensionality_reduction), la construcción de modelos, y la comprensión y exploración de datos. Tienen aplicaciones en prácticamente todas las áreas de la ciencia, machine learning (https://relopezbriega.github.io/tag/machine-learning.html), procesamiento de imágenes (https://en.wikipedia.org/wiki/Image_processing), ingeniería (https://es.wikipedia.org/wiki/Ingenier%C3%ADa), genética (https://es.wikipedia.org/wiki/Gen%C3%A9tica), computación cognitiva (https://en.wikipedia.org/wiki/Cognitive_computing), química (https://es.wikipedia.org/wiki/Qu%C3%ADmica), meteorología (https://es.wikipedia.org/wiki/Meteorolog%C3%ADa), y redes neuronales

(https://relopezbriega.github.io/tag/redes-neuronales.html), sólo por nombrar algunas; en dónde nos encontramos con grandes conjuntos de datos (https://es.wikipedia.org/wiki/Conjunto_de_datos). El propósito del análisis de componentes principales PCA (https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales) es derivar un número relativamente pequeño de combinaciones lineales no correlacionadas (componentes principales) de una conjunto de variables aleatorias de media (https://es.wikipedia.org/wiki/Media_aritm%C3%A9tica) cero mientras que conserva la mayor cantidad de información de las variables originales como sea posible. Entre los objetivos del PCA (https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales) podemos encontrar los siguientes:

1. Reducción de dimensionalidad (https://en.wikipedia.org/wiki/Dimensionality_reduction).
2. Determinación de combinaciones lineales de variables.
3. Selección de características o *features*: la elección de las variables más útiles.
4. Visualización de datos multidimensionales.
5. Identificación de las variables subyacentes.
6. Identificación grupos de objetos o de valores atípicos (https://es.wikipedia.org/wiki/Valor_at%C3%ADpico).

Veamos algunos ejemplos con Python (https://www.python.org/)

In [18]:

```
# Ejemplo SVD con scipy.linalg.svd
# Matriz A a factorizar
A = np.array([[2, 4]
             ,[1, 3]
             ,[0, 0]
             ,[0, 0]])
A
```

Out[18]:

```
array([[2, 4],
       [1, 3],
       [0, 0],
       [0, 0]])
```

In [19]:

```
# Factorización con svd
# svd factoriza la matriz A  en dos matrices unitarias U y Vh, y una
# matriz s de valores singulares (reales, no negativo) de tal manera que
# A == U * S * Vh, donde S es una matriz con s como principal diagonal y ceros
U, s, Vh = la.svd(A)
U.shape, Vh.shape, s.shape
```

Out[19]:

```
((4, 4), (2, 2), (2,))
```

In [20]:

```
# Matriz unitaria
U
```

Out[20]:

```
array([[-0.81741556, -0.57604844,  0.        ,  0.        ],
       [-0.57604844,  0.81741556,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  1.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  1.        ]])
```

In [21]:

```
# Valores singulares
s
```

Out[21]:

```
array([ 5.4649857 ,  0.36596619])
```

In [22]:

```
# Matriz unitaria
Vh
```

Out[22]:

```
array([[-0.40455358, -0.9145143 ],
       [-0.9145143 ,  0.40455358]])
```

In [23]:

```
# Generando S
S = la.diagsvd(s, 4, 2)
S
```

Out[23]:

```
array([[ 5.4649857 ,  0.        ],
       [ 0.        ,  0.36596619],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ]])
```

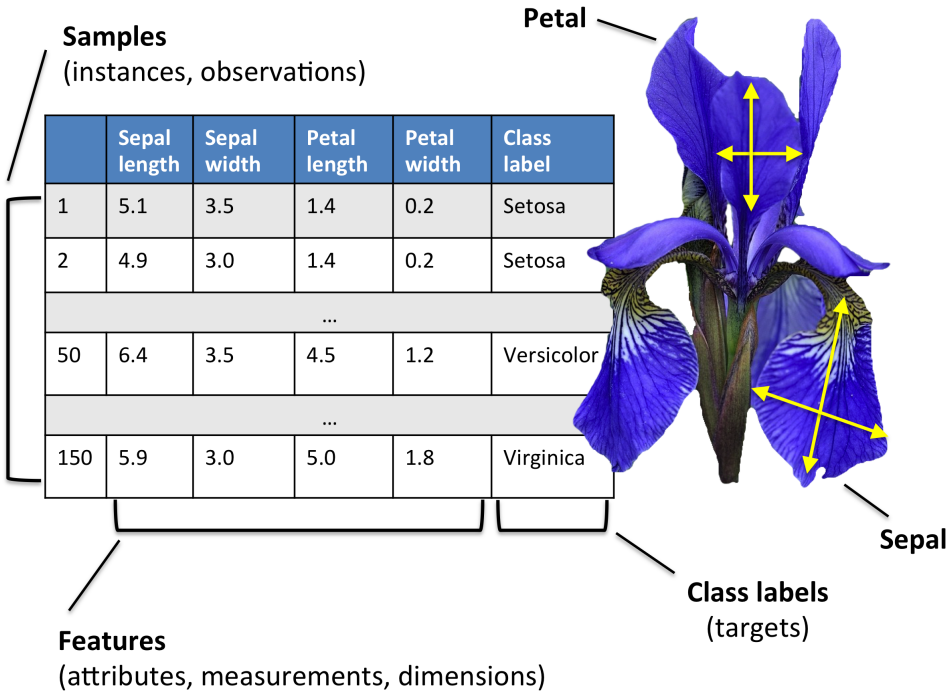
In [24]:

```
# Reconstruyendo la Matriz A.
U @ S @ Vh
```

Out[24]:

```
array([[ 2.,  4.],
       [ 1.,  3.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

Ejemplo de SVD y PCA con el dataset Iris



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

El dataset (https://es.wikipedia.org/wiki/Conjunto_de_datos) iris contiene mediciones de 150 flores de iris de tres especies diferentes.

Las tres clases en el dataset (https://es.wikipedia.org/wiki/Conjunto_de_datos) son:

1. setosa (n = 50).
2. versicolor (n = 50).
3. virginica (n = 50).

Las cuales están representadas por cuatro características:

1. longitud en cm del sépalo (https://es.wikipedia.org/wiki/S%C3%A9palo).
2. ancho en cm del sépalo (https://es.wikipedia.org/wiki/S%C3%A9palo).
3. longitud en cm del pétalo (https://es.wikipedia.org/wiki/P%C3%A9talo).
4. ancho en cm del pétalo (https://es.wikipedia.org/wiki/P%C3%A9talo).

In [25]:

```
# Ejemplo svd con iris dataset
iris = sns.load_dataset("iris")
print(iris.shape)
iris.head()
```

```
(150, 5)
```

Out[25]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [26]:

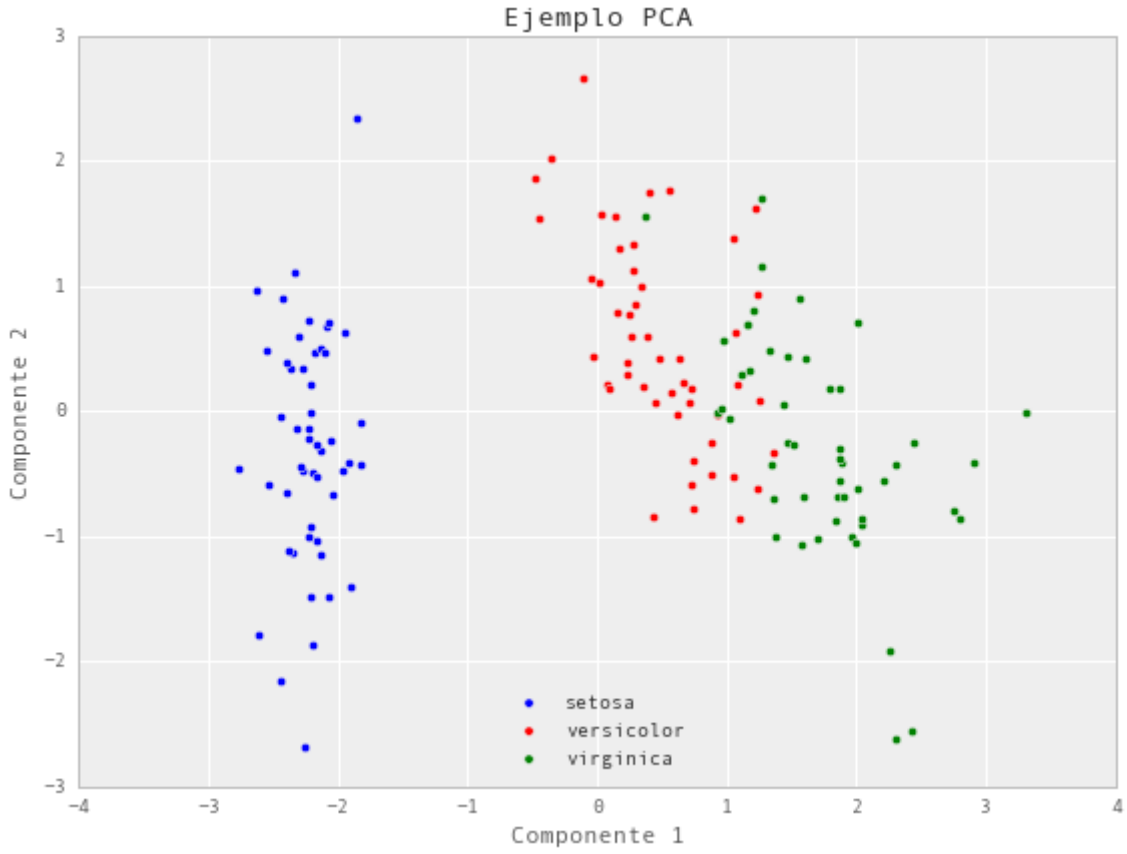
```
# Aplicando svd
U, s, Vh = sp.linalg.svds((iris - iris.mean()).iloc[:,:-1],2)
```

In [27]:

```
# Creando los componentes principales
pc = U @ np.diag(s)
pc = pc[:,::-1]

# graficando el dataset reducido a 2 componenetes
iris_svd = pd.concat((pd.DataFrame(pc, index=iris.index
                        , columns=('c0','c1')), iris.loc[:,'species']),1)

g = sns.lmplot('c0', 'c1', iris_svd, hue='species', fit_reg=False, size=8
              ,scatter_kws={'alpha':0.7,'s':60})
```

In [28]:

```
# Ejemplo de PCA con Scikit-Learn e Iris dataset

# Divido el dataset en datos y clases
X = iris.ix[:,0:4].values
y = iris.ix[:,4].values

# Estandarizo los datos
X_std = StandardScaler().fit_transform(X)

pca = PCA(n_components=2)
Y_pca = pca.fit_transform(X_std)

# Visualizo el resultado
for lab, col in zip(('setosa', 'versicolor', 'virginica'),
                    ('blue', 'red', 'green')):
    plt.scatter(Y_pca[y==lab, 0],
                Y_pca[y==lab, 1],
                label=lab,
                c=col)

plt.xlabel('Componente 1')
plt.ylabel('Componente 2')
plt.legend(loc='lower center')
plt.tight_layout()
plt.title('Ejemplo PCA')
plt.show()
```



Factorización de matrices en sistemas de recomendación

En un sistemas de recomendación (https://es.wikipedia.org/wiki/Sistema_de_recomendaci%C3%B3n) como Netflix (https://www.netflix.com/) o MovieLens (https://movielens.org/), hay un grupo de usuarios y un conjunto de elementos (películas en los dos sistemas anteriores). Teniendo en cuenta que cada usuario ha valorado algunos elementos en el sistema, nos gustaría predecir cómo los usuarios calificarían los artículos que aún no se han valorado, de tal manera que podemos hacer recomendaciones a los usuarios. En este caso, toda la información que tenemos sobre las calificaciones existentes pueden ser representados en una matriz

(https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)). Supongamos ahora que tenemos 5 usuarios y 4 películas y las calificaciones son números enteros de 1 a 5, la matriz (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) resultante puede ser algo como la siguiente (el guión significa que el usuario aún no ha calificado la película):

	P1	**P2**	**P3**	**P4**
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

Por lo tanto, la tarea de predecir las calificaciones que faltan se puede considerar como un problema de llenar los espacios en blanco (los guiones en la matriz) de tal manera que los valores resultantes serían consistentes con las calificaciones existentes en la matriz (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)).

La intuición detrás de usar **Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices)** para resolver este problema es que deberían existir algunas características latentes que determinen cómo un usuario califica una película. Por ejemplo, dos usuarios darían una alta calificación a una cierta película si a ambos les gusta los actores / actrices de la película, o si la película es una película de acción, que es un género preferido por ambos usuarios. Por lo tanto, si podemos descubrir estas características latentes, deberíamos ser capaces de predecir una calificación con respecto a un determinado usuario y una cierta película, porque las características asociadas con el usuario deben coincidir con las características asociadas con la película.

Al tratar de descubrir las diferentes características latentes, estamos suponiendo implícitamente que el número de características va a ser menor que el número de usuarios y el número de elementos. No debería ser difícil de entender este supuesto ya que no sería razonable que cada usuario está asociado con una característica única (aunque esto no es imposible). Y de todos modos, si este es el caso, no tendría sentido hacer recomendaciones, porque ninguno de estos usuarios estarían interesados en los artículos calificados por otros usuarios.

Si llevamos este ejemplo sencillo a Python (https://www.python.org/), podríamos modelarlo utilizando Scikit-learn (https://scikit-learn.org/stable/) NMF (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html) o Nimfa (https://nimfa.biolab.si/nimfa.methods.factorization.snmf.html).

In [29]:

```
# Ejemplo en python
# Matriz de ratings de los usuarios
R = np.array([[5, 3, 0, 1]
             ,[4, 0, 0, 1]
             ,[1, 1, 0, 5]
             ,[1, 0, 0, 4]
             ,[0, 1, 5, 4]])
```

In [30]:

```
# Armado del modelo
modelo = NMF(n_components= 2, init='random',random_state=1982,
            alpha=0.0002, beta=0.02, max_iter=5000)
U = modelo.fit_transform(R)
V = modelo.components_
```

In [31]:

```
# Reconstrucción de la matriz
U @ V
```

Out[31]:

```
array([[ 5.25534386,  1.99295604,  0.        ,  1.45508737],
       [ 3.50398031,  1.32879578,  0.        ,  0.97017391],
       [ 1.31288534,  0.94413861,  1.9495384 ,  3.94596646],
       [ 0.98125417,  0.7217855 ,  1.52757221,  3.07874315],
       [ 0.        ,  0.65011225,  2.84008987,  5.21892884]])
```

In [32]:

```
# Error del modelo
modelo.reconstruction_err_
```

Out[32]:

```
4.276529876124528
```

In [33]:

```
# Ejemplo utilizando nimfa
snmf = nimfa.Snmf(R, seed="random_vcol", rank=2, max_iter=30, version='r', eta=1.,
                  beta=1e-4, i_conv=10, w_min_change=0)
```

In [34]:

```
# Armando el modelo
fit = snmf()
U = fit.basis()
V = fit.coef()
```

In [35]:

```
# Reconstruyendo la matriz
np.around(U @ V, decimals=2)
```

Out[35]:

```
array([[ 5.18,  1.96,  0.  ,  1.44],
       [ 3.45,  1.31,  0.  ,  0.96],
       [ 1.32,  0.93,  1.91,  3.87],
       [ 0.98,  0.71,  1.5 ,  3.02],
       [ 0.  ,  0.63,  2.79,  5.11]])
```

Librerías de Python para factorización de matrices

Para concluir, podemos enumerar algunas de las librerías de Python (https://www.python.org/) que nos pueden ser de mucha utilidad a la hora de trabajar con Factorización de matrices (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_matrices), como ser:

- **NumPy (https://www.numpy.org/)**: El popular paquete matemático de Python (https://python.org/), nos va a permitir crear *vectores (https://es.wikipedia.org/wiki/Vector)*, *matrices (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas))* y *tensores (https://es.wikipedia.org/wiki/C%C3%A1lculo_tensorial)*; y poder manipularlos y realizar operaciones sobre ellos con mucha facilidad.
- **SciPy (https://docs.scipy.org/doc/scipy/reference/index.html)**: El paquete que agrupa un gran número de herramientas científicas en Python (https://www.python.org/). Nos va a permitir crear matrices dispersas (https://es.wikipedia.org/wiki/Matriz_dispersa) y realizar factorizaciones (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n) con facilidad.
- **Scikit-learn (https://scikit-learn.org/stable/)**: Es una librería especializada en algoritmos para data mining (https://es.wikipedia.org/wiki/Miner%C3%ADa_de_datos) y machine learning (https://es.wikipedia.org/wiki/Machine_learning). Principalmente el submódulo de Descomposiciones

(https://scikit-learn.org/stable/modules/decomposition.html#decompositions) en dónde vamos a encontrar las herramientas para reducciones de dimensionalidad (https://en.wikipedia.org/wiki/Dimensionality_reduction).

- **SymPy (https://www.sympy.org/es/)**: Esta librería nos permite trabajar con matemática simbólica, convierte a Python (https://python.org/) en un sistema algebraico computacional (https://es.wikipedia.org/wiki/Sistema_algebraico_computacional). Nos va a permitir trabajar con las factorizaciones (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n) en forma analítica.
- **Nimfa (https://nimfa.biolab.si/)**: Nimfa (https://nimfa.biolab.si/) es una librería para factorización de matrices no negativas (https://en.wikipedia.org/wiki/Non-negative_matrix_factorization). Incluye las implementaciones de varios métodos de factorización (https://es.wikipedia.org/wiki/Factorizaci%C3%B3n), enfoques de inicialización y calificación de calidad. Soporta representaciones tanto de matrices densas (https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas)) como dispersas (https://es.wikipedia.org/wiki/Matriz_dispersa). Se distribuye bajo licencia BSD (https://es.wikipedia.org/wiki/Licencia_BSD).

Con esto termina este artículo, espero les haya parecido interesante y les sea de utilidad en sus proyectos.

Saludos!

Este post fue escrito utilizando IPython notebook. Pueden descargar este notebook (https://github.com/relopezbriega/relopezbriega.github.io/blob/master/downloads/MatrixFact.ipynb) o ver su version estática en nbviewer (https://nbviewer.ipython.org/github/relopezbriega/relopezbriega.github.io/blob/master/downloads/MatrixFact.ipynb).

Posted by Raul E. Lopez Briega Tue 13 September 2016 python (https://relopezbriega.github.io/tag/python.html) programacion (https://relopezbriega.github.io/tag/programacion.html) algebra (https://relopezbriega.github.io/tag/algebra.html) Machine Learning (https://relopezbriega.github.io/tag/machine-learning.html) Redes Neuronales (https://relopezbriega.github.io/tag/redes-neuronales.html) matrices (https://relopezbriega.github.io/tag/matrices.html) recomendaciones (https://relopezbriega.github.io/tag/recomendaciones.html) matematica (https://relopezbriega.github.io/tag/matematica.html)

Comments