Non-Negative Matrix Factorization

A quick tutorial

Matrices (also Matrixes)

In mathematics, a matrix (plural matrices) is a rectangular array of numbers arranged in rows and columns. The individual items in a matrix are called its elements or entries.

An example of a matrix with 2 rows and 3 columns is:

$$\begin{bmatrix} 2 & 3 & 19 \\ 5 & 7 & 11 \end{bmatrix}$$

Source: [Wikipedia]

Size of Matrices

The size of a matrix is defined by the number of rows and columns that it contains. A matrix with *m* rows and *n* columns is called an *m* × *n* matrix or *m-by-n* matrix, while *m* and *n* are called its dimensions.

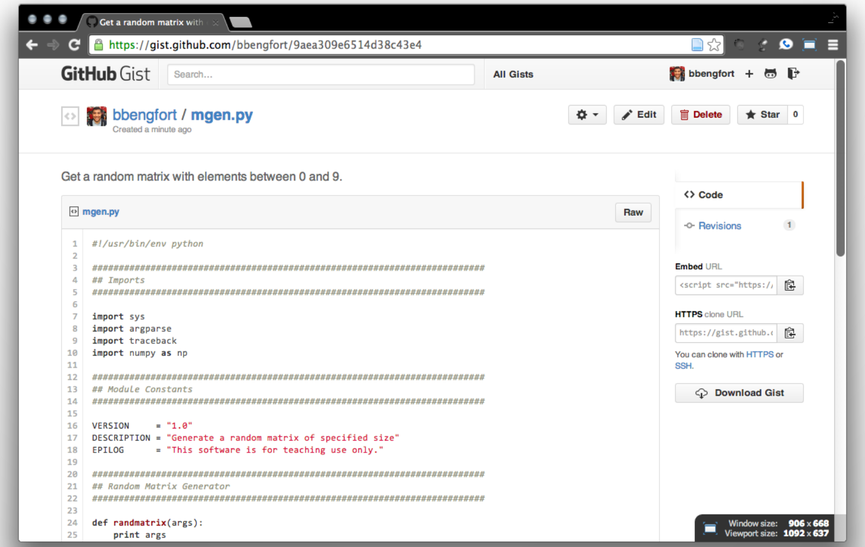
$$\begin{bmatrix} 2.2 & 3.0 & 1.9 \\ 6.2 & 1.2 & -2.1 \\ 5.1 & 0.7 & 1.1 \\ 9.2 & 1.2 & -2.2 \end{bmatrix}$$

Size?

Basic Matrix Operations

We'll discuss some basic matrix operations next, and we'll practice by generating random matrices* using a simple python script that can be found here:

http://bit.ly/mgen-py



Get a random matrix with elements between 0 and 9.

```
#!/usr/bin/env python

##########################################################################
## Imports
##########################################################################

import sys
import argparse
import traceback
import numpy as np

##########################################################################
## Module Constants
##########################################################################

VERSION     = "1.0"
DESCRIPTION = "Generate a random matrix of specified size"
EPILOG      = "This software is for teaching use only."

##########################################################################
## Random Matrix Generator
##########################################################################

def randmatrix(args):
    print args
```

*Please applaud instructor level of difficulty and excuse whiteboard arithmetic mistakes.

Adding (and Subtracting) Matrices

For matrices of the same dimension, add together the corresponding elements to result in another matrix of the same dimension.

$$A+B=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{1n}+b_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \dots & a_{mn}+b_{mn} \end{bmatrix}$$

Scalar Multiplication

To multiply a scalar denoted as i, or any real number against a matrix, A - simply multiply i to each element of the matrix.

$$i \times A = \begin{bmatrix} i(a_{11}) & i(a_{12}) & \dots & i(a_{1n}) \\ i(a_{21}) & i(a_{21}) & \dots & i(a_{1n}) \\ \vdots & \vdots & \ddots & \vdots \\ i(a_{m1}) & i(a_{m1}) & \dots & i(a_{mn}) \end{bmatrix}$$

Transposing Matrices

Reflect a matrix along its diagonal by swapping the matrix’s rows and columns. Such that a *m* x *n* matrix becomes an *n* x *m* matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix}$$

Matrix Multiplication

A matrix product between a matrix A with size *n* x *m* and B with size *m* x *p* will produce an *n* x *p* matrix in which the *m* columns of A are multiplied against the *m* rows of B as follows:

$$AB = \begin{bmatrix} (AB)_{11} & (AB)_{12} & \dots & (AB)_{1p} \\ (AB)_{21} & (AB)_{22} & \dots & (AB)_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ (AB)_{n1} & (AB)_{n2} & \dots & (AB)_{np} \end{bmatrix}$$

$$(AB)_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \longleftarrow$$

The Dot Product of the row in A and column in B

The Dot Product

The dot product of two vectors (e.g. 1 x *m* or *n* x 1 matrices) of the same size is the sum of the products of each element at every direction.

$$A \cdot B = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

$$A \cdot B = \sum_{i=1}^{n} a_i b_i$$

Matrix Sparsity (or Density)

The percentage of non-zero elements to the number of elements in total (and 1 - the ratio of non-zero elements to the number of elements).

```
[  11  22   0   0   0   0   0  ]
[   0  33  44   0   0   0   0  ]
[   0   0  55  66  77   0   0  ]
[   0   0   0   0   0  88   0  ]
[   0   0   0   0   0   0  99  ]
```

Matrix Factorization (or Decomposition)

Given a *n* x *m* matrix, R - find two smaller matrices, P and Q with *k*-dimensional features - e.g. that P is size *n* x *k* and Q is size *m* x *k* - such that their product approximates R. (Adjusting *k* allows for more accuracy when reconstructing the original matrix.)

$$R \approx P \times Q^T = \hat{R}$$

Note: Sparse Matrix Factorization is used when the matrix is populated primarily by zeros

Recommendations

In order to compute recommendations, we will construct a *users* x *movies* matrix such that every element of the matrix is the user’s rating, if any:

	Star Wars	Bridget Jones	The Hobbit ...
Bob	5	2	0
Joe	3	4	2
Jane	0	0	3
...	...	...	...

As you can see - this is a pretty sparse matrix!

Factored Matrices

- P is the *features* matrix. It has a row for each feature and a column for each column in the original matrix (movie).
- Q is the *weights* matrix. It has a column for each feature, and a row for each row in the original matrix (user).
- Therefore when you multiply the dot product, you’re finding the sum of the latent features by the weights for each element in the original matrix.

Latent Features

- The features and weights described before measure “latent” features - e.g. hidden, non-human describable features and their weights, but could relate to things like genre.
- You need less features than items, otherwise the best answer is simply every item (no similarity).
- The magic is where there are zero values - the product will fill them in, *predicting* their value.

Non-negative

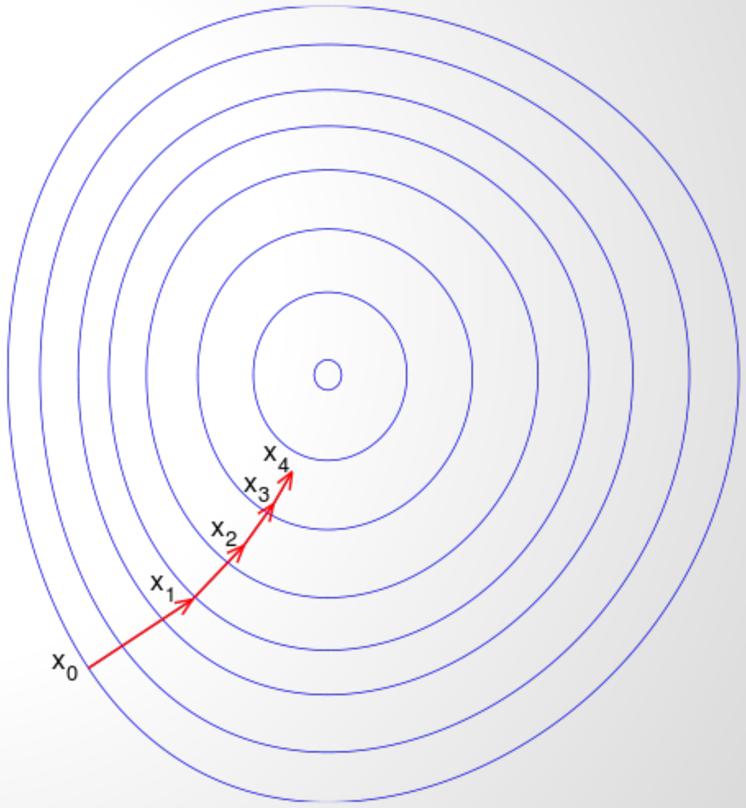
Called non-negative matrix factorization because it returns features and weights with no negative values. Therefore all features must be positive or zero values.

Clustering

Non-Negative Matrix Factorization is closely related to both supervised and unsupervised methodologies (supervised because R can be seen as a training set) - but in particular NNMF is closely related to other clustering (unsupervised) algorithms.

Gradient Descent

- The technique we will use to factor is called gradient descent, which attempts to *minimize error*.
- We can calculate error from our product using the squared error (actual - predicted)^2
- Once we know the error, we can calculate the gradient in order to figure out what direction to go to minimize the error. We keep going until we have no more error.



The Algorithm

```
→ Initialize P and Q with random small numbers
→ for step until max_steps:
      for row, col in R:
          if R[row][col] > 0:
              compute error of element
              compute gradient from error
              update P and Q with new entry

          compute total error
          if error < some threshold:
              break
    return P, Q.T
```

Needed Computations

Compute* predicted element for each user-movie pair (dot product of row and column in P and Q):

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Compute the squared error for each user-movie pair (in order to compute the gradient):

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj})^2$$

*computations for non-zero values only

Needed Computations

Find gradient (slope of error curve) by taking the differential of the error for each element:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj}$$

$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}q_{ik}$$

Update Rule

Update each element in P and Q by using a learning rate (called α) - this determines how far to travel along the gradient. α is usually small, because if we choose a step size that is too large, we could miss the minimum.

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} p_{ik}$$

Convergence and Regularization

We converge once the sum of the errors has reached some threshold, usually very small.

$$E = \sum_{(p_i, q_j, r_{ij})} (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj})^2$$

Extending this algorithm will introduce regularization to avoid overfitting by adding a beta parameter. This forces the algorithm to control the magnitudes of the feature vectors.

Predicted Recommendations

Our predicted matrix for our movies rating, will end up looking something like what follows: Where there were zeros before, we now have predictions!

	Star Wars	Bridget Jones	The Hobbit ...
Bob	4.98148768	2.02748447	3.29852779
Joe	3.0157327	3.968359	2.01139212
Jane	4.50410968	2.93580899	2.98826278
...	...	...	...

As you can see - the predictions are very close to the actual values, but we now suspect that Jane will really like Star Wars!