Image Compression & Recommendation Systems

Tool : Singular Value Decomposition

The Analytics Edge

Images are an important type of unstructured data on the Internet. Companies that share user photo such as Instagram, Facebook, Flicker, Twitter need efficient ways to deal with photos of Users. Data Compression is an important tool that companies use to tradeoff quality of photos and file size (memory). Tools of matrix factorizatn such as singular value decomposition is an useful tool. Similarly such techniques are useful in recommendation systems where singular value decomposition on matrices with missing entries are used to make user recommendations.

Photos

Instagram is an online photo sharing site that enables users to take and share photos. It was created in 2010 as a free mobile app and acquired by Facebook in 2012. The company valuation is today between 30 to 37 billion. The total number of photos uploaded has been over 1 billion.

Just recently (May 2015), Google has launched Google Photos, a photo and video sharing and storage service in an attempt to improve on the popularity of Google +.

"A picture is worth a thousand words"

So just how many pictures (photos) are uploaded to the Internet?

It is estimated that 2400 Instagram photos are uploaded in 1 second (internetlivestat.com/one-second)

Each day, a total of 1.8 billion photos are estimated to be uploaded to the Internet

Assuming that an image (photo) on average is 100 Kb. This means that for just one day, the amount of bytes needed = 1.8×10^{14} bytes which is 180 Terabytes.

Image representation

$$\underbrace{\text{Image}}_{\text{Pixel representation}} \longrightarrow \text{Matrix}$$

Grayscale image

A grayscale image is an image where the value of each pixel is a single number that carries the intensity information.
For example the images have intensity going from white (0) to black (1) with the different shades of gray taking values in (0, 1).

Color image

Color images can be built as a set of stacked color channels, each of them representing value levels of the given channel.

$$\text{RGB images} \longrightarrow \begin{array}{c} \text{Three channels} \\ \underbrace{\text{Red, Green, Blue}}_{\text{Three primary colors}} \end{array}$$

Singular Value Decomposition (SVD)

Given a rectangular matrix X of dimension $m \times n$, a singular value decomposition of X is of the form: (say $m \geq n$)

$$\underbrace{X}_{m \times n} = \underbrace{U}_{m \times m} \; \underbrace{S}_{m \times n} \; \underbrace{V^T}_{n \times n}$$

$$\begin{array}{cc} & n \\ m & \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \end{array} = \begin{array}{cc} & m \\ m & \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \end{array} \begin{array}{cc} & n \\ m & \left(\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \\ \hline & 0 & \end{array}\right) \end{array} \begin{array}{cc} & n \\ n & \begin{pmatrix} & \\ & \end{pmatrix} \end{array}$$

where:

1) Both matrices U and V are orthogonal matrices, namely $U^TU = UU^T = I$ and $V^TV = VV^T = I$ where I is an identity matrix

2) Matrix S has diagonal entries $\sigma_1, \ldots, \sigma_r \geq 0$ where $r = \min(m, n)$ at the top and 0's filling the rest of the matrix. ($\sigma_1, \ldots, \sigma_r$ are the singular values)

Equivalently matrix X can be expressed as:

$$X = \underbrace{\sum_{j=1}^{r} \sigma_j u_j v_j^T}_{\text{Sum of rank one matrices}}$$

where $\sigma_j \geq 0$, u_j is the jth column of U and v_j is the jth column of V

Example :

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{pmatrix}$$

$\downarrow$ svd

$$X = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{pmatrix}$$

Low rank approximations

Without loss of generality say $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$

The main idea is that we develop a simpler versio (low rank approximation) of X by dropping the smaller singular values.

A rank k approximation to X is given by:

$$\hat{X} = \sum_{j=1}^{k} \sigma_j u_j v_j^T \quad (\text{where } k \leq r)$$

$$\overset{n}{m\begin{pmatrix} \hat{X} \end{pmatrix}} = \overset{k}{m\underbrace{\begin{pmatrix} \quad \\ \quad \end{pmatrix}}_{\substack{\text{Use first} \\ k \text{ columns} \\ \text{of } U}}} \; \overset{k}{k\begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{pmatrix}} \; \overset{n}{k\underbrace{\begin{pmatrix} \quad \end{pmatrix}}_{\substack{\text{Use first } k \\ \text{columns of } V \\ \text{to create } V^T}}}$$

Example : Rank 2 approximation to X on the previous page is :

$$\hat{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

svd approximation

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{pmatrix}$$

One of the big advantages of performing a low rank approximation of X is that to represent a $m \times n$ matrix, the number of parameters needed $= mk + k + nk$

$$= k(m+n+1)$$

For example :

Say $m = 200$, $n = 320$. This matrix has 64000 entries.

If we use $k = 20$, we would need $20(200+320+1) = 10420$ entries to approximate the matrix.

Relation to eigenvalues

The SVD applies to any $m \times n$ (rectangular) matrix where eigenvalue decomposition applies to square matrices

$$X = U S V^T$$

$$X X^T = U S V^T V S^T U^T$$

$$= U S I S^T U^T \quad (\text{Since } V^T V = I)$$

$$= U (S S) U^T \quad (\text{Since } S^T = S \text{ is diagonal})$$

$$= U S^2 U^T$$

Thus the column vectors in U are the eigenvectors of $X X^T$ and the nonzero elements σ_k are the square root of the nonzero eigenvalues of $X X^T$

Similarly $X^T X = V S U^T U S V^T$

$$= V (S S) V^T$$

$$= V S^2 U^T$$

Here $S^2 = \begin{pmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_r^2 \\ & 0 & \end{pmatrix}$

The Frobenius norm of a matrix is defined as:

$$\|X\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^2}$$

Equivalently $\|X\|_F = \sqrt{\sigma_1^2 + \dots + \sigma_r^2}$ where $\sigma_1, \dots, \sigma_r$ are the singular values of X.

In approximating a matrix by a low rank matrix, the accuracy of the approximation (in terms of the explained variance) is given as:

$$\frac{\|\hat{X}\|_F^2}{\|X\|_F^2} = \frac{\sigma_1^2 + \dots + \sigma_k^2}{\sigma_1^2 + \dots + \sigma_r^2}$$

Analytics on image data

install.packages("jpeg")

library(jpeg)

Package to read, write and display images in JPEG format

Read in data from image

lky ← readJPEG("lky.jpg")

This reads the image into a three dimensional array. The figure is of size 100 KB. This is read into an array of size 410 × 640 × 3 where the dimensions are height × width × channels. (You can use file.info("lky.jpg"))

min(lky[, , 1])

max(lky[, , 1])

Values in the data representation of image range from 0.0313 to 1

max(abs(lky[, , 1] - lky[, , 2]))

max(abs(lky[, , 2], lky[, , 3]))

It can be verified that all three channels have same values

Perform SVD

S ← svd(lky[, , 1])

Perform a singular value decomposition on the matrix

str(S)

S$d consists of the singular values (410)
S$u consists of left singular vectors (410 × 410)
S$v consists of right singular vectors (640 × 410)

Develop a low rank approximation

```
lky10 <- S$u[,1:10] %*% diag(S$d[1:10]) %*% t(S$v[,1:10])
```

Computes a rank 10 approximation to the matrix by using the top 10 singular values. Note %*% is used here for matrix multiplication

```
writeJPEG(lky10, "lky10.jpg")
```

Note that this writes a grayscale image of size about 16 KB. However this file is lossy and loses a lot of clarity.

```
lky50 <- S$u[,1:50] %*% diag(S$d[,1:50])
         %*% t(S$v[,1:50])
```

Computes a rank 50 approximation to the matrix

```
writeJPEG(lky50, "lky50.jpg")
```

Note that this writes a grayscale image of size about 24 KB. which is already very clear

```
var <- cumsum(S$d^2)
plot(1:410, var)
plot(1:410, var/max(var))
```

In about 18 singular values out of 410, The approximation contains about 99% of the total variation in the picture

Read in from image

pansy <- readJPEG("pansy.jpg")

This reads the image of the flower into an array of size 600 × 465 × 3.
Each of the numbers in the third dimension corresponds to an intensity in the R, G or B channel.

Note that in this case pansy[,,1] ≠ pansy[,,2] ≠ pansy[,,3].

Perform SVD

S1 <- svd(pansy[,,1])

S2 <- svd(pansy[,,2])

S3 <- svd(pansy[,,3])

str(S1) — List with S1$d consisting of 465 singular values, S1$u consisting of left singular vectors (600×465) and S1$v consisting of right singular vectors (465 × 465). Similarly for S2 and S3

Develop low rank approximations

pansy50 <- array(dim = c(600, 465, 3))

pansy50[,,1] <- S1$u[,1:50] %*% diag(S1$d[1:50]) %*% t(S1$v[,1:50])

:

pansy50[,,3] <- S3$u[,1:50] %*% diag(S3$d[1:50]) %*% t(S3$v[,1:5

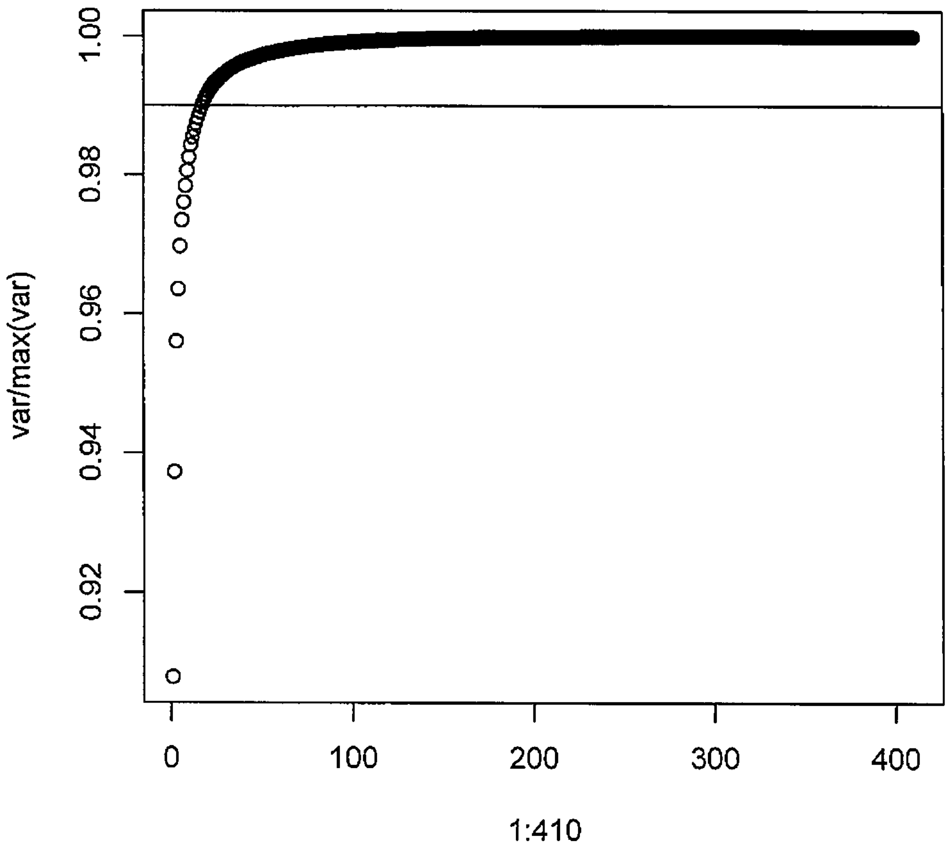
```
writeJPEG(pansy50, "pansy50.jpg")
```

This creates an image of size about 40 KB which has some blurring along edges of the flower and leaves

```
pansy350 <- array(dim = c(600,465,3))
pansy350[,,1] <- S1$u[,,1:350] %*% diag(S1$d[1:350])
                 %*% t(S1$v[,1:350])
  :
pansy350[,,3] <- ...

writeJPEG(pansy350, "pansy350.jpg")
```

This creates an image of size about 45 KB which is quite a good representation of the original image. The original file size here is around 80 KB.



1.00
0.98
0.96
0.94
0.92
var/max(var)
0
100
200
300
400
1:410

SVD in recommendation systems

$$\begin{array}{cc} & \text{Movies} \\ \text{Users} & \begin{pmatrix} \\ \text{Rating matrix} \\ \\ \end{pmatrix} \end{array} \xrightarrow[\text{approximation}]{\text{SVD}} \begin{array}{cc} & \text{Factors} \\ \text{Users} & \begin{pmatrix} - \; p_u^T \; - \\ \\ \\ \end{pmatrix} \end{array} \begin{array}{cc} & \text{Movies} \\ \text{Factors} & \begin{pmatrix} | \\ q_i \\ | \end{pmatrix} \end{array}$$

Users are associated with a vector p_u and movies (items) with a vector q_i, both of dimension "k".

Rating is approximated as:

$$\hat{r}_{ui} = p_u^T q_i \quad (\text{or } q_i^T p_u)$$

Such a factorization is computed as a SVD where the diagonal matrix can be absorbed into either the U or V matrix. However the major challenge is that in recommendation systems such as Netflix, many entries are missing and hence conventional SVD fails

There are a few ways to tackle this:

1) Impute (fill in) the entries using a reasonable method such as taking the average. Apply SVD to the full matrix. In the Netflix application, however this implies the matrix to be factorized is very large. Also inaccurate imputation can affect the data.

2) An alternate approach is to work only with the observed ratings and using regularization as in the LASSO model for overfitting. This was one of the approaches (models) used in the final Netflix winning contribution

$$\min_{\substack{q_i, p_u \\ \forall (i,u)}} \sum_{\substack{(u,i) \in \text{Observed} \\ \text{user-rating pairs}}} \left[\left(r_{ui} - q_i^T p_u \right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 \right) \right]$$

One can use simple stochastic gradient methods to solve this problem. (also known as incremental gradient method)

For example:

Start with q_i & p_u chosen arbitrarily:

For each training set user-item pair,

Compute error $e_{ui} = r_{ui} - q_i^T p_u$

Use a gradient method by

$$q_i \leftarrow q_i + \gamma\left((r_{ui} - q_i^T p_u) p_u - \lambda q_i \right)$$

$$p_u \leftarrow p_u + \gamma\left(\underbrace{(r_{ui} - q_i^T p_u) q_i - \lambda p_u)}_{\text{Gradients negative}} \right)$$

This can be implemented using the funkSVD(.) function in the recommenderlab package.

```
Data1 <- Data
Data1[spl1c, spl2c] <- NA
P <- funkSVD(Data1, k=2)
```

k sets the rank of the SVD approximation

```
Predict <- tcrossprod(P$U, P$V)
```

This is equivalent to `P$U %*% t(P$V)`

```
Err <- sqrt(mean((Predict[spl1c, spl2c] - Data[spl1c, spl2c])^2, na.rm=TRUE))
```

You can vary k to get better approximation.

As k ↑, running time increases

In the training set, a smaller probe set was excluded. The models got trained using this data so as to minimize the errors in the probe set.

Afterwards the probe set was included & then using the same parameters, the second training was done on the entire training set. This was used to make predictions on the qualifying set.

Initially each predictive model was chosen to minimize the RMSE on the probe set.

Afterwards the team realized that the best blend of models is the one where the models were less correlated with the rest of the ensemble (blend) & achieved a low RMSE individually.

So instead of building each model individually, they built it iteratively each time adding new predictive models to blend with the earlier set of predictive models to minimize blend RMSE.. This helped in the final winning entry. Details can be found on netflixprize.com.

Final Comments on Netflix Prize

The final solution that won the Netflix prize involved a blend of various models developed by three teams. This involved models such as:

- Nearest neighbor models (K nearest neighbors)
- SVD models (matrix factorization)
- Models that incorporated temporal variations
- Regression models
- Classification models

Over 500 models were used in the final solution. This was important in winning the final competition but in the winners own words, a few good models could already land them on the leaderboard.

The approach in blending the various predictive models was an important step in the Netflix prize winning entry.