Recommender Systems: Pseudo SVD

There is another interesting use of Matrix Factorization that we will briefly review.

It will show both a case study and interesting extension of SVD.

Netflix Prize competition

- Predict user ratings for movies
- Dataset
    - Ratings assigned by users to movies: 1 to 5 stars
    - 480K users, 18K movies; 100MM ratings total
- $1MM prize
- Awarded to team that beat Netflix existing prediction system by at least 10 percentage points

User preference matrix

We will try to use same language as PCA (examples, features, synthetic features)

- But map them to Netflix terms
    - Examples: Viewers
    - Features: Movies ("items")

Matrix $\mathbf{X}$: user rating of movies

$\mathbf{X}_j^{(i)}$ is i^{th} user's rating of movie j

$\mathbf{X}$ is huge: $m * n$

- $m = .5$ million viewers
- $n = 18,000$ items (movies).

About 9 billion entries for a full matrix !

Idea: Linking Viewer to Movies via concepts

- Come up with your own "concepts" (synthetic features)
    - Concept = attribute of a movie
        - Map user preference to concept
        - Map movie style to concept
        - Supply and demand:
            - User demands concept, Movie provides concept

Human defined concepts

- Style: Action, Adventure, Comedy, Sci-fi
- Actor
- Typical audience segment

Making recommendations based on concepts

- Create user profile P: maps user to concept
- Create item profile Q: maps movies (features, items) to concept
- $\mathbf{X} = PQ^T$

To "recommend" a movie to a new user

- Given a sparse feature vector for the new user
- Obtain a dense vector
    - By mapping the sparse vector to concept space (synthetic features)
    - Finding a cluster of similar synthetic feature vectors, summarizing
    - Inverse transformation back to original features

The original features (movies) newly populated in the formerly sparse vector are the recommendations

One advantage of the $\mathbf{X} = PQ^T$ approach is a big space reduction.

With $k \leq n$ concepts:

- $\mathbf{X}$ is $m \times n$
- P is $m \times k$
- Q is $n \times k$

SVD to discover concepts

Why let a human guess concepts when Machine Learning can discover them ?

- Factor $\mathbf{X}$ by SVD !
    - Let SVD discovers the k "best" synthetic features, rather than leaving it to a human

Here's how to use SVD to discover P, Q:

$$\begin{aligned}\mathbf{X} &= U\Sigma V^T && \text{SVD of } \mathbf{X} \\ &= (U\Sigma)V^T \\ &= PQ && \text{Letting , } P = U\Sigma, Q = V^T\end{aligned}$$

Anyone spot the problem(s) ?

The matrix $\mathbf{X}$ with 9 billion entries is a handful !

But the problem is more acute than one of size.

Each row $\mathbf{X^{(i)}}$ is *sparse*

- Any single user views only a fraction of the n movies

How can we perform SVD on a matrix with missing values ?

Missing value imputation is not attractive

- Of the 9 billion potential entries in $\mathbf{X}$, only 100 million are defined
- Would impute more missing values than actual values

What can we do ?

The ML mantra

- It's all about the Loss function
- The essence of ML is finding a Loss function that describes a solution to your problem
- Gradient Descent is the "Swiss Army Knife" used for optimization of Loss functions

We will use "Pseudo SVD", a form of matrix decomposition based on minimizing a Loss.

Pseudo SVD Loss function

The Froebenius Norm

- Used in PCA as a metric with which to find the "best" low rank approximation
- Is modified to exclude missing values

$$\mathcal{L}(\mathbf{X}', \mathbf{X}) \quad = \quad \sum_{\substack{1 \le i \le m, \\ 1 \le j \le n \\ \mathbf{X}_j^{(i)} \text{defined}}} \left(\mathbf{X}_j^{(i)} - \mathbf{X'}_j^{(i)} \right)^2$$

That is: the loss is computed *only for the defined entries* of $\mathbf{X}$.

We can interpret the loss as a Reconstruction Error

Note that $\mathcal{L}(\mathbf{X}', \mathbf{X})$ is parameterized by P, Q

$$\begin{aligned}
\mathcal{L}(\mathbf{X}', \mathbf{X}) &= \sum_{\substack{1 \le i \le m, \\ 1 \le j \le n \\ \mathbf{X}_j^{(i)} \text{defined}}} \left(\mathbf{X}_j^{(i)} - \mathbf{X}'^{(i)}_j \right)^2 \\
&= \sum_{\substack{1 \le i \le m, \\ 1 \le j \le n \\ \mathbf{X}_j^{(i)} \text{defined}}} \left(\mathbf{X}_j^{(i)} - (PQ^T)_j^{(i)} \right)^2 \quad \text{since } \mathbf{X}' = PQ^T
\end{aligned}$$

P, Q are our *parameters* (e.g., Θ)

So we search for the P^*, Q^* that minimize $\mathcal{L}(\mathbf{X}', \mathbf{X})$

$$P^*, Q^* = \underset{P,Q}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{X}', \mathbf{X})$$

How ? Gradient Descent !

Pseudo SVD algorithm

- Define $\mathbf{X}' = PQ^T$
- Initialize elements of P, Q randomly.
- Take analytic derivatives of $\mathcal{L}(\mathbf{X}', \mathbf{X})$ with respect to
  - $P_j^{(i)}$ for $1 \leq i \leq m, 1 \leq j \leq k$
  - $Q_j^{(i)}$ for $1 \leq i \leq m, 1 \leq j \leq k$
- Use Gradient Descent to solve for optimal entries of P, Q.
  - Find entries of P, Q such that product matches non-empty part of $\mathbf{X}$

Note

- No guarantee that the P, Q obtained are
    - Orthonormal, etc. (which SVD would give you)

But SVD won't work for $\mathbf{X}$ with missing values.

Filling in missing values

Once you have P, Q

- to predict a missing rating for user i movie j:
$$\hat{r}_{j,i} = q^{(\mathbf{i})} \cdot p_j^T$$

- $q^{(\mathbf{i})}$ is row i of Q
- p_j is column j of P^T

Some intuition

The rating vector of a user may have missing entries.

But we can still project to synthetic feature space based on the non-empty entries.

The projection winds up in a "neighborhood" of concepts.

Inverse transformation

- Gets us to a completely non-empty rating vector that is a resident of this neighborhood.

Example

User rates

- Sci-Fi movies A and B very highly
- Does not rate Sci-Fi movie C.

Since A,B, C express same concept (Sci-Fi) they will be close in synthetic feature space.

Hence, the implied rating of User for movie C will be close to what other users rate C.

In [3]:
```
print("Done")
```

```
Done
```