Matrix Factorization and Collaborative Filtering

Daryl Lim

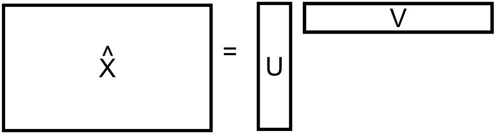
University of California, San Diego

February 7, 2013

Presentation Outline

Matrix Factorization: Overview

- Given a matrix $X \in \mathbb{R}^{m \times n}$, $m \leq n$, we want to find matrices U, V such that $X \approx UV = \hat{X}$
- In this talk, we consider the scenario when $\hat{X}$ is *low-rank*



- Why are low-rank approximations important?
  - Intuitively, if matrix is low rank, then the observations can be explained by linear combinations of *few* underlying factors
  - Want to know which factors control the observations

Collaborative Filtering: Overview

- Given a list of users and items, and user-item interactions, predict user behavior
- **Key idea:** Use *only* user-item interactions to predict behavior
  - iTunes Genius
- Contrast with *content-based filtering*, which builds a model for each item
  - e.g. Pandora hires musicologists to characterize songs, built the Music Genome

Matrix Factorization

This Talk

- Matrix factorization for dimensionality reduction (Principal Components Analysis or PCA)
- Non-negative Matrix Factorization, a related factorization method
- Matrix factorization methods for collaborative filtering applications

What can we do with PCA?

- Dimensionality reduction: Can we represent each data point with fewer features, without losing much?
  - Yes, if there is redundancy in the data!
- Data understanding: Which variables contribute to the largest variations in the data?
  - Look at the components of each principal component

What does PCA do?

- PCA finds a set of vectors called *principal components* that describe the data in an alternative way.
- The first principal component (PC) is the vector that maximizes the variance of the projected data.
- The kth PC is the vector orthogonal to all $k-1$ PCs that maximizes the variance of the projected data
- Equivalently, we can think of PCA as learning a rotation of the canonical axes to align with these principal components, while the data cloud is fixed
- Nice effect of this transformation is that the covariance matrix of the transformed data is diagonal

Aside: Sample Covariance matrix

- Representation of the data that contains second-order statistics
- Let X be the matrix of features vs examples, so each column represents one data point. Assume data is centered around origin
- Form empirical covariance matrix $C = \frac{1}{n}XX^T$

$$C_{i,j} = \frac{1}{n}\sum_{k=1}^{n} x_k(i)x_k(j)$$

- Interpretation: $C_{i,j}$ gives an estimate of correlation between features i, j that we observe in the data
- When $C_{i,j}$ is close to 0, i and j are uncorrelated.
- When magnitude of $C_{i,j}$ is large, i and j tend to vary in tandem, or inversely

Principal Components Analysis: Intuition

- Diagonal entries of C denote variance of each attribute.
  - Assumption: Large values of variance are more interesting, and we want to preserve them.

- Off-diagonal entries of C denote covariances of each attribute.
  - Assumption: Large magnitudes on $C_{ij}, i \neq j$ denote redundancy between i and j
  - e.g. mean weekly rainfall, mean daily rainfall

Principal Components Analysis: Intuition

Why is a diagonal covariance matrix desirable?

- Diagonal covariance matrix allows us to see exactly how much variance each feature contains
- Diagonal covariance matrix means features are decorrelated
- Gives us a principled way to perform dimensionality reduction by removing features of lowest variance!

Principal Components Analysis Formulation

- Present a formulation which leads to solution in a simple way (though it is not from first principles)

- PCA problem: Find an orthonormal matrix W such that $Y = WX$ has a diagonal covariance matrix $\frac{1}{n}YY^T$, where the variances are ordered from largest to smallest

- Then, rows of W give us the principal components

Principal Components Analysis Formulation

- Fact: Any symmetric matrix M can be represented as $M = Q\Lambda Q^T$, where Q is orthonormal and Λ is diagonal.
- Columns of Q are the eigenvectors of X and the diagonal entries of Λ are associated eigenvalues

- Since $C = XX^T$ is symmetric, we can express $\frac{1}{n}YY^T$ as

$$\begin{aligned}
\frac{1}{n}YY^T &= \frac{1}{n}(WX)(WX)^T \\
&= \frac{1}{n}WXX^TW^T \\
&= WCW^T \\
&= WQ\Lambda QW^T
\end{aligned}$$

- Setting W $= Q^T$ gives $\frac{1}{n}YY^T = Q^TQ\Lambda Q^TQ = I\Lambda I = \Lambda$ which is diagonal as desired.

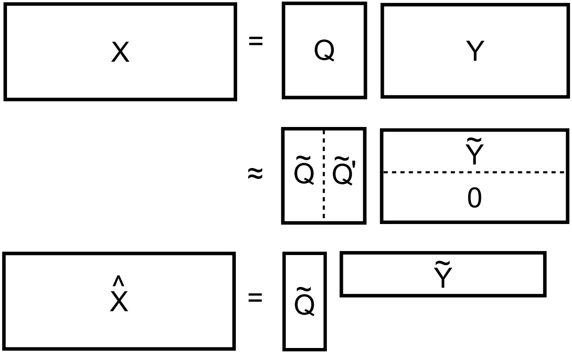
PCA: Matrix Factorization Viewpoint

- So far, the PCA solution is $Y = Q^T X$, where $Q = [q_1, q_2 \cdots q_n]$ is a matrix of principal components, and Λ is a diagonal matrix of corresponding eigenvalues ordered from largest to smallest
- Let's consider a single data point $y = Q^T x$
- Then the ith coordinate $y(i) = q_i^T x$ is the orthogonal projection of x onto the ith principal component.
- To perform dimensionality reduction, discard the bottom features of y, as they correspond to directions of smallest variance

PCA: Matrix Factorization Viewpoint

- If we retain all values of Y, We can recover X perfectly as $X = (Q^T)^{-1}Y = QY$ (property of orthonormal matrix Q)
- However, what if we removed the k features of Y corresponding to smallest variance
- Then, we can only recover $\hat{X} = Q\hat{Y}$ where $\hat{Y} = Y$ with the k features with smallest variance set to 0

PCA: Matrix Factorization Viewpoint



- It turns out, approximation of X in this way by the top k components of Y minimizes $\|X - \hat{X}\|_F^2$ over *all* rank-k matrices $\hat{X}$

Singular Value Decomposition

- Any matrix $X \in R^{m \times n}$ can be represented as

$$X = USV^T$$

  where $U \in R^{m \times m}$ and $V \in R^{n \times n}$ are orthonormal and $S \in R^{m \times n}$ is nonzero only on the diagonal
- We can obtain the PCA solution from SVD, without having to form the covariance matrix $\frac{1}{n}XX^T$ explicitly
- Popular languages such as R, Matlab, Python have SVD routines.

Link between PCA and SVD

- Substitute $X = USV^T$ in the equation for the covariance matrix.
- Then, we obtain

$$XX^T = USV^T VS^T U = U(SS^T)U$$

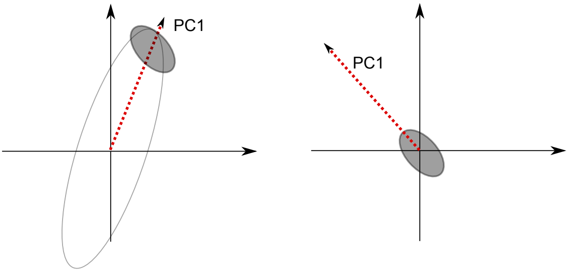
  where (SS^T) is diagonal
- We can see immediately that $U(SS^T)U$ has the same structure as $Q\Lambda Q^T$
- Therefore, the U matrix in the SVD is exactly the Q matrix we are trying to obtain in the PCA problem (up to sign changes on eigenvectors)

PCA: Considerations

- **How many components to keep?**
- Typically, retain enough dimensions to explain large proportion of the variance (common heuristic is 95%)
- The total variance is exactly the sum of the eigenvalues of $\frac{1}{n}XX^T$
- Keep adding successive PCs until the cumulative sum of respective eigenvalues hits the desired proportion

PCA: Considerations

- **Is it necessary to preprocess or scale the data?**
- Should always make data zero mean, otherwise the PCs found will not be what is expected



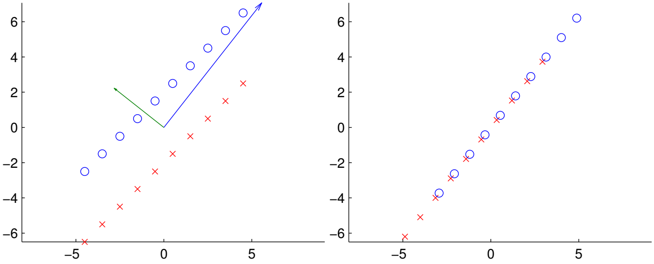
PCA: Considerations

- **Is it necessary to preprocess or scale the data?**
- Scaling features arbitrarily (for example converting height in feet to inches) skews the PCs considerably
- In general, if features are of different orders of magnitude, or different units, usually scale each feature to zero mean and unit standard deviation (z-scoring)
- If features are on same scale (e.g. exam scores for different subjects) then no scaling is needed
- Much discussion about the issue

PCA: Limitations

- Depending on task, the strongly predictive information may lie in directions of small variance, which gets removed by PCA
  - e.g. predicting size of clothes purchased, using income, taxes, height
- Solution: Use *supervised* dimensionality reduction techniques (e.g. distance metric learning algorithms) which retain keeping features useful for a (classification/regression) task
  - Tradeoff: computational and implementational complexity.

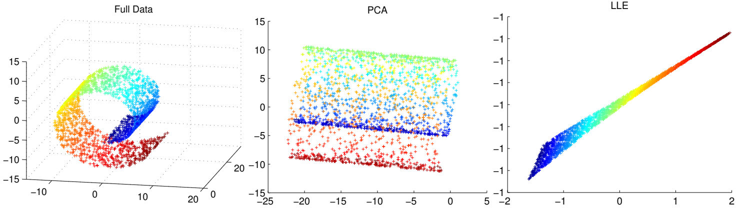
PCA: Limitations



- Blue and red dots are data points from different classes
- Without class information, 1-dimensional PCA projection shown on right
- Classes are separable with both features but not after projection

PCA: Limitations

- PCA finds linear projections
  - If data lies near or on a non-linear manifold (e.g. swiss roll), linear projection may not preserve distances along manifold
  - Solution: Use *manifold learning* techniques such as Locally Linear Embedding (LLE) [1], which may be able to learn better projections for the data



[1]Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding.
SCIENCE, 290:2323–2326, 2000

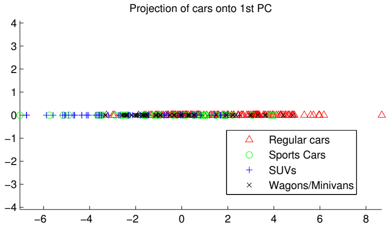
Principal Components Analysis Worked Example

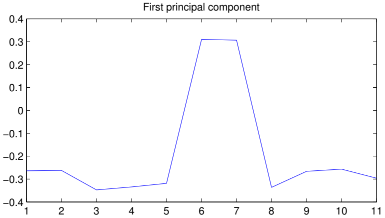
- For a concrete example, let's use the 2004 `Cars` dataset from the Journal of Statistics Education
- Dataset comprises different models of cars and their features
- Each car described by 11 features
- Matlab code only takes 4 lines (the `svd` function already sorts PC by eigenvalue

```
load cars
X = zscore(X')';
[PC Sigma V] = svd(X); %PC = principal components
Y = PC'*X;
```

Principal Components Analysis Worked Example

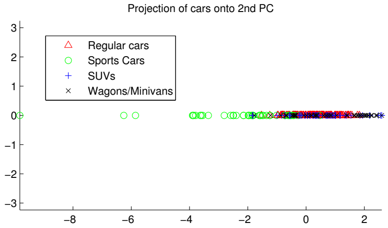
1	Retail Price
2	Dealer Cost
3	Engine Size
4	#Cylinders
5	Horsepower
6	City MPG
7	Highway MPG
8	Weight
9	Wheelbase
10	Length
11	Width

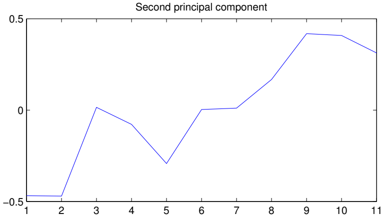




Principal Components Analysis Worked Example

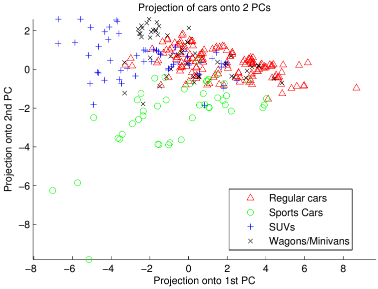
1	Retail Price	7	Highway MPG
2	Dealer Cost	8	Weight
3	Engine Size	9	Wheelbase
4	#Cylinders	10	Length
5	Horsepower	11	Width
6	City MPG		

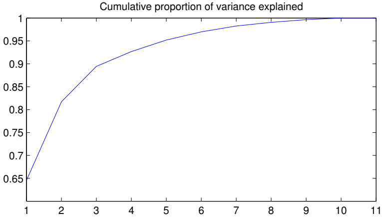




Principal Components Analysis Worked Example

1	Retail Price
2	Dealer Cost
3	Engine Size
4	#Cylinders
5	Horsepower
6	City MPG
7	Highway MPG
8	Weight
9	Wheelbase
10	Length
11	Width





Non-negative Matrix Factorization

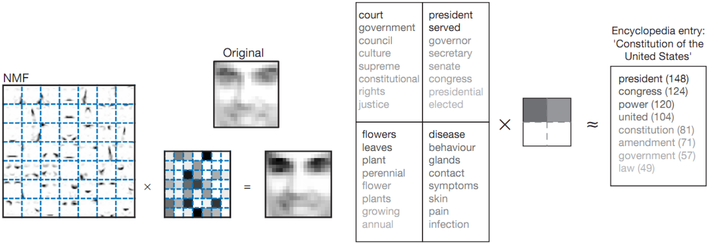
- Another popular matrix factorization method
- We want to minimize the norm $\|X - WY\|_F^2$, where $W \in R^{m \times k}, \ Y \in R^{k \times n}$
- So far, exactly the same as PCA
- As the name suggests, all matrices X, W and Y must contain only *non-negative* elements.

Non-negative Matrix Factorization

What does NMF do?

- Coefficient vectors Y are constrained to be non-negative
- Learned columns of W tend to be semantic features which can be combined additively to reconstruct data.
- NMF on encyclopedia entries produces documents containing words about a single topic
- NMF on facial images produces images containing facial parts

Non-negative Matrix Factorization



[2]

[2] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization.
Nature, 401(6755):788–791, October 1999

Non-negative Matrix Factorization

Learning NMF

- NMF can only find a local minimum of the error as the problem is nonconvex
- However, it can be minimized by a simple multiplicative form

$$Y_{ij} \leftarrow \mathcal{Y}_{ij} \frac{(W^T X)_{ij}}{(W^T W Y)_{ij}}, \quad W_{ij} \leftarrow W_{ij} \frac{(XY^T)_{ij}}{(XYY^T)_{ij}}$$

  and iterating until convergence
- Can be implemented in fewer than 10 lines

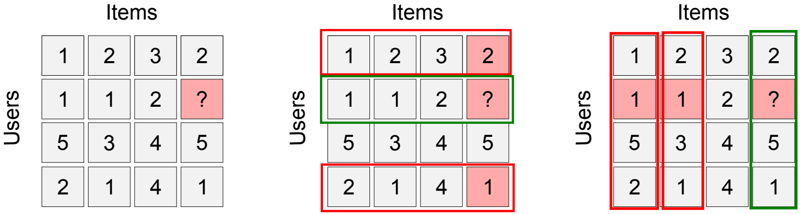
Collaborative Filtering Algorithms

- Given a list of users and items, and user-item interactions, predict user behavior
- What do we want to predict?
- **This talk:** Predict unobserved ratings of item j for user i
- Recent work on predicting rankings of unrated/unpurchased items for each user

Collaborative Filtering Algorithms

- **Problem: Predict score/affinity of item j for user i.**
- **User-based approach**
  - Find a set of users S_i who rated item j, that are most similar to u_i
  - compute predicted V_{ij} score as a function of ratings of item j given by S_i (usually weighted linear combination)
- **Item-based approach**
  - Find a set of most similar items S_j to the item j which were rated by u_i
  - compute predicted V_{ij} score as a function of u_i's ratings for S_j

Collaborative Filtering Algorithms



How to compute similarity between users/items?

- Cosine similarity

$$s(x, y) = \frac{x^T y}{\|x\| \|y\|}$$

and its variants (e.g. adding weights/biases)

Matrix Factorization for Collaborative Filtering - Model

- Let's consider items for sale on an e-commerce site
- Represent user i as a vector of *preferences* for a few factors p_i
  - e.g. "ease of use", "value for money", "aesthetic appeal"
  - $p_i = [0.7\ 0.1\ 0.6]$ = user i likes items which are easy to use and look nice, and doesn't mind paying for it

Matrix Factorization for Collaborative Filtering - Model

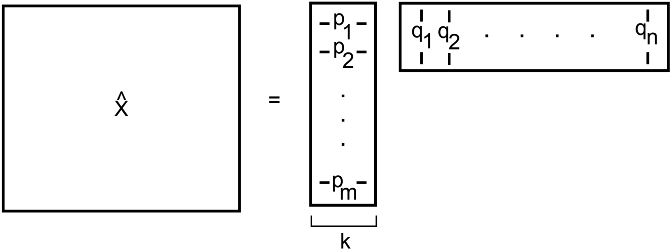
- Represent item j as a vector q_j where each element expresses how much the item exhibits that factor
- Rating of an item is estimated by $p_i^T q_j$
  - Intuition: if there is high correlation between the characteristics item exhibits that a user likes, he should give the item a high rating

Matrix Factorization for Collaborative Filtering - Model

- Problem: How to describe items with factors?
- Solution: Learn latent representation using the SVD
- Caveat: We don't learn semantic interpretations of the factors, just the numerical representations of the users/items
- However, for each factor, we can see which items have high/low scores, and assign human interpretations to that factor
- This is what Netflix does for movies, in their recommendation system (example factors: "Strong Female Lead", "Cerebral Suspense")

Matrix Factorization for Collaborative Filtering

- Assume we are given data $X \in \mathbb{R}^{m \times n}$ is a large matrix, comprising ratings of n movies by m users.
- Using our earlier model $(X_ij = p_i^T q_j)$, we can approximate X by $\hat{X} = PQ$, and rank $\hat{X} = k$, the number of factors



- For a given number of factors k, SVD gives us the optimal factorization which globally minimizes $\|X - \hat{X}\|_F^2$ the mean squared prediction error over all user-item pairs!

Limitations of SVD method for Collaborative Filtering

- SVD can only be applied if we know *all* the user-item ratings
  - e.g. Netflix Prize - only 1% of ratings given
- SVD is only able to minimize the (squared) Frobenius norm loss - may not be appropriate
- SVD is very slow for a large, dense user-item matrix

Approach 1 - Imputation

- Guess missing values of the matrix(imputation)
- Many approaches, such as mean imputation (across users or items), neighborhood-based imputation

However:

- Even if we had all imputed values, optimizing over all imputed values directly is computationally expensive
- Inappropriate imputations can cause considerable distortions in the data
- Doesn't allow for other loss functions or reduce computational complexity

Matrix Factorization for Collaborative Filtering

Approach 2 - Beyond SVD

- Minimize loss only over observed values, i.e. $\sum_{i,j}(X_{ij} - p_i^T q_j)$ where (i, j) is an observed user-item rating pair
  - Reduce computational complexity
  - Have to *regularize* P, Q to prevent overfitting

- Instead of using SVD, use methods like alternating least squares or stochastic gradient descent to update parameters
  - Allows more flexible model, and incorporating different loss functions (e.g. adding biases/temporal dynamics)
  - Drawback: Lots of parameters/hyperparameters to tune!

- Simon Funk "Try This at Home"

Performance on Netflix Dataset

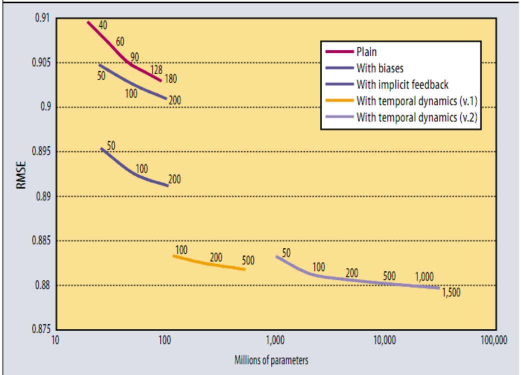


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more accurate. For comparison, the Netflix system achieves RMSE = 0.9514 on the same dataset, while the grand prize's required accuracy is RMSE = 0.8563. [3]

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009

Other approaches to collaborative filtering

- Collaborative Ranking
  - Instead of rating prediction, predict a ranked list of items user might find useful
  - Promising idea which can achieve state-of-the-art performance

- Nonnegative Matrix Factorization for Collaborative Filtering
  - NMF has also been used successfully to model movie ratings
  - Uses Expectation-Maximization approach to deal with missing values

Other approaches to collaborative filtering

- Hybrid recommendation systems
  - Combine content-based and collaborative filtering into a single model
  - Allows to integrate prior domain knowledge (e.g. information about items) into recommendation
  - Alleviates the cold-start problem for items with very few ratings

Conclusion

- Matrix factorizations are an important class of tools today
- Use spans dimensionality reduction, data understanding, prediction
- Simple methods like PCA can produce interesting insights of the data

Questions

Thank you!

Questions?