BUSI97287: Advanced Machine Learning

Dimension Reduction Techniques

Martin Haugh

Imperial College Business School
Email: martin.b.haugh@gmail.com

Additional References: Christopher Bishop's *PRML*, David Barber's *BRML*, Hastie, Tibshirani and Friedman's *ESL*, Hastie, Tibshirani and Wainwright's *SLS* and Chiang's *Networked Life*

Principal Components Analysis
- The Maximum Variance Formulation
- The Minimum Reconstruction Error Formulation

Some PCA Applications in Machine Learning & OR/FE
- Data Compression: Digit Recognition
- Data Compression: Eigen Faces
- Financial Modeling
- Nearest-Neighbor Classification
- Latent Semantic Analysis (LSA)
- Information Retrieval

Matrix Completion and Collaborative Filtering
- Baseline Estimators
- Neighborhood Methods
- Matrix Factorization (Revisited)
- The Netflix Prize

Principal Components Analysis

Let $\mathbf{x} = [x_1 \ \ldots \ x_d]^\top$ denote a d-dimensional random vector with variance-covariance matrix, $\mathbf{\Sigma}$.

The goal of PCA is to construct linear combinations

$$p_i \ = \ \sum_{j=1}^{d} w_{ij} \, x_j \, , \quad \text{for } i = 1, \ldots, d$$

in such a way that:

(1) The p_i's are orthogonal so that $\mathsf{E}[p_i \, p_j] = 0$ for $i \neq j$

(2) The p_i's are ordered in such a way that:

(i) p_1 explains the largest percentage of the total variance

(ii) each p_i explains the largest percentage of the total variance that has not already been explained by $p_1, \ldots, p_{i-1}$.

The Eigen Decomposition of a Matrix

In practice it is common to apply PCA to the normalized random variables so that $\mathbb{E}[x_j] = 0$ and $\text{Var}(x_j) = 1$ for $j = 1, \ldots, d$

- achieved by subtracting the means from the original random variables and dividing by their standard deviations
- done to ensure that no one component of **x** can influence the analysis by virtue of that component's measurement units.

Key tool of PCA is the eigen decomposition of a square matrix ...

The Eigen Decomposition of a Matrix

Eigen decomposition implies any symmetric matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ can be written as

$$\mathbf{A} \;=\; \mathbf{\Gamma}\,\mathbf{\Delta}\,\mathbf{\Gamma}^{\top} \tag{1}$$

where:

(i) $\mathbf{\Delta}$ is a diagonal matrix, $\text{diag}(\lambda_1, \ldots, \lambda_d)$, of the eigen values of $\mathbf{A}$.

- Without loss of generality eigen values are ordered so that

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d.$$

(ii) $\mathbf{\Gamma}$ is an orthogonal matrix with i^{th} column of $\mathbf{\Gamma}$ containing i^{th} standardized eigen-vector, $\boldsymbol{\gamma}_i$, of $\mathbf{A}$.

- "Standardized" means $\boldsymbol{\gamma}_i^{\top}\boldsymbol{\gamma}_i = 1$.
- Orthogonality of $\mathbf{\Gamma}$ implies $\mathbf{\Gamma}\,\mathbf{\Gamma}^{\top} \;=\; \mathbf{\Gamma}^{\top}\,\mathbf{\Gamma} \;=\; \mathbf{I}_d$.

PCA and the Factor Loadings

Since $\mathbf{\Sigma}$ is symmetric can take $\mathbf{A} = \mathbf{\Sigma}$ in (1). The positive semi-definiteness of $\mathbf{\Sigma}$ implies $\lambda_i \geq 0$ for all $i = 1, \ldots, d$.

Principal components of $\mathbf{x}$ then given by $\mathbf{p} = (p_1, \ldots, p_d)$ satisfying

$$\mathbf{p} = \mathbf{\Gamma}^\top \mathbf{x}. \tag{2}$$

Note that:

(a) $\mathsf{E}[\mathbf{p}] = \mathbf{0}$ since $\mathsf{E}[\mathbf{x}] = \mathbf{0}$

(b) $\mathsf{Cov}(\mathbf{p}) = \mathbf{\Gamma}^\top \mathbf{\Sigma}\, \mathbf{\Gamma} = \mathbf{\Gamma}^\top (\mathbf{\Gamma}\, \mathbf{\Delta}\, \mathbf{\Gamma}^\top)\, \mathbf{\Gamma} = \mathbf{\Delta}$

- so components of $\mathbf{p}$ are uncorrelated as desired
- and $\mathsf{Var}(p_i) = \lambda_i$.

The matrix $\mathbf{\Gamma}^\top$ is called the matrix of factor loadings. Can invert (2) to obtain

$$\mathbf{x} = \mathbf{\Gamma}\, \mathbf{p} \tag{3}$$

- so easy to go back and forth between $\mathbf{x}$ and $\mathbf{p}$.

Explaining The Total Variance

Can measure ability of first few principal components to explain total variance:

$$\begin{aligned} \sum_{i=1}^{d} \mathrm{Var}(p_i) &= \sum_{i=1}^{d} \lambda_i \\ &= \mathrm{trace}(\Sigma) \\ &= \sum_{i=1}^{d} \mathrm{Var}(x_i). \end{aligned} \tag{4}$$

If we take $\sum_{i=1}^{d} \mathrm{Var}(p_i) = \sum_{i=1}^{d} \mathrm{Var}(x_i)$ to measure the total variability then by (4) can interpret

$$\frac{\sum_{k=1}^{K} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

as the % of total variability explained by first K principal components.

Approximating x in a Low-Dimensional Sub-Space

Know from (3) that

$$\begin{aligned} \mathbf{x} &= \mathbf{\Gamma}\,\mathbf{p} \\ &= \mathbf{\Gamma}_{1:K}\,\mathbf{p}_{1:K} + \mathbf{\Gamma}_{K+1:d}\,\mathbf{p}_{K+1:d} \end{aligned} \tag{5}$$

where:

- $\mathbf{p}_{1:K}$ denotes first K components of $\mathbf{p}$
- $\mathbf{\Gamma}_{1:K}$ denotes first K columns of $\mathbf{\Gamma}$.

If data has been centered (so $\mathbb{E}[p_j] = 0$) and $\sum_{k=1}^{K} \lambda_i \approx \sum_{i=1}^{d} \lambda_i$ then by (5)

$$\mathbf{x} \approx \mathbf{\Gamma}_{1:K}\,\mathbf{p}_{1:K}. \tag{6}$$

If K small then $\mathbf{p}_{1:K} \in \mathbb{R}^K$ is a low-dimensional representation of $\mathbf{x} \in \mathbb{R}^d$.

Remark: When using (6), $\mathbf{x}$ and $\mathbf{p}$ vary but $\mathbf{\Gamma}_{1:K}$ does not. So for the i^{th} data-point we would have $\mathbf{x}^{(i)} \approx \mathbf{\Gamma}_{1:K}\,\mathbf{p}^{(i)}_{1:K}$.

The Maximum Variance Formulation

Let $p_1 = \boldsymbol{\gamma}_1^\top \mathbf{x}$ be the 1^{st} principal component. Can easily show that $\boldsymbol{\gamma}_1$ solves

$$\begin{aligned} \max_{\mathbf{a}} \quad & \text{Var}(\mathbf{a}^\top \mathbf{x}) \\ \text{subject to} \quad & \mathbf{a}^\top \mathbf{a} = 1. \end{aligned}$$

More generally, let $p_i = \boldsymbol{\gamma}_i^\top \mathbf{x}$ be the i^{th} principal component for $i = 1, \ldots, d$.

Then $\boldsymbol{\gamma}_i$ solves

$$\begin{aligned} \max_{\mathbf{a}} \quad & \text{Var}(\mathbf{a}^\top \mathbf{x}) \\ \text{subject to} \quad & \mathbf{a}^\top \mathbf{a} = 1 && (7) \\ & \mathbf{a}^\top \boldsymbol{\gamma}_j = 0, \quad j = 1, \ldots, i-1. && (8) \end{aligned}$$

So each successive principal component finds the linear combination of the components of $\mathbf{x}$ that has maximal variance subject to the normalization constraint (7) and the orthogonal constraints (8).

An Aside: The Minimum Reconstruction Formulation

Can also obtain the PCs of $\mathbf{x}$ by posing the problem as one of minimizing the sum of squared differences between each sample vector, $\mathbf{x}_i$ and it's representation, $\tilde{\mathbf{x}}_i$, in some lower K-dimensional sub-space

- here we let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ denote (centered) sample observations of $\mathbf{x}$.

Have the following problem:

$$\begin{aligned} & \min_{\mathbf{B},\mathbf{Z}} \sum_{i=1}^{n} \sum_{j=1}^{d} \left[x_{j,i} - \sum_{k=1}^{K} b_{j,k} z_{k,i} \right]^2 \\ \equiv \quad & \min_{\mathbf{B},\mathbf{Z}} \|\mathbf{X} - \mathbf{BZ}\|_F^2 \end{aligned} \tag{9}$$

where:

- $\mathbf{X}$ is the $d \times n$ matrix whose i^{th} column is $\mathbf{x}_i$
- $\mathbf{B}$ is $d \times K$ with $(j,k)^{th}$ element $b_{j,k}$
- $\mathbf{Z}$ is $K \times n$ with $(k,i)^{th}$ element $z_{k,i}$

An Aside: The Minimum Reconstruction Formulation

So (9) approximates each $\mathbf{x}_i$ with $\tilde{\mathbf{x}}_i \approx \sum_{k=1}^{K} z_{k,i}\, \mathbf{b}_k$ where $\mathbf{b}_k$ is the k^{th} column of $\mathbf{B}$.

There are many optimal solutions to (9) but one optimal solution is:

$$\begin{aligned} \mathbf{B}^* &= \mathbf{\Gamma}_{1:K} \\ \mathbf{Z}^* &= \mathbf{P}_{1:K} \end{aligned}$$

where the i^{th} column of $\mathbf{P}_{1:K}$ is $\mathbf{p}_{1:K}^{(i)}$ from slide 8.

So we recover (6) as the optimal approximation!

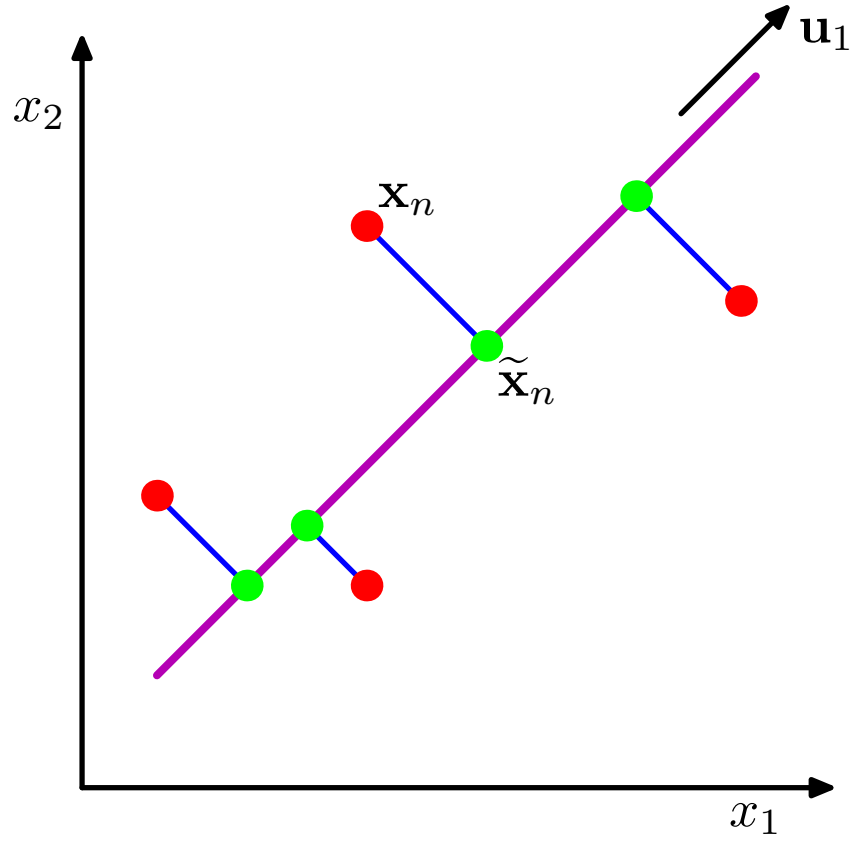


Figure 12.2 from Bishop: Principal component analysis seeks a space of lower dimensionality, known as the principal subspace and denoted by the magenta line, such that the orthogonal projection of the data points (red dots) onto this subspace maximizes the variance of the projected points (green dots). An alternative definition of PCA is based on minimizing the sum-of-squares of the projection errors, indicated by the blue lines.

Some PCA Applications in Machine Learning & OR/FE

Will consider several application domains for PCA:

1. Data compression
   - hand-writing
   - eigen-faces
2. Financial modeling
3. Nearest neighbor classification
4. Latent semantic analysis
5. Information retrieval
6. Missing data and collaborative filtering / recommender systems.

There are many, many more application domains.

Data Compression: Digit Recognition

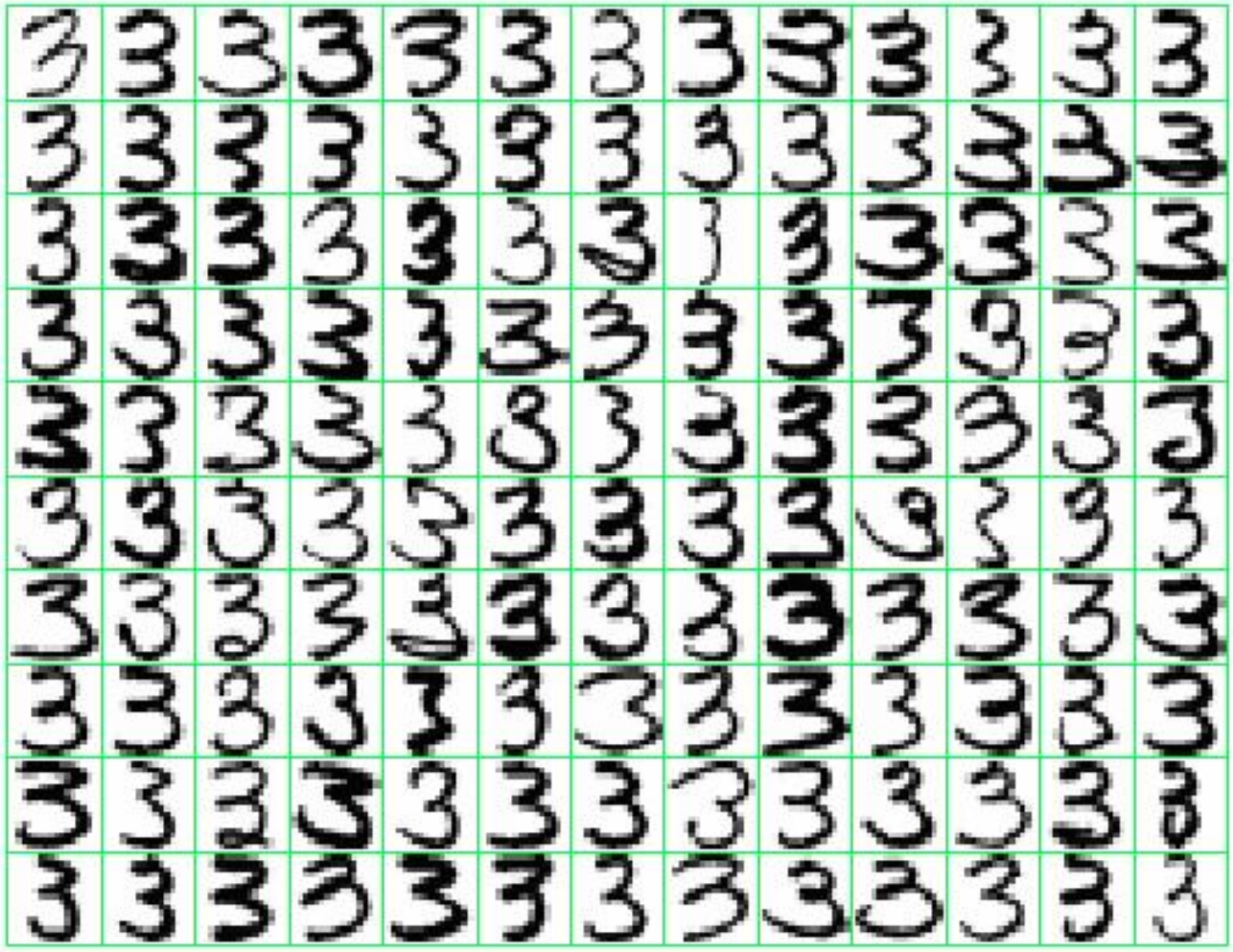


Fig. 14.22 from HTF: A sample of 130 handwritten 3's shows a variety of writing styles.

Images digitized as 16×16 grayscale images so can view them as points in $\mathbb{R}^{256}$.

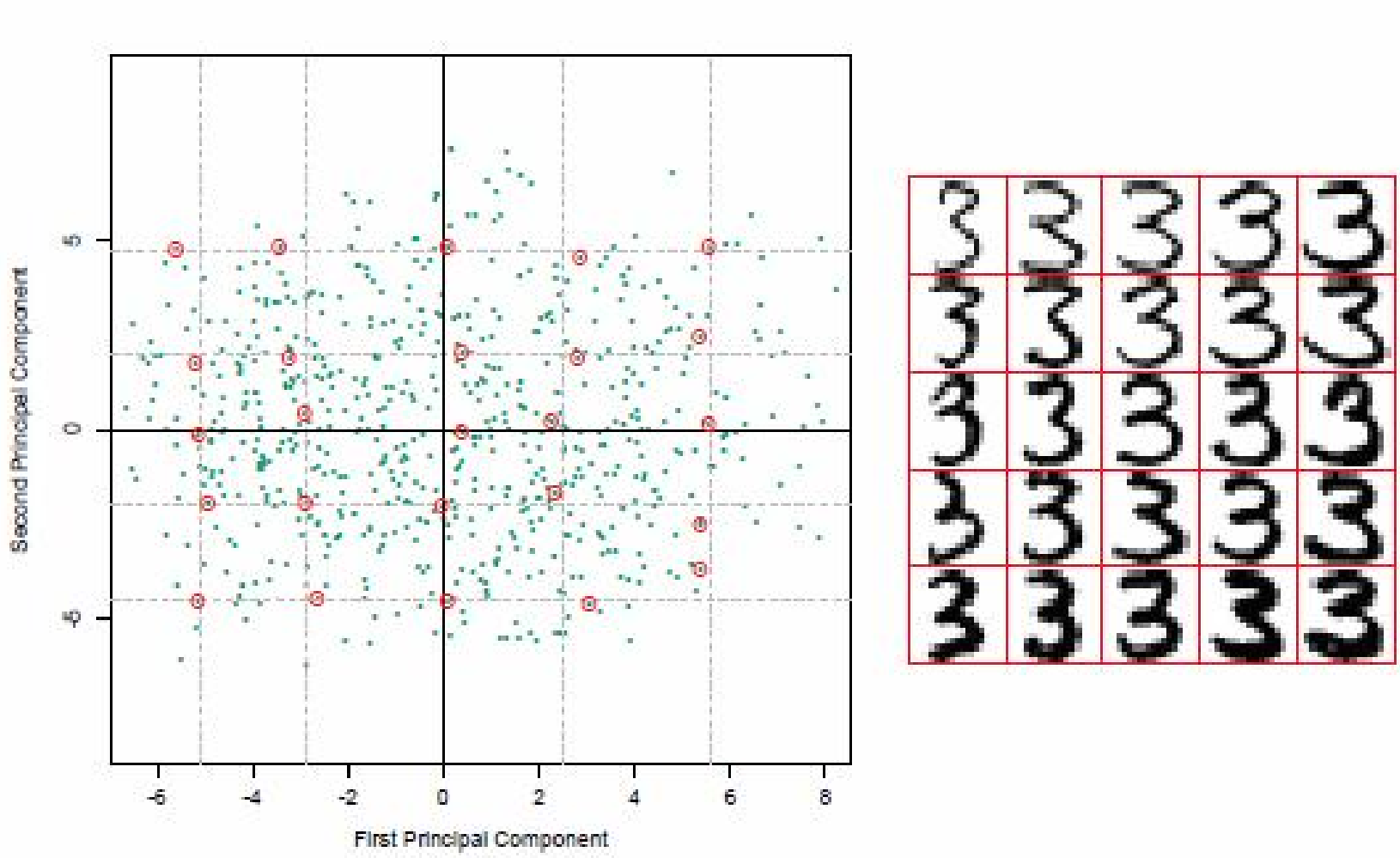


Figure 14.23 from HTF: Left panel: the first two principal components of the hand-written threes. The circled points are the closest projected images to the vertices of a grid, defined by the marginal quantiles of the principal components. Right panel: The images corresponding to the circled points. These show the nature of the first two principal components.

Data Compression: Digit Recognition

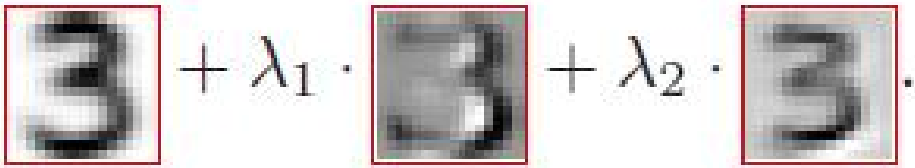
Figure 14.23 from HTF plots the first 2 principal components along with images that are close to the vertices of the grid

- vertices are placed at 5%, 25%, 50%, 75% and 95% quantile points.

- First component accounts for "lengthening of the lower tail of the three".
- Second component accounts for "character thickness".

Equation (14.55) from HTF yields the 2-component model

$$\begin{aligned} \hat{f}(\lambda) &= \bar{x} + \lambda_1 v_1 + \lambda_2 v_2 \\ &= \end{aligned}$$



(In our notation we equate $\lambda_i \equiv p_i$ and $v_i \equiv \boldsymbol{\gamma}_i$.)

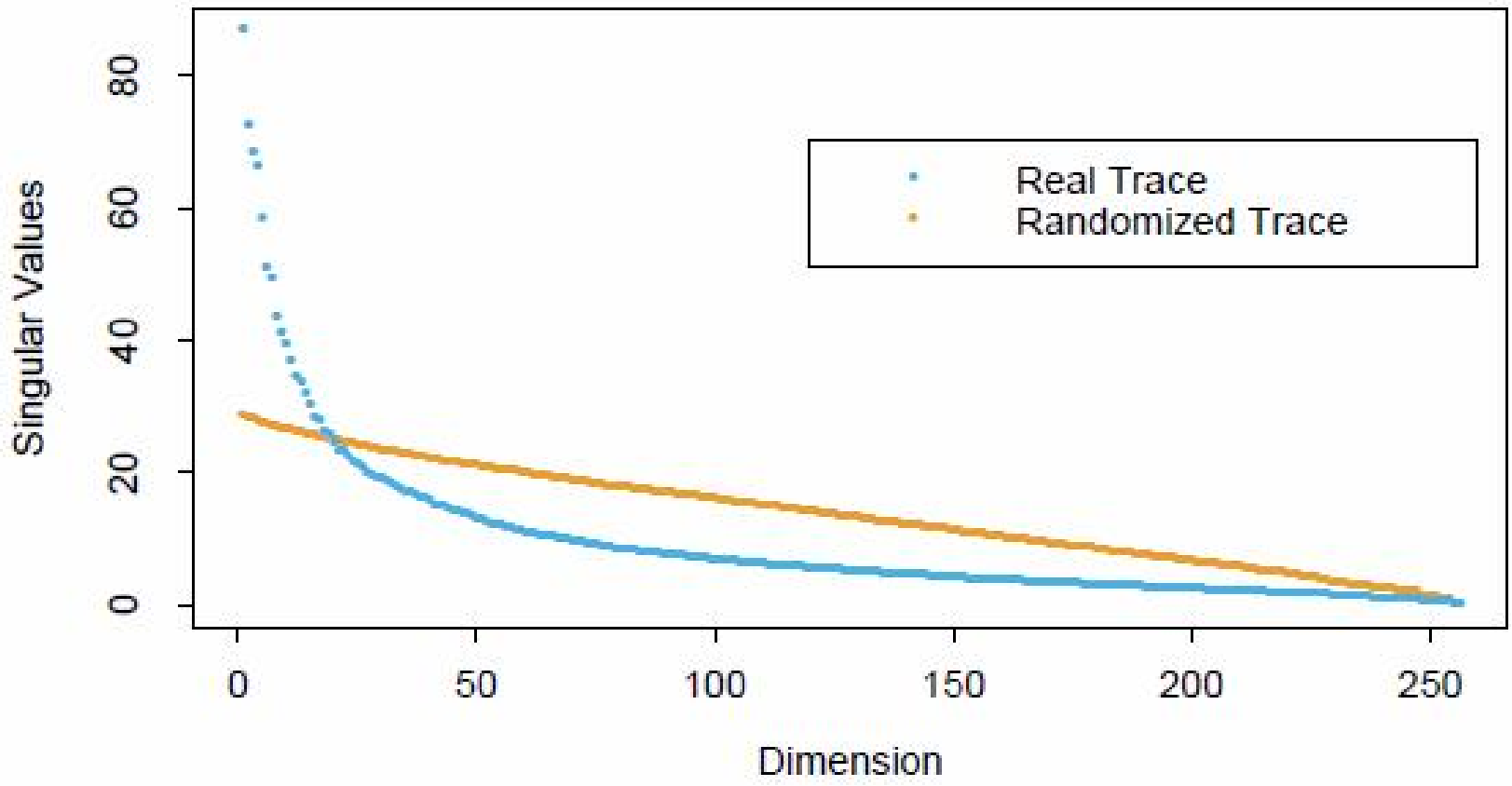


Figure 14.24 from HTF: The 256 singular values for the digitized threes, compared to those for a randomized version of the data (each column of **X** was scrambled).

Note: The eigen values are the squared singular values (from the SVD).

Data Compression: Eigen Faces

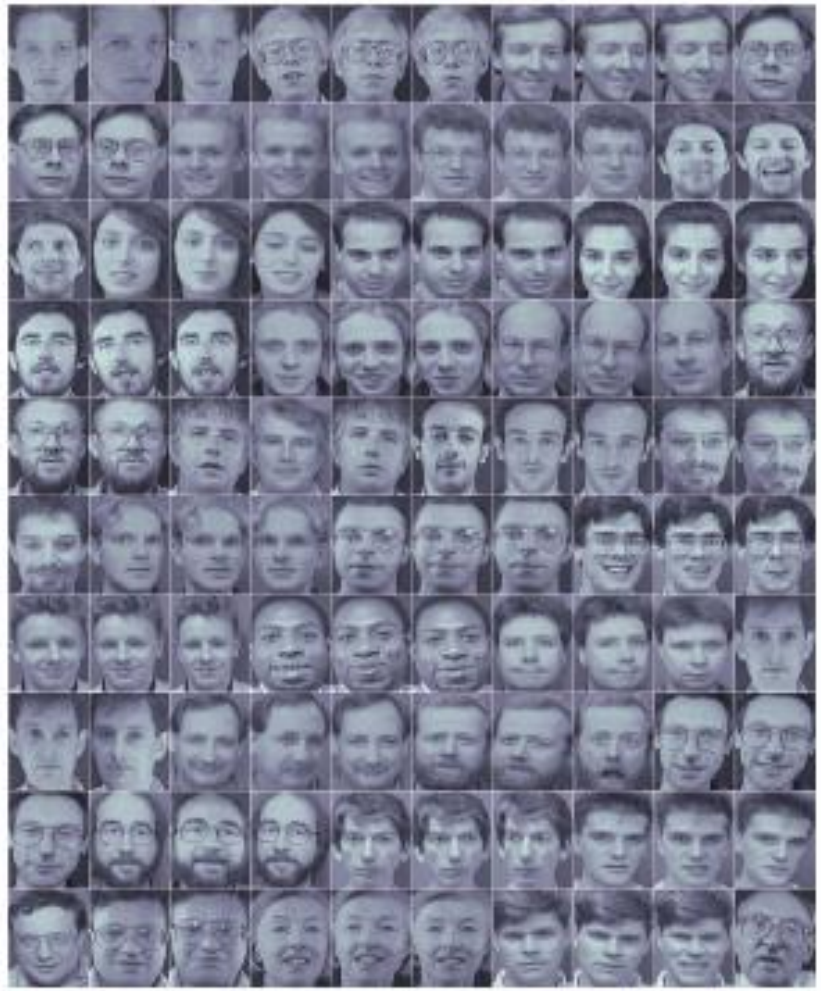


Figure 15.5 from Barber: 100 training images. Each image consists of $92 \times 112 = 10304$ greyscale pixels. The train data is scaled so that, represented as an image, the components of each image sum to 1. The average value of each pixel across all images is 9.70×10^{-5}. This is a subset of the 400 images in the full Olivetti Research Face Database.

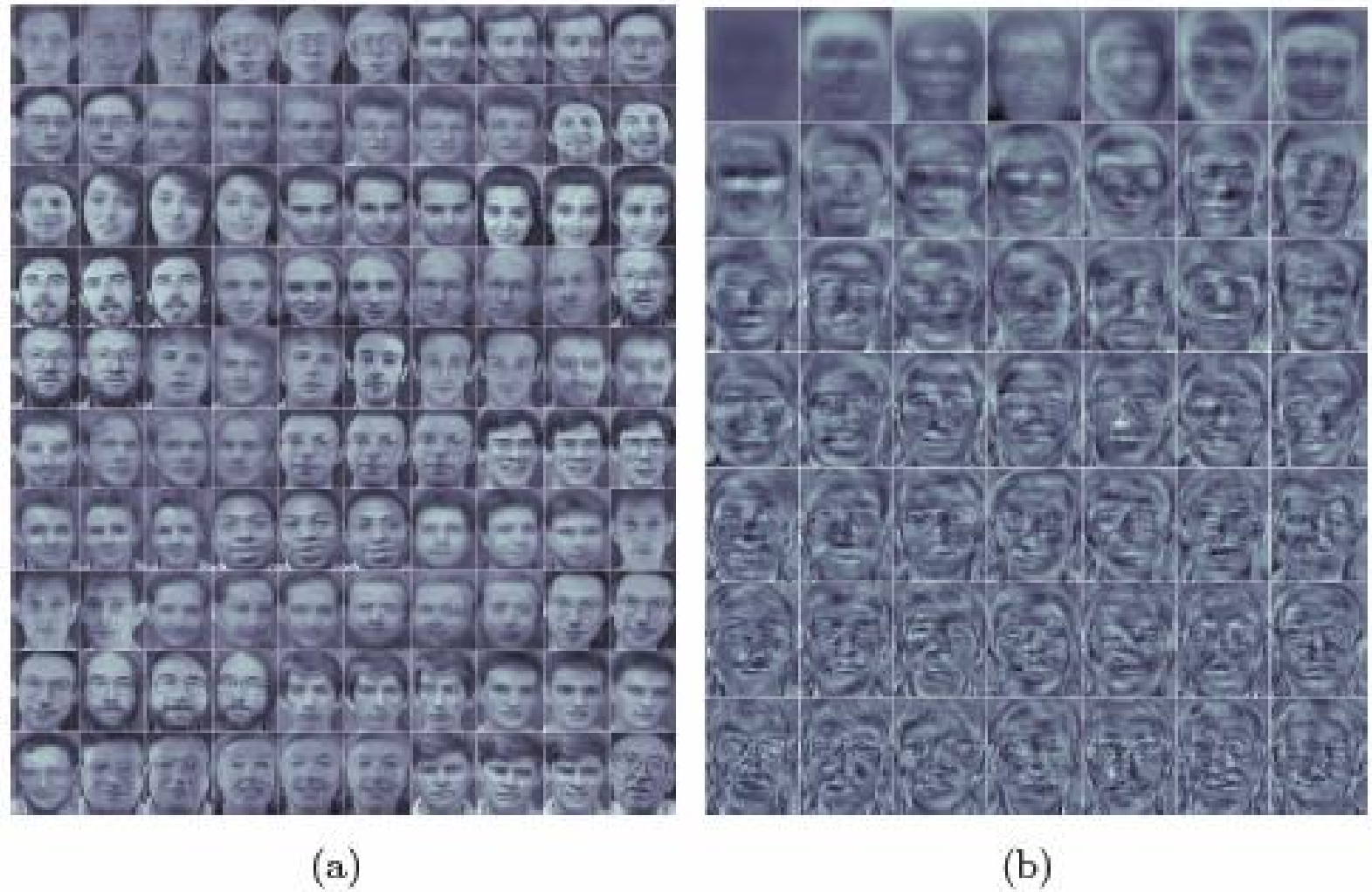


Figure 15.6 from Barber: (a): SVD reconstruction of the images in fig(15.5) using a combination of the 49 eigen-images. (b): The eigen-images are found using SVD of the images in fig(15.5) and taking the mean and 48 eigenvectors with largest corresponding eigenvalue. The images corresponding to the largest eigenvalues are contained in the first row, and the next 7 in the row below, etc. The root mean square reconstruction error is 1.121×10^{-5}, a small improvement over PLSA (see fig(15.16)).

PCA and Financial Modeling

There are many applications of PCA in finance:

1. Representing movements in term-structure of interest-rates
   - useful for interpretation
   - building factor models
   - and hedging.
2. Representing movements in futures strips
   - useful for interpretation
   - building factor models
   - and hedging.
3. Building factor models for equities.
4. Scenario generation in risk-management.
5. Estimation of risk measures such as VaR and CVaR.
6. And others ...

Nearest-Neighbor Classification

In nearest-neighbor classification it can be expensive to compute the distance between data-points when dimensionality is high.

Can overcome this problem by using PCA to approximate distances.

Let $\mathbf{\Gamma}_1$ contain the first K principal components with k large enough so that

$$\mathbf{x} = \mathbf{\Gamma}\mathbf{p} \approx \mathbf{\Gamma}_1\mathbf{p}_1$$

provides a good approximation. If $\mathbf{x}^a$ and $\mathbf{x}^b$ are any two points in $\mathbb{R}^d$ then

$$\begin{aligned}
\left(\mathbf{x}^a - \mathbf{x}^b\right)^\top \left(\mathbf{x}^a - \mathbf{x}^b\right) &= (\mathbf{\Gamma}\mathbf{p}^a - \mathbf{\Gamma}\mathbf{p}^b)^\top(\mathbf{\Gamma}\mathbf{p}^a - \mathbf{\Gamma}\mathbf{p}^b) \\
&\approx (\mathbf{\Gamma}_1\mathbf{p}_1^a - \mathbf{\Gamma}_1\mathbf{p}_1^b)^\top(\mathbf{\Gamma}_1\mathbf{p}_1^a - \mathbf{\Gamma}_1\mathbf{p}_1^b) \\
&= (\mathbf{p}_1^a - \mathbf{p}_1^b)^\top\mathbf{\Gamma}_1^\top\mathbf{\Gamma}_1(\mathbf{p}_1^a - \mathbf{p}_1^b) \\
&= (\mathbf{p}_1^a - \mathbf{p}_1^b)^\top(\mathbf{p}_1^a - \mathbf{p}_1^b). \qquad (10)
\end{aligned}$$

Can be much cheaper to compute (10)!

And data often noisy so projecting onto a lower dimensional sub-space can produce superior classification: see Figure 15.7 from Barber.

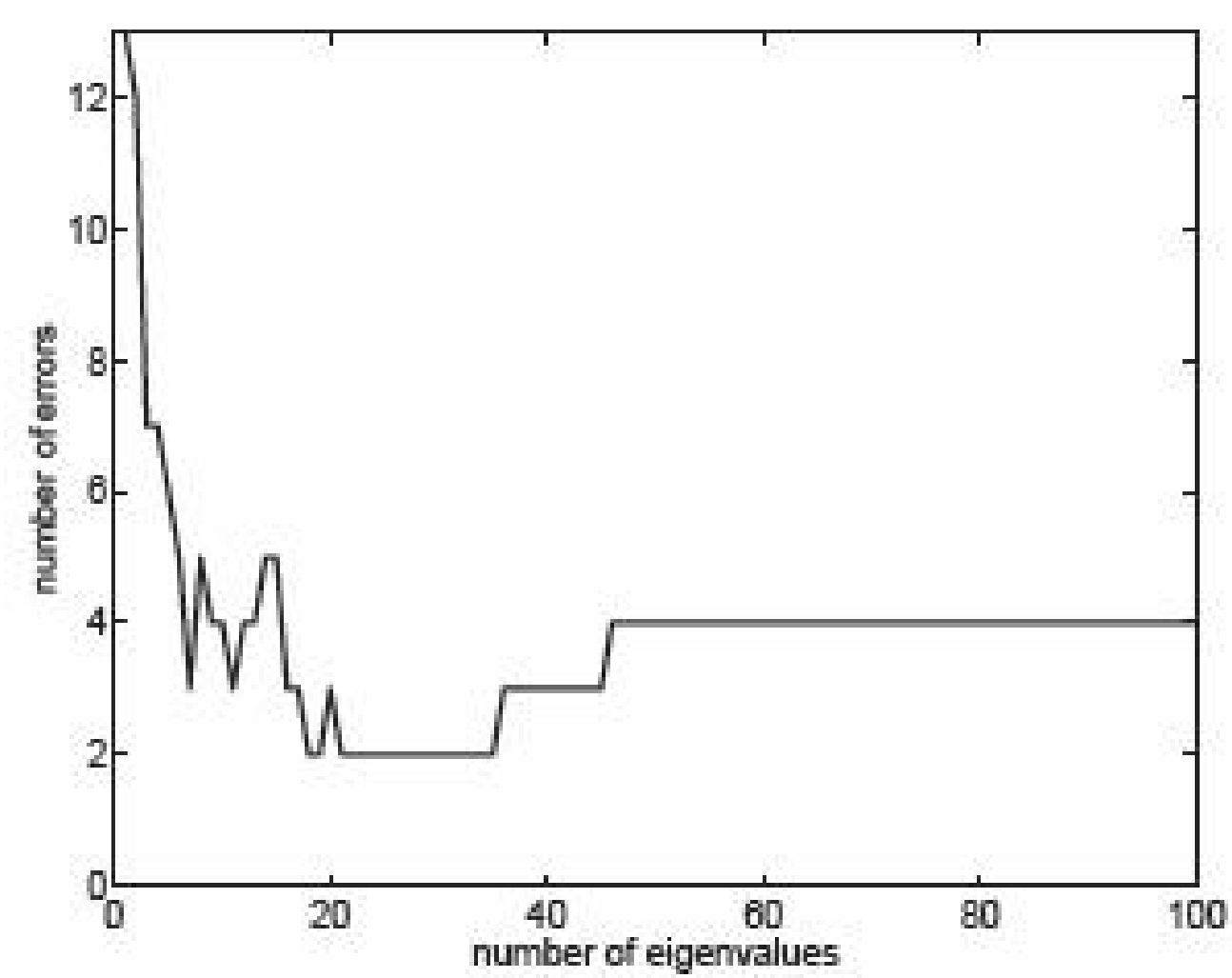


Figure 15.7 from Barber: Finding the optimal PCA dimension to use for classifying hand-written digits using nearest neighbours. 400 training examples are used, and the validation error plotted on 200 further examples. Based on the validation error, we see that a dimension of 19 is reasonable.

Latent Semantic Analysis (LSA)

In the document analysis literature, PCA is called latent semantic analysis (LSA).

We assume:

1. There are d words in our dictionary, $\mathcal{D}$
2. And n documents in the corpus.

The j^{th} document could then be represented by $\mathbf{x}^j = (x_1^j, \ldots, x_d^j)^\top$

- the so-called bag-of-words representation
- where x_i^j refers to the $\#$ occurrences of the i^{th} word in the j^{th} document.

But more common to normalize this number:

Definition. The term-frequency, tf_i^j, is the number of times that the word i appears in document j divided by the number of words in document j, i.e.

$$\text{tf}_i^j := \frac{\#_{i,j}}{\sum_i \#_{i,j}}$$

where $\#_{i,j}$ is the number of times that word i appears in document j.

The TF-IDF Representation

Definition. The inverse-document-frequency, idf_i, is defined as

$$\text{idf}_i \;:=\; \log\left(\frac{n}{\#\text{ of documents that contain word } i}\right).$$

Possible to use different definitions of idf_i

- as long as rarely occurring words are given more weight when they do occur.

Can now define the TF-IDF representation:

$$x_i^j \;:=\; \text{tf}_i^j \times \text{idf}_i$$

A corpus of documents is then represented by

$$\mathbf{X} \;=\; [\mathbf{x}^1 \cdots \mathbf{x}^n]$$

– a $d \times n$ matrix which is typically very large!

Latent Semantic Analysis (LSA)

LSA is simply a matter of applying PCA to the columns of $\mathbf{X}$

- with the interpretation that the principal directions, i.e. eigen vectors, now define topics.

One weakness of LSA is that eigen vectors can have negative components

- but how can a word contribute negatively to a "topic"?

Probabilistic latent semantic analysis (PLSA) overcomes this problem

- based on non-negative matrix factorization.

Topic modeling has been a "hot topic" in the machine learning community.

Example 15.4 from Barber

The dictionary, $\mathcal{D}$, contains 10 words:

`influenza, flu, headache, nose, temperature,`
`bed, cat, dog, rabbit, pet`

The document corpus contains 2000 documents:

1. some articles discuss ailments, and some of these discuss influenza
2. some articles related to pets
3. some articles are background articles unrelated to ailments

Some of the ailment articles are informal and use the word "`flu`" whereas others use the more formal "`influenza`".

Each document is therefore represented by a 10-dimensional vector

- $x_i^j = 1$ if i^{th} word occurs in j^{th} document
- $x_i^j = 0$ otherwise.

See Figures 15.8 and 15.9 from Barber.

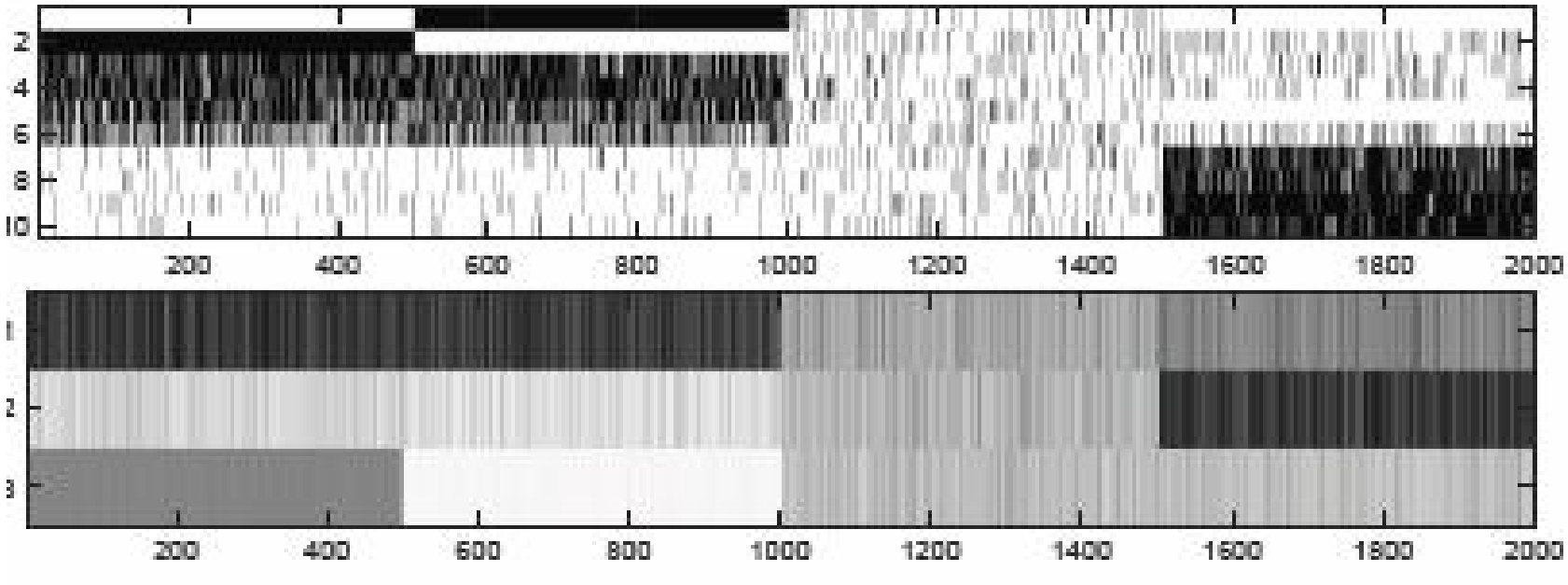


Figure 15.8 from Barber: (Top) Document data for a dictionary containing 10 words and 2000 documents. Black indicates that a word was present in a document. The data consists of two distinct topics and a random background topic. The first topic contains two sub-topics which differ only in their usage of the first two words, 'influenza' and 'flu'. (Bottom) The projections of each datapoint onto the three principal components.

Information Retrieval and Example 15.5 from Barber

Consider a large collection of documents from which a dictionary, $\mathcal{D}$, is created.

Given a document, $\mathbf{x}^f$, how do we find the "closest" document to it in the collection?

First need a measure of dissimilarity between documents

- could use $d(\mathbf{x}^f, \mathbf{x}^i) := (\mathbf{x}^f - \mathbf{x}^i)^\top (\mathbf{x}^f - \mathbf{x}^i)$.

And then select the document that solves

$$\min_i \; d(\mathbf{x}^f, \mathbf{x}^i).$$

In this case it's a good idea (why?) to scale each vector so that it has unit length

- leads to the equivalent cosine similarity

$$s(\mathbf{x}^f, \mathbf{x}^i) \;:=\; \cos(\theta)$$

where θ is the angle between $\mathbf{x}^f$ and $\mathbf{x}^i$.

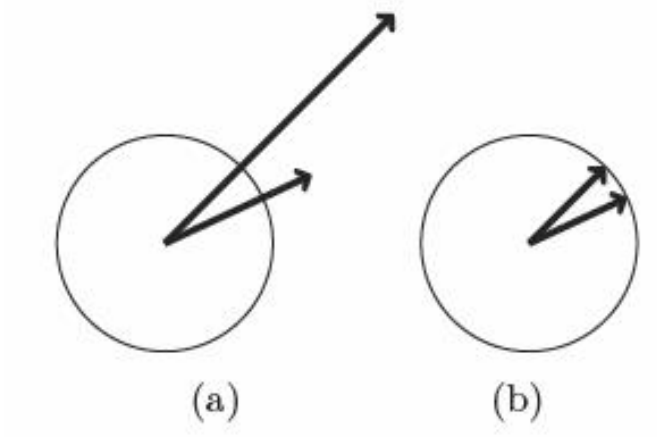


Figure 15.10 from Barber: (a): Two bag-of-word vectors. The Euclidean distance between the two is large. (b): Normalised vectors. The Euclidean distance is now related directly to the angle between the vectors. In this case two documents which have the same relative frequency of words will both have the same dissimilarity, even though the number of occurrences of the words is different.

Problem with bag-of-words representation is that TD matrix will have mainly zeros

- so differences may be due to noise.

LSA can help (why?) solve this problem

- consider Example 15.4 from Barber.

Matrix Completion and Collaborative Filtering

	Dirty Dancing	Meet the Parents	Top Gun	The Sixth Sense	Catch Me If You Can	The Royal Tenenbaums	Con Air	Big Fish	The Matrix	A Few Good Men
Customer 1					4					
Customer 2			3				3			3
Customer 3		2		4					2	
Customer 4	3									
Customer 5	5	5			4					
Customer 6						2	4			
Customer 7			5					3		
Customer 8						2				3
Customer 9	3				5			5		
Customer 10										

Table 7.2 from Hastie, Tibshirani and Wainwright's *Statistical Learning with Sparsity* (2015): Excerpt of the `Netflix` movie rating data. The movies are rated from 1 (worst) to 5 (best). A symbol shaded grey circles represents a missing value: a movie that was not rated by the corresponding customer.

Matrix Completion and Collaborative Filtering

In many applications we are missing many / most of the elements of a matrix $\mathbf{X}$.

e.g. Consider a $d \times n$ movies-ratings matrix with d movies and n users

- then x_{ui} represents the rating (on some scale) of user u for movie i
- $\mathbf{X}$ will then be very sparse in that very few of the x_{ui}'s will be known.

Goal then is to somehow "fill in" or impute the missing values

- has obvious applications in collaborative filtering and recommender systems
- **e.g.** the Netflix prize.

Two main approaches to solving this problem:

1. Neighborhood methods
2. Matrix factorization methods

Will discuss each of them but first we establish some baseline estimators.

Baseline Estimators

A couple of obvious baseline estimators:

1. Set $\hat{x}_{ui} = \bar{\mathbf{x}}$, average of all ratings in the (training) data-set.

2. Introduce biases and set

$$\hat{x}_{ui} = \bar{\mathbf{x}} + b_u + b_i \tag{11}$$

where

$$\begin{aligned} b_i &:= \frac{\sum_u x_{ui}}{M_i} - \bar{x} \\ b_u &:= \frac{\sum_i x_{ui}}{M_u} - \bar{x} \end{aligned}$$

and

- $M_i = \#$ users that rated movie i.
- $M_u = \#$ movies rated by user u.

Baseline Estimators

3. Use (11) but choose b_i's, b_u's via

$$\min_{b_i, b_u} \sum_{(u,i)} (x_{ui} - \hat{x}_{ui})^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \right).$$

Question: How might we choose λ?

(11) seems sensible (why?) so will henceforth work with the residual matrix $\tilde{\mathbf{X}}$

$$\begin{aligned} \tilde{\mathbf{X}}_{u,i} &:= \tilde{x}_{ui} \\ &:= x_{ui} - \hat{x}_{ui}. \end{aligned}$$

Neighborhood Methods

Will continue to use the movie-ratings matrix application to develop ideas.

Neighborhood methods rely on calculating nearest neighbors:

1. Two rows are near neighbors if corresponding users have similar taste in movies.
2. Two columns are near neighbors if the corresponding movies obtained similar ratings from users.

To compute nearest neighbors we need a distance / similarity metric.

Can use cosine similarity so

$$\begin{aligned} d_{ij} := \cos(\theta_{ij}) &= \frac{\tilde{\mathbf{x}}_i^\top \tilde{\mathbf{x}}_j}{\sqrt{||\tilde{\mathbf{x}}_i||_2 ||\tilde{\mathbf{x}}_j||_2}} \\ &= \frac{\sum_u \tilde{x}_{ui}\tilde{x}_{uj}}{\sqrt{\sum_u \tilde{x}_{ui}^2 \sum_u \tilde{x}_{uj}^2}} \end{aligned} \tag{12}$$

where summation is only over users u that rated both movies i and j.

Can define d_{uv} for users u and v similarly.

Neighborhood Methods

Given a fixed movie i can rank all other movies in descending order of $|d_{ij}|$.

Then top L movies in the ranking are the L nearest neighbors of movie i.

Yields new estimator

$$\hat{x}_{ui}^{N} = \bar{\mathbf{x}} + b_u + b_i + \underbrace{\frac{\sum_{j \in \mathcal{L}_i} d_{ij}\, \tilde{x}_{uj}}{\sum_{j \in \mathcal{L}_i} |d_{ij}|}}_{\text{collaborative filtering term}} \tag{13}$$

where $\mathcal{L}_i$ denotes the neighborhood of movie i.

Question: Why rank movies according to $|d_{ij}|$ rather than d_{ij} when defining $\mathcal{L}_i$?

Remarks:

1. Could just as easily use d_{uv}'s rather than d_{ij}'s and adjust (13) appropriately.
2. Still need to choose value of L and other obvious tweaks can be made.
3. Can use (13) to define new residual matrix $\tilde{\mathbf{X}}_{u,i}$
   - and then start working on predicting missing elements of $\tilde{\mathbf{X}}_{u,i}$.

Matrix Factorization Methods

General matrix factorization problem formulated as

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{M} \in \mathbb{R}^{d \times n}} ||\mathbf{X} - \mathbf{M}||_F^2 \tag{14}$$

subject to

$$\Phi(\mathbf{M}) \leq k \tag{15}$$

where $\Phi(\cdot)$ a constraint function used to encourage sparsity or regularization etc. and $||\cdot||_F$ denotes Frobenius norm:

$$||\mathbf{A}||_F^2 = \sum_{i=1}^{d} \sum_{j=1}^{n} a_{ij}^2 \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{d1} & \dots & a_{dn} \end{bmatrix}$$

Many interesting problems obtained by varying Φ.

PCA Revisited

e.g. Take $\Phi(\mathbf{M}) = \text{rank}(\mathbf{M})$. Then solution to (14) and (15) is

$$\hat{\mathbf{X}} = \mathbf{\Gamma}_{1:K}\, \mathbf{P}_{1:K} \tag{16}$$

where (16) is simply the r.h.s. of (6) in matrix form so that

$$\mathbf{\Gamma}_{1:K} = \begin{bmatrix} | & \cdots & | \\ \gamma_1 & \cdots & \gamma_K \\ | & \cdots & | \end{bmatrix}$$

and

$$\mathbf{P}_{1:K} = \begin{bmatrix} | & \cdots & | \\ \mathbf{p}_{1:K}^{(1)} & \cdots & \mathbf{p}_{1:K}^{(n)} \\ | & \cdots & | \end{bmatrix}.$$

In this case (14), (15) just an equivalent version of (9) once you know the following:

Fact: Any rank K matrix $\mathbf{M}$ can be factored as $\mathbf{M} = \mathbf{BZ}$ where $\mathbf{B} \in \mathbb{R}^{d \times K}$ is orthogonal and $\mathbf{Z} \in \mathbb{R}^{K \times n}$.

Back to Matrix Completion

Let $\Omega \subseteq \{1, \ldots, d\} \times \{1, \ldots, n\}$ denote observed entries of $\mathbf{X}$ (or $\tilde{\mathbf{X}}$).

Then one version of matrix completion problem is

$$\min_{\text{rank}(\mathbf{M}) \leq K} \sum_{(u,i) \in \Omega} (x_{ui} - m_{ui})^2 \tag{17}$$

$$\equiv \min_{\mathbf{B}, \mathbf{Z}} \sum_{(u,i) \in \Omega} \left(x_{ui} - \sum_{k=1}^{K} b_{uk} z_{ki} \right)^2 .$$

Unfortunately a non-convex problem and solution not known

- but many algorithms available for finding local minima.

Algorithm #1: Find optimal $\mathbf{Z}$ in terms of $\mathbf{B}$ and then try to minimize over $\mathbf{B}$.

Question: Can you see how the PCA approximation of (16) might be used to develop an alternative algorithm? (Hint: Start with an initial guess $\hat{\mathbf{X}}_0$.)

Matrix Completion: Alternating Least Squares

Algorithm #2 (ALTERNATING LEAST SQUARES):

Select an initial $\hat{\mathbf{B}}$ and then iterate the following two steps until convergence:

(i) Optimize over $\mathbf{Z}$ for given $\hat{\mathbf{B}}$. Let $\hat{\mathbf{Z}}$ be the optimal solution.

(ii) Optimize over $\mathbf{B}$ for given $\hat{\mathbf{Z}}$. Let $\hat{\mathbf{B}}$ be the optimal solution.

- Guaranteed to converge to a local min but may do so slowly in practice.
- Should use cross-validation or validation / test set to choose r.

Figure 15.11 from Barber based on this method.

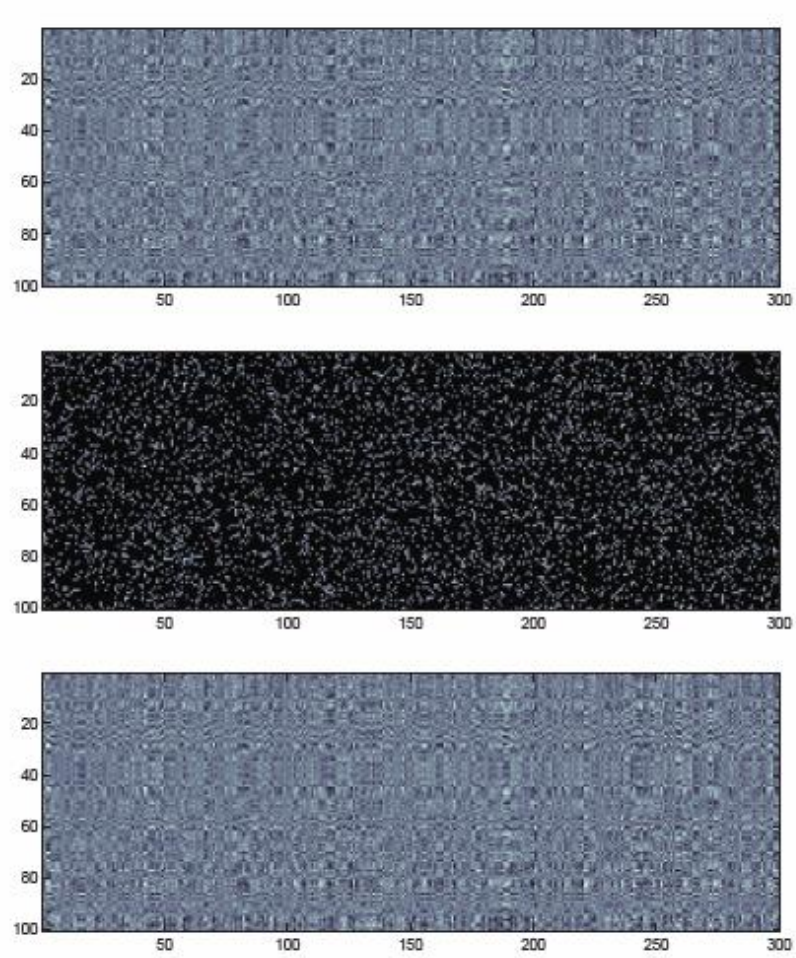


Figure 15.11 from Barber: Top: original data matrix **X**. Black is missing, white present. The data is constructed from a set of only 5 basis vectors. Middle : **X** with missing data (80% sparsity). Bottom : reconstruction found using `svdm.m`, SVD for missing data. This problem is essentially easy since, despite there being many missing elements, the data is indeed constructed from a model for which SVD is appropriate. Such techniques have application in collaborative filtering and recommender systems where one wishes to 'fill in' missing values in a matrix.

The Netflix Prize

Define the root mean-squared error (RMSE) to be

$$\text{RMSE} := \sqrt{\sum_{(u,i)} \frac{(x_{ui} - \hat{x}_{ui})^2}{C}}.$$

In 2006 Netflix launched a competition with a \$1m prize – the Netflix Prize.

- Goal of competition was to develop a recommender system that improved *Cinematch* (Netflix's proprietary system) by at least 10%.
- If goal not achieved in 3 years, \$50k consolation prize awarded to best team.
- Progress prizes of \$50k also awarded each year.
- Competition sparked huge interest — over 5k teams and 44k submissions!

Data from period 1999 to 2005 on 480k users and 17,770 movies.

- Each movie rated by more than 5,000 users on average.
- Each user rated more than 200 movies on average.

The Netflix Data-Set

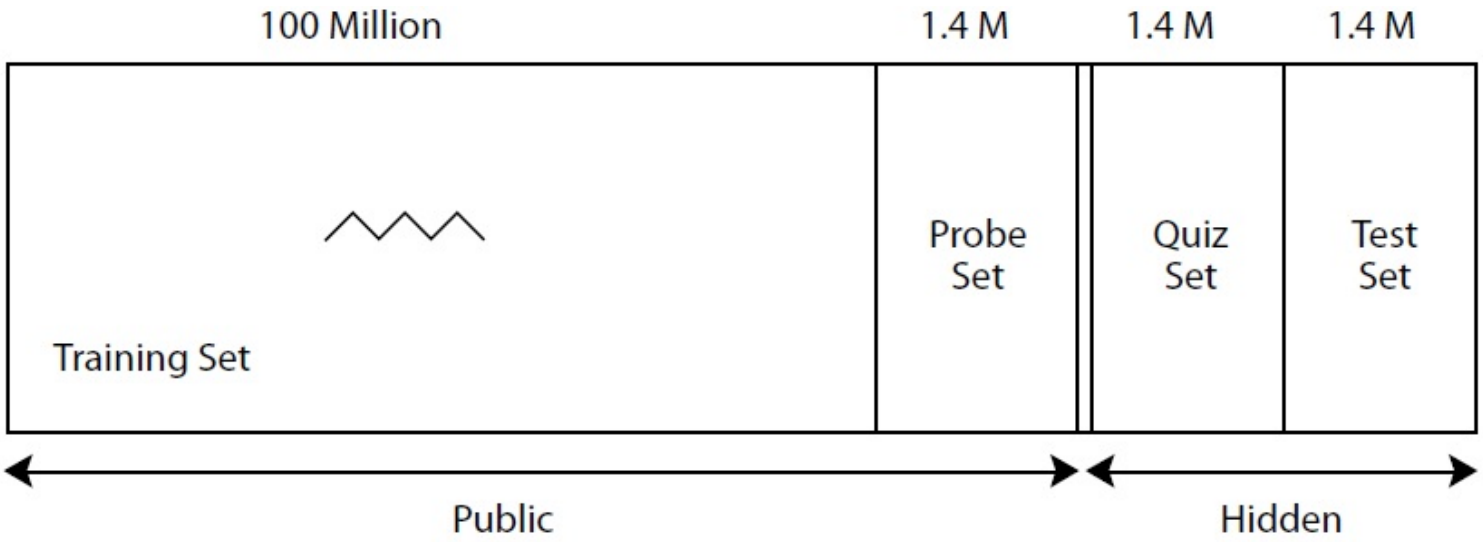


Figure 4.1 from Chiang's *Networked Life: 20 Questions and Answers* (2012): The Netflix Prize's four data sets. The training set and probe set were publicly released, whereas the quiz set and test set were hidden from the public and known only to Netflix. The probe, quiz, and test sets had similar statistical properties, but the probe set could be used by each competing team as often as they want, and the quiz set at most once a day. The final decision was based on comparison of the RMSE on the test set.

Improving *Cinematch* by 10% meant achieving an RMSE of

- 0.8563 on the quiz set
- 0.8572 on the test set.

Winner determined by performance on test set.

Matrix Factorization Results on the `Netflix` Data-set

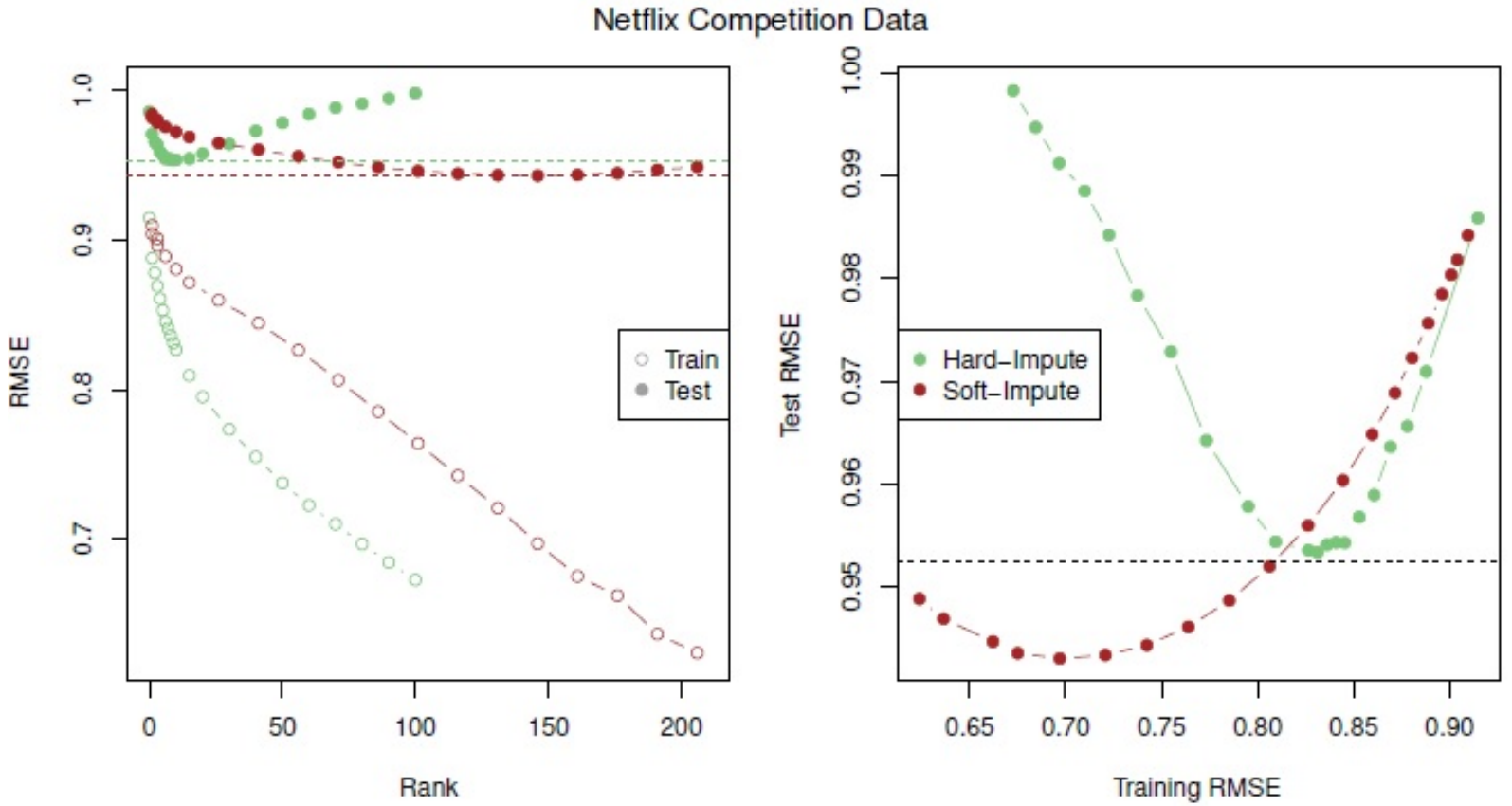


Figure 7.2 from Hastie, Tibshirani and Wainwright's *Statistical Learning with Sparsity* (2015): Left: Root-mean-squared error for the `Netflix` training and test data for the iterated-SVD (HARD-IMPUTE) and the convex spectral-regularization algorithm (SOFT-IMPUTE). Each is plotted against the rank of the solution, an imperfect calibrator for the regularized solution. Right: Test error only, plotted against training error, for the two methods. The training error captures the amount of fitting that each method performs. The dotted line represents the baseline "Cinematch" score.

And the Winner Is ...

Turns out that 10% was an excellent choice by `Netflix`:

- Relatively easy to improve on *Cinematch* by approx 8%
- But much(!) harder to improve by 10%.

Most successful submissions used combinations or ensembles of nearest neighbors and matrix factorization

- enough to get approx 8% improvement.

But many "tricks" required to get additional 2%.

Winning team's tricks included temporal effects and implicit feedback

- hard to develop without domain knowledge!

Final leader-board available at at

`http://www.netflixprize.com/leaderboard.html`

- $1m prize determined by 20 minute time differential in submission of top two entries!

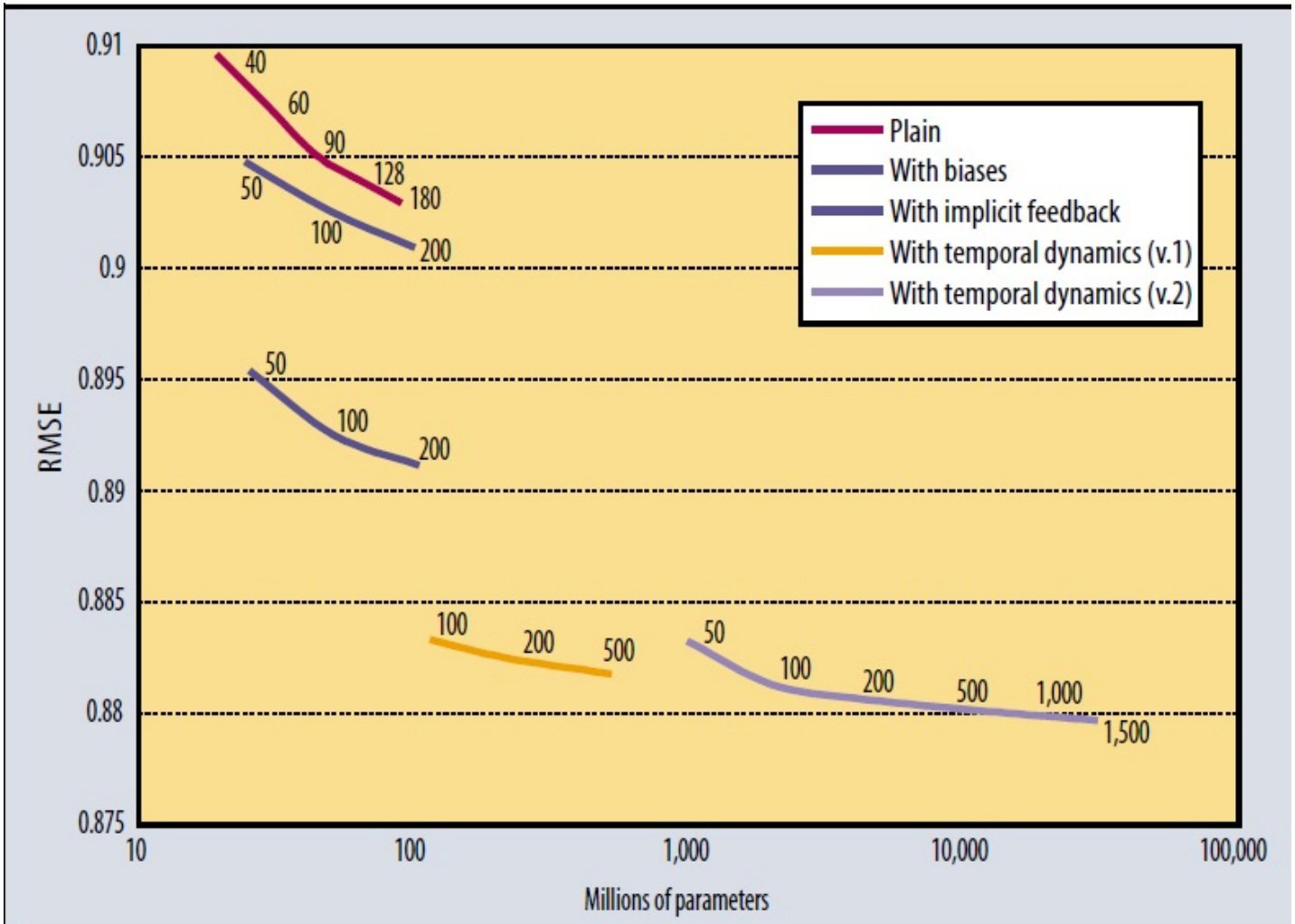


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose 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.

Source: Matrix Factorization Techniques for Recommender Systems by Koren, Bell and Volinsky, *Computer* (2009).