CPSC 340: Machine Learning and Data Mining

Sparse Matrix Factorization

Fall 2017
Admin

• Assignment 4:
  – Due Friday.

• Assignment 5:
  – Posted, due Monday of last week of classes
Last Time: PCA with Orthogonal/Sequential Basis

• When \( k = 1 \), PCA has a **scaling problem**.

• When \( k > 1 \), have **scaling**, **rotation**, and **label switching**.
  – Standard fix: use normalized orthogonal rows \( \mathbf{W}_c \) of \( \mathbf{W} \).
    \[
    \|\mathbf{W}_c\| = 1 \quad \text{and} \quad \mathbf{W}_c^T\mathbf{W}_{c'} = 0 \quad \text{for} \quad c' \neq c
    \]
  – And **fit the rows in order**:
    • First row “explains the most variance” or “reduces error the most”.
Colour Opponency in the Human Eye

• Classic model of the eye is with 4 photoreceptors:
  – Rods (more sensitive to brightness).
  – L-Cones (most sensitive to red).
  – M-Cones (most sensitive to green).
  – S-Cones (most sensitive to blue).

• Two problems with this system:
  – Not orthogonal.
    • High correlation in particular between red/green.
  – We have 4 receptors for 3 colours.

http://oneminuteastronomer.com/astro-course-day-5/
https://en.wikipedia.org/wiki/Color_vision
Colour Opponency in the Human Eye

• Bipolar and ganglion cells seem to code using “opponent colors”:
  – 3-variable orthogonal basis:

• This is similar to PCA (d = 4, k = 3).

http://oneminuteastronomer.com/astro-course-day-5/
https://en.wikipedia.org/wiki/Color_vision
http://5sensesnews.blogspot.ca/
Colour Opponency Representation

For this pixel, eye gets 4 signals

\[ = w_1 \]
\[ \sqrt{\text{First row of } W} \]
\[ \text{(First PC)} \]
\[ \text{Analogs to means in k-means.} \]

Can represent 4 original values with these 3 z1 values and matrix 'W'

\[ + w_2 \]
\[ \sqrt{\text{Second row (4x1)}} \]
\[ \text{brightness} \]

\[ + w_3 \]
\[ \sqrt{\text{Third row (4x1)}} \]
\[ \text{red/green} \]

\[ + w_4 \]
\[ \text{blue/yellow} \]
Application: Face Detection

• Consider problem of face detection:

• Classic methods use “eigenfaces” as basis:
  – PCA applied to images of faces.

Application: Face Detection
Eigenfaces

- Collect a bunch of images of faces under different conditions:

Each row of $X$ will be pixels in one image.

If you have $n$ images that are $m$ by $m$, then $X$ is $n$ by $m^2$. 
Eigenfaces

Compute mean $\mu_j$ of each column. Each row of $X$ will be pixels in one image:

Replace each $x_{ij}$ by $x_{ij} - \mu_j$
Compute top 'k' PCs on centered data: Each row of X will be pixels in one image:

\[ X = \begin{bmatrix}
  x_1 - \mu \\
  x_2 - \mu \\
  \vdots \\
  x_n - \mu 
\end{bmatrix} \]
Compute top 'k' PCs on centered data:

Note that these are "signed" images.

"gray" represents values close to 0. "dark" represents negative values. "bright" represents positive values.
Eigenfaces

Compute top 'k' PCs on centered data:

\[ x_i = \mu + \mathbf{Z}_i \]

"Eigen-face" representation:

\[ \mathbf{PC}_1 \quad (\text{first row of } W) \quad + \mathbf{PC}_2 \quad + \mathbf{PC}_3 \quad + \ldots \]
106 of the original faces:

Eigenfaces

"Eigenface" representation:

\[ x_i = \mu + z_{i1} \cdot \text{PC1} + z_{i2} \cdot \text{PC2} + z_{i3} \cdot \text{PC3} + \ldots \]
Eigenfaces

Reconstruction with $k=0$

"Eigenface" representation:

$$x_i = \mu + z_{i1} \text{ PC1 (first row of W)} + z_{i2} \text{ PC2} + z_{i3} \text{ PC3} + \ldots$$

Variance explained: 0%
Eigenfaces

Reconstruction with $k=1$

Variance explained: 34%

$z_i = w^T x_i$

"Eigenface" representation:

$\mathbf{x}_i = \mathbf{\mu} + z_{i1} \mathbf{PC1} + z_{i2} \mathbf{PC2} + z_{i3} \mathbf{PC3} + \ldots$
Reconstruction with $k=2$

Variance explained: 71%
Eigenfaces

Reconstruction with $k=3$

"Eigenface" representation:

\[ x_i = \mu + Z_{i1} + Z_{i2} + Z_{i3} + \ldots \]

Variance explained: 76%
Eigenfaces

Reconstruction with $k=5$

Variance explained: 80%
Eigenfaces

Reconstruction with $k=10$

Variance explained: 85%
Eigenfaces

Reconstruction with $k=21$

Variance explained: 90%
Reconstruction with $k = 54$

Variance explained: 95%
Eigenfaces

Original Images again:

We can replace 1024 $x_i$ values by 54 $z_i$ values.

Plus these "eigenfaces" and the mean.
VQ vs. PCA vs. NMF

• But how **should** we represent faces?
  – **Vector quantization** (k-means).
    • Replace face by the **average face in a cluster**.
    • ‘Grandmother cell’: one neuron = one face.
    • Can’t distinguish between people in the same cluster (only ‘k’ possible faces).
    • Almost certainly not true: too few neurons.
VQ vs. PCA vs. NMF

• But how should we represent faces?
  – Vector quantization (k-means).
  – PCA (orthogonal basis).
    • Global average plus linear combination of “eigenfaces”.
    • “Distributed representation”.
      – Coded by pattern of group of neurons: can represent infinite number of faces by changing \( z_i \).
  • But “eigenfaces” are not intuitive ingredients for faces.
    – PCA tends to use positive/negative cancelling bases.
VQ vs. PCA vs. NMF

• But how should we represent faces?
  – Vector quantization (k-means).
  – PCA (orthogonal basis).
  – NMF (non-negative matrix factorization):
    • Instead of requiring orthogonality/ordering, require \( W \) and \( Z \) to be non-negativity.
    • Example of “sparse coding”: 
      – The \( z_i \) are sparse so each face is coded by a small number of neurons.
      – The \( w_j \) are sparse so neurons tend to be “parts” of the object.
Representing Faces

• Why sparse coding?
  – “Parts” are intuitive, and brains seem to use sparse representation.
  – Energy efficiency if using sparse code.
  – Increase number of concepts you can memorize?
    • Some evidence in fruit fly olfactory system.

Warm-up to NMF: Non-Negative Least Squares

• Consider our usual least squares problem:

$$f(w) = \frac{1}{2} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

• But assume $y_i$ and elements of $\mathbf{x}_i$ are non-negative:
  – Could be sizes (‘height’, ‘milk’, ‘km’) or counts (‘vicodin’, ‘likes’, ‘retweets’).
• Assume we want elements of ‘$\mathbf{w}$’ to be non-negative, too:
  – No physical interpretation to negative weights.
  – If $x_{ij}$ is amount of product you produce, what does $w_j < 0$ mean?

• Non-negativity leads to sparsity...
Sparsity and Non-Negative Least Squares

• Consider 1D non-negative least squares objective:
  \[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 \text{ with } w > 0 \]

• Plotting the (constrained) objective function:

• In this case, non-negative solution is least squares solution.
Sparsity and Non-Negative Least Squares

• Consider 1D non-negative least squares objective:
  \[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 \text{ with } w > 0 \]

• Plotting the (constrained) objective function:

• In this case, non-negative solution is \( w = 0 \).
Sparsity and Non-Negativity

• Similar to L1-regularization, non-negativity leads to sparsity.
  – Also regularizes: \( w_j \) are smaller since can’t “cancel” out negative values.

• How can we minimize \( f(w) \) with non-negative constraints?
  – Naive approach: solve least squares problem, set negative \( w_j \) to 0.
    
    Compute \( w = (X^T X) \backslash (X^T y) \)
    
    Set \( w_j = \max\{0, w_j\} \)
    
    – This is correct when \( d = 1 \).
    – Can be worse than setting \( w = 0 \) when \( d \geq 2 \).
Sparsity and Non-Negativity

• Similar to L1-regularization, non-negativity leads to sparsity.
  – Also regularizes: \( w_j \) are smaller since can’t “cancel” out negative values.

• How can we minimize \( f(w) \) with non-negative constraints?
  – A correct approach is projected gradient algorithm:
    • Run a gradient descent iteration:
      \[
      w^{t+\frac{1}{2}} = w^t - \alpha^t \nabla f(w^t)
      \]
    • After each step, set negative values to 0.
      \[
      w_j^{t+1} = \max \{ 0, w_j^{t+\frac{1}{2}} \}
      \]
    • Repeat.
Sparsity and Non-Negativity

• Similar to L1-regularization, non-negativity leads to sparsity.
  – Also regularizes: \( w_j \) are smaller since can’t “cancel” out negative values.

• How can we minimize \( f(w) \) with non-negative constraints?
  – A correct approach is projected gradient algorithm:

\[
\begin{align*}
  w^{t+1} &= w^t - \alpha t \nabla f(w^t) \\
  w_j^{t+1} &= \max \{ 0, w_j^t \}
\end{align*}
\]

  – Similar properties to gradient descent:
   • Guaranteed decrease of ‘f’ if \( \alpha_t \) is small enough.
   • Reaches local minimum under weak assumptions (global minimum for convex ‘f’).
   • Generalizations allow things like L1-regularization instead of non-negativity.

(findMinL1.m)
Projected-Gradient for NMF

- Back to the non-negative matrix factorization (NMF) objective:
  \[ f(W, Z) = \sum_{i=1}^{n} \sum_{j=1}^{d} (w_j^T z_i - x_{ij})^2 \]  
  with \( w_{cj} \geq 0 \) and \( z_{ij} \geq 0 \)

- Different ways to use projected gradient:
  - Alternate between projected gradient steps on ‘W’ and on ‘Z’.
  - Or run projected gradient on both at once.
  - Or sample a random ‘i’ and ‘j’ and do stochastic projected gradient.

\[
\begin{align*}
  s_i^{t+1} &= z_i^{t} - \alpha \nabla z_i f(W, Z) \\
  w_j^{t+1} &= w_j^{t} - \alpha \nabla w_j f(W, Z)
\end{align*}
\]

- Non-convex and (unlike PCA) is sensitive to initialization.
  - Hard to find the global optimum.
  - Typically use random initialization.
Application: Sports Analytics

• NBA shot charts:

  ![NBA Shot Charts](image)

• NMF (using “KL divergence” loss with k=10 and smoothed data).
  - Negative values would not make sense here.

<table>
<thead>
<tr>
<th>Player</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>Value 6</th>
<th>Value 7</th>
<th>Value 8</th>
<th>Value 9</th>
<th>Value 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeBron James</td>
<td>0.21</td>
<td>0.16</td>
<td>0.12</td>
<td>0.09</td>
<td>0.04</td>
<td>0.07</td>
<td>0.00</td>
<td>0.07</td>
<td>0.08</td>
<td>0.17</td>
</tr>
<tr>
<td>Brook Lopez</td>
<td>0.06</td>
<td>0.27</td>
<td>0.43</td>
<td>0.09</td>
<td>0.01</td>
<td>0.03</td>
<td>0.08</td>
<td>0.03</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>Tyson Chandler</td>
<td>0.26</td>
<td>0.65</td>
<td>0.03</td>
<td>0.00</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>Marc Gasol</td>
<td>0.19</td>
<td>0.02</td>
<td>0.17</td>
<td>0.01</td>
<td>0.33</td>
<td>0.25</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td>Tony Parker</td>
<td>0.12</td>
<td>0.22</td>
<td>0.17</td>
<td>0.07</td>
<td>0.21</td>
<td>0.07</td>
<td>0.08</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Kyrie Irving</td>
<td>0.13</td>
<td>0.10</td>
<td>0.09</td>
<td>0.13</td>
<td>0.16</td>
<td>0.02</td>
<td>0.13</td>
<td>0.00</td>
<td>0.10</td>
<td>0.14</td>
</tr>
<tr>
<td>Stephen Curry</td>
<td>0.08</td>
<td>0.03</td>
<td>0.07</td>
<td>0.01</td>
<td>0.10</td>
<td>0.08</td>
<td>0.22</td>
<td>0.05</td>
<td>0.10</td>
<td>0.24</td>
</tr>
<tr>
<td>James Harden</td>
<td>0.34</td>
<td>0.00</td>
<td>0.11</td>
<td>0.00</td>
<td>0.03</td>
<td>0.02</td>
<td>0.13</td>
<td>0.00</td>
<td>0.11</td>
<td>0.26</td>
</tr>
<tr>
<td>Steve Novak</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.27</td>
<td>0.35</td>
<td>0.34</td>
</tr>
</tbody>
</table>

Application: Cancer “Signatures”

- What are common sets of mutations in different cancers?
  - May lead to new treatment options.
Regularized Matrix Factorization

• For many PCA applications, ordering orthogonal PCs makes sense.
  – Latent factors are independent of each other.
  – We definitely want this for visualization.

• In other cases, ordering orthogonal PCs doesn’t make sense.
  – We might not expect a natural “ordering”.

Regularized Matrix Factorization

• More recently people have considered L2-regularized PCA:

\[ f(W, Z) = \frac{1}{2} \|ZW - X\|_F^2 + \frac{\lambda_1}{2} \|W\|_F^2 + \frac{\lambda_2}{2} \|Z\|_F^2 \]

• Replaces normalization/orthogonality/sequential-fitting.
  – But requires regularization parameters \( \lambda_1 \) and \( \lambda_2 \).

• Need to regularize \( W \) and \( Z \) because of scaling problem:
  – If you only regularize ‘\( W \)’ it doesn’t do anything:
    • I could take unregularized solution, replace \( W \) by \( \alpha W \) for a tiny \( \alpha \) to shrink \( \|W\|_F \) as much as I want, then multiply \( Z \) by \( (1/\alpha) \) to get same solution.
  – Similarly, if you only regularize ‘\( Z \)’ it doesn’t do anything.
Sparse Matrix Factorization

• Instead of non-negativity, we could use L1-regularization:

\[ f(W, Z) = \frac{1}{2} \| ZW - X \|_F^2 + \frac{\lambda_1}{2} \sum_{i=1}^n \| z_i \|_1 + \frac{\lambda_2}{2} \sum_{j=1}^d \| w_j \|_1 \]

  – Called sparse coding (L1 on ‘Z’) or sparse dictionary learning (L1 on ‘W’).

• **Disadvantage of using L1-regularization** over non-negativity:
  – Sparsity controlled by \( \lambda_1 \) and \( \lambda_2 \) so you need to set these.

• **Advantage of using L1-regularization**:
  – Negative coefficients usually make sense.
  – Sparsity controlled by \( \lambda_1 \) and \( \lambda_2 \), so you can control amount of sparsity.
Sparse Matrix Factorization

• Instead of non-negativity, we could use L1-regularization:
  \[ f(Z) = \frac{1}{2} ||Zw - X||^2_F + \frac{\lambda_1}{2} \sum_{i=1}^{n} ||z_i||_1 + \frac{\lambda_2}{2} \sum_{j=1}^{d} ||w_j||_1 \]

  – Called sparse coding (L1 on ‘Z’) or sparse dictionary learning (L1 on ‘W’).

• Many variations exist:
  – Mixing L2-regularization and L1-regularization or making one a constraint.
  – K-SVD constrains each \( z_i \) to have at most ‘k’ non-zeroes:
    • K-means is special case where \( k = 1 \).
    • PCA is special case where \( k = d \).
Matrix Factorization with L1-Regularization

(a) PCA

PCA without orthogonality

(b) Dictionary Learning

Sparsity due to non-negativity

(c) NMF

(d) SPCA, $\tau = 30\%$

Sparsity due to $L_1$-regularization

blue: negative
red: positive
Recent Work: Structured Sparsity

• “Structured sparsity” considers dependencies in sparsity patterns.
  – Can enforce that parts are convex regions.

Beyond Squared Error

• Our (unregularized) objective for latent-factor models (LFM):
  \[ f(w_j, Z) = \sum_{i=1}^{n} \sum_{j=1}^{d} (w_i^T z_i - x_{ij})^2 \]

• As before, there are squared error alternatives.

• We can get a LFM for binary +1/-1 \( x_{ij} \) using the logistic loss:
  \[ f(w_j, Z) = \sum_{i=1}^{n} \sum_{j=1}^{d} \log(1 + \exp(-x_{ij} w_i^T z_i)) \]
Robust PCA

• Robust PCA methods use the absolute error:

\[ f(W, Z) = \sum_{i=1}^{n} \sum_{j=1}^{d} |w_j^T z_i - x_{ij}| \]

• Will be robust to outliers in the matrix ‘X’.
• Encourages “residuals” \( r_{ij} \) to be exactly zero.
  – Non-zero \( r_{ij} \) are where the “outliers” are.

http://statweb.stanford.edu/~candes/papers/RobustPCA.pdf
Robust PCA

• Miss Korea contestants and robust PCA:
Topic Models

• For modeling data as combinations of non-negative parts, NMF has largely replaced by “topic models”.
  – A “fully-Bayesian” model where sparsity arises naturally.
  – Most popular example is called “latent Dirichlet allocation” (CPSC 540).

Summary

- **Biological motivation** for orthogonal and/or sparse latent factors.
- **Non-negative matrix factorization** leads to sparse LFM.
- **Non-negativity** constraints lead to sparse solution.
  - Projected gradient adds constraints to gradient descent.
  - Non-orthogonal LFM makes sense in many applications.
- **L1-regularization** leads to other sparse LFM.
- **Robust PCA** allows identifying certain types of outliers.

- Next time: the NetFlix challenge.
Motivation for Topic Models

• Want a model of the “factors” making up documents.
  – Instead of latent-factor models, they’re called topic models.
  – The canonical topic model is latent Dirichlet allocation (LDA).

Suppose you have the following set of sentences:
• I like to eat broccoli and bananas.
• I ate a banana and spinach smoothie for breakfast.
• Chinchillas and kittens are cute.
• My sister adopted a kitten yesterday.
• Look at this cute hamster munching on a piece of broccoli.

What is latent Dirichlet allocation? It’s a way of automatically discovering topics that these sentences contain. For example, given these sentences and asked for 2 topics, LDA might produce something like

• Sentences 1 and 2: 100% Topic A
• Sentences 3 and 4: 100% Topic B
• Sentence 5: 60% Topic A, 40% Topic B
• Topic A: 30% broccoli, 15% bananas, 10% breakfast, 10% munching, ... (at which point, you could interpret topic A to be about food)
• Topic B: 20% chinchillas, 20% kittens, 20% cute, 15% hamster, ... (at which point, you could interpret topic B to be about cute animals)

“Topics” could be useful for things like searching for relevant documents.
Term Frequency – Inverse Document Frequency

• In information retrieval, classic word importance measure is TF-IDF.

• First part is the term frequency $tf(t,d)$ of term ‘t’ for document ‘d’.
  – Number of times “word” ‘t’ occurs in document ‘d’, divided by total words.
  – E.g., 7% of words in document ‘d’ are “the” and 2% of the words are “Lebron”.

• Second part is document frequency $df(t,D)$.
  – Compute number of documents that have ‘t’ at least once.
  – E.g., 100% of documents contain “the” and 0.01% have “LeBron”.

• TF-IDF is $tf(t,d) \times \log(1/df(t,D))$. 
Term Frequency – Inverse Document Frequency

- The **TF-IDF** statistic is \( tf(t,d) \ast \log(1/df(t,D)) \).
  - It’s high if word ‘t’ happens often in document ‘d’, but isn’t common.
  - E.g., seeing “LeBron” a lot it tells you something about “topic” of article.
  - E.g., seeing “the” a lot tells you nothing.

- There are *many* variations on this statistic.
  - E.g., avoiding dividing by zero and all types of “frequencies”.

- Summarizing ‘n’ documents into a matrix X:
  - Each row corresponds to a document.
  - Each column gives the TF-IDF value of a particular word in the document.
Latent Semantic Indexing

- TF-IDF features are very redundant.
  - Consider TF-IDFs of “LeBron”, “Durant”, “Harden”, and “Kobe”.
  - High values of these typically just indicate topic of “basketball”.

- We can probably compress this information quite a bit.

- Latent Semantic Indexing/Analysis:
  - Run latent-factor model (like PCA or NMF) on TF-IDF matrix $X$.
  - Treat the principal components as the “topics”.
  - Latent Dirichlet allocation is a variant that avoids weird df(t,D) heuristic.