CPSC 340: Machine Learning and Data Mining

Multi-Dimensional Scaling

Fall 2022
Last Time: Collaborative Filtering with Latent Factors

• We discussed recommender systems using collaborative filtering:
  – Methods that only looks at ratings, not features of movies/users.


• We discussed collaborative filtering with matrix factorization:
  
  \[ Y \approx ZW \quad y_{ij} \approx \langle w_j, z_i \rangle \]

  – Fit to minimize regularized squared error on available ratings (with biases).
  
  • The learned \( w_j \) and \( z_i \) can be used to predict unknown \( y_{ij} \) values.
  
  – Can be viewed as “PCA on the available entries”.

• We discussed collaborative filtering with matrix factorization:
  – Methods that only looks at ratings, not features of movies/users.

Beyond Accuracy in Recommender Systems

• Winning system of Netflix Challenge was never adopted.
• Other issues important in recommender systems:
  – Diversity: how different are the recommendations?
    • If you like ‘Battle of Five Armies Extended Edition’, recommend Battle of Five Armies?
    • Even if you really really like Star Wars, you might want non-Star-Wars suggestions.
  – Persistence: how long should recommendations last?
    • If you keep not clicking on ‘Hunger Games’, should it remain a recommendation?
  – Trust: tell user why you made a recommendation.
    • Quora gives explanations for recommendations.
  – Social recommendation: what did your friends watch?
  – Freshness: people tend to get more excited about new/surprising things.
    • Collaborative filtering does not predict well for new users/movies.
      – New movies don’t yet have ratings, and new users haven’t rated anything.
Content-Based vs. Collaborative Filtering

• Consider **content-based filtering**, our usual supervised learning (Part 3):

\[ \hat{y}_{ij} = w^T x_{ij} \]

– Here \( x_{ij} \) is a fixed vector of features for the movie/user.
  • Usual supervised learning setup: ‘\( y \)’ would contain all the \( y_{ij} \), \( X \) would have \( x_{ij} \) as rows.
  • Can predict on new users/movies, but can’t learn about each user/movie.
    • If two users have the same features, then they get the exact same recommendations.

• Our latent-factor approach to **collaborative filtering** (Part 4):

\[ \hat{y}_{ij} = \langle w^j, z_i \rangle \]

"hidden" features of movie \( \rightarrow \) "hidden" features of user

– Learns vector of features \( z_i \) for each user ‘\( i \).’
– But can’t predict on new users (with no ratings).
Hybrid Content/Collaborative: SVDfeature

- SVDfeature combines content-based/collaborative filtering:

\[
\hat{y}_{ij} = w^T x_{ij} + \langle w_j, z_i \rangle
\]

- Learns weights ‘\(w\)’ on fixed features \(x_{ij}\).
  - Allows predictions for generic users/movies (including new ones).
- And learns movie-specific weights \(w_j\) on learned user-specific features \(z_i\).
  - Allows more-accurate predictions for users/movies with lots of data.
- Typically you also have a global bias \(\beta\), user-specific bias \(\beta_i\), and movie-specific \(\beta_j\).
  - And train with SGD (see bonus slides).
- Won “KDD Cup” competition in 2011 and 2012.
Social Regularization

• Many recommenders are now connected to social networks.
  – “Login using your Facebook account”.

• Often, people like similar movies to their friends.

• Recent recommender systems use social regularization.
  – Add a “regularizer” encouraging friends’ weights to be similar:

\[
\lambda \sum \frac{1}{2} \|(z_i - z_j)\|^2_{ij} \text{ for } (i,j) \in \text{“friends”}
\]

  – If we get a new user, recommendations are based on friend’s preferences.
Next Topic: Multi-Dimensional Scaling
Visualization High-Dimensional Data

• PCA for visualizing high-dimensional data:
  – Use PCA ‘W’ matrix to linearly transform data to get the $z_i$ values.
  – And then we plot the $z_i$ values as locations in a scatterplot.

http://scienceblogs.com/gnxp/2008/08/14/the-genetic-map-of-europe/
Visualization High-Dimensional Data

• PCA for **visualizing high-dimensional data**:
  – Use PCA ‘W’ matrix to **linearly transform data** to get the $z_i$ values.
  – And then we plot the $z_i$ values as locations in a scatterplot.

• An common alternative is **multi-dimensional scaling (MDS)**:
  – Directly optimize the pixel locations of the $z_i$ values.
    • “Gradient descent on the points in a scatterplot”.
  – Needs a “cost” function saying how “good” the $z_i$ locations are.
    • Traditional **MDS cost function**:
      \[
      f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|z_i - z_j\| - \|x_i - x_j\|)^2
      \]
MDS Method ("Sammon Mapping") Video
Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the $z_i$ values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|z_i - z_j\| - \|x_i - x_j\|)^2$$
Multi-Dimensional Scaling

- **Multi-dimensional scaling (MDS):**
  - Directly optimize the final locations of the $z_i$ values.
    \[
    f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \|z_i - z_j\| - \|x_i - x_j\| \right)^2
    \]
  - **Non-parametric** dimensionality reduction and visualization:
    - No ‘$W$’: just trying to make $z_i$ preserve high-dimensional distances between $x_i$. 

---

Diagram showing PCA rotation and projection.
Multi-Dimensional Scaling

• Multi-dimensional scaling (MDS):
  – Directly optimize the final locations of the $z_i$ values.
    \[
    f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|z_i - z_j\| - \|x_i - x_j\|)^2 \]
  – Non-parametric dimensionality reduction and visualization:
    • No ‘W’: just trying to make $z_i$ preserve high-dimensional distances between $x_i$. 

\[\text{PCA}\] only moves points closer.
\[\text{MDS}\] can preserve distance.
Multi-Dimensional Scaling

- **Multi-dimensional scaling (MDS):**
  - Directly optimize the final locations of the $z_i$ values.
    \[ f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \| z_i - z_j \| - \| x_i - x_j \| \right)^2 \]
  - Non-parametric dimensionality reduction and visualization:
    - No ‘$W$’: just trying to make $z_i$ preserve high-dimensional distances between $x_i$. 

![PCA rotation and projection diagram](image-url)
Multi-Dimensional Scaling

• Multi-dimensional scaling (MDS):
  – Directly optimize the final locations of the $z_i$ values.
    
    $$J(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|z_i - z_j\| - \|x_i - x_j\|)^2$$
  
  – Non-parametric dimensionality reduction and visualization:
    • No ‘$W$’: just trying to make $z_i$ preserve high-dimensional distances between $x_i$. 
Multi-Dimensional Scaling

• **Multi-dimensional scaling (MDS):**
  – Directly optimize the final locations of the $z_i$ values.
    \[
    f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \|z_i - z_j\| - \|x_i - x_j\| \right)^2
    \]

• **Cannot use SVD** to compute solution:
  – Instead, do gradient descent on the $z_i$ values.
  – You “learn” a scatterplot that tries to visualize high-dimensional data.
  – Not convex and sensitive to initialization.
    • And solution is not unique due to various factors like translation and rotation.
Different MDS Cost Functions

• Unfortunately, **MDS often does not work well in practice**.
• Problem with traditional MDS methods: **focus on large distances**.
  – MDS tends to “crowd/squash” all the data points together like PCA.
• But we could consider **different distances/similarities**:
  – Where the functions are **not necessarily the same**:
    • $d_1$ is the high-dimensional distance we want to match.
    • $d_2$ is the low-dimensional distance we can control.
    • $d_3$ controls how we compare high-/low-dimensional distances.
• Early example was **Sammon’s Mapping** (details in bonus).
  – We next discuss t-SNE, a more recent method that tends to work better.
MDS with Squared Distances vs. Sammon’s Map

- MDS based on Euclidean distances (left) vs. Sammon’s Map (right):

http://www.mdpi.com/1422-0067/15/7/12364/htm
Next Topic: t-SNE
Data on Manifolds

- Consider data that lives on a low-dimensional “manifold”.
  - Where Euclidean distances make sense “locally”.
    - But Euclidean distances may not make sense “globally”.
  - Wikipedia example: Surface of the Earth is “locally” flat.
    - Euclidean distance accurately measures distance “along the surface” locally.
    - For far points Euclidean distance is a poor measure of distance “along the surface”.

http://www.biomedcentral.com/content/pdf/1471-2105-13-S7-S3.pdf
Data on Manifolds

• Consider data that lives on a **low-dimensional “manifold”**.
  
  – Where Euclidean distances make sense “locally”.
  
  • But Euclidean distances may not make sense “globally”.

• Example is the ‘Swiss roll’:

![Swiss roll diagram]

We want an MDS method that visualizes manifolds

This visualization "unrolls" the Swiss roll

http://www.biomedcentral.com/content/pdf/1471-2105-13-57-53.pdf
Example: Manifolds in Image Space

• Slowly-varying image transformations exist on a manifold:
  • “Neighbouring” images are close in Euclidean distance.
    – But distances between very-different images are not reliable.

Learning Manifolds

• With usual distances, PCA/MDS do not discover non-linear manifolds.

Original data

PCA
Learning Manifolds

• With usual distances, PCA/MDS do not discover non-linear manifolds.

• We could use change of basis or kernels: but still need to pick basis.
A classic way to visualize manifolds is **ISOMAP**.

- Uses approximation of geodesic distance within MDS (see bonus slides).

Sammon’s Map vs. ISOMAP vs. t-SNE (MNIST)

- A modern way to visualize manifolds and clusters is t-SNE.

Sammon’s Map vs. ISOMAP vs. t-SNE (MNIST)

Sammon Map

ISOMAP

t-SNE

Remember this is unsupervised algorithms do not know the labels.

Sammon’s Map vs. ISOMAP vs. t-SNE (MNIST)

Remember this is **unsupervised** algorithms do not
know the labels.

Sammon’s Map vs. ISOMAP vs. t-SNE (MNIST)

Remember this is unsupervised algorithms do not know the labels.

Sammon’s Map vs. ISOMAP vs. t-SNE (MNIST)

Remember this is unsupervised, algorithms do not know the labels.

t-Distributed Stochastic Neighbour Embedding

• One key idea in t-SNE:
  – Focus on distance to “neighbours” (allow large variance in other distances)
t-Distributed Stochastic Neighbour Embedding

- **t-SNE** is a special case of MDS (specific \(d_1\), \(d_2\), and \(d_3\) choices):
  - \(d_1\): for each \(x_i\), compute probability that each \(x_j\) is a ‘neighbour’.
    - Computation is similar to k-means++, but most weight to close points (Gaussian).
    - Does not require explicit geodesic distance approximation.
  
  - \(d_2\): for each \(z_i\), compute probability that each \(z_j\) is a ‘neighbour’.
    - Similar to above, but uses student’s t (grows really slowly with distance).
    - Avoids ‘crowding’, because you have a huge range that large distances can fill.

  - \(d_3\): Compares \(x_i\) and \(z_i\) using an entropy-like measure:
    - How much ‘randomness’ is in probabilities of \(x_i\) if you know the \(z_i\) (and vice versa)?

- Interactive demo: [https://distill.pub/2016/misread-tsne](https://distill.pub/2016/misread-tsne)
t-SNE on Wikipedia Articles

t-SNE on Product Features

t-SNE on Leukemia Heterogeneity

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4076922/
Next Topic: Word2Vec
Latent-Factor Representation of Words

• For natural language, we often represent words by an index.
  – E.g., “cat” is word 124056 among a “bag of words”.

• But this may be inefficient:
  – Should “cat” and “kitten” features be related in some way?

• We want a latent-factor representation of individual words:
  – Closeness in latent space should indicate similarity.
  – Distances could represent meaning?

• Recent alternative to PCA is word2vec...
Using Context

• Consider these phrases:
  – “the cat purred”
  – “the kitten purred”
  – “black cat ran”
  – “black kitten ran”

• Words that occur in the same context likely have similar meanings.

• Word2vec uses this insight to design an MDS distance function.
Word2Vec (Continuous Bag of Words)

- A common word2vec approaches (called continuous bag of words):
  - Each word ‘i’ is represented by a vector of real numbers \( z_i \).
  - Training data: sentence fragments with “hidden” middle word:
    - “We introduce basic principles and techniques in”
    - “the fields of data mining and machine”
    - “tools behind the emerging field of data”
    - “techniques are now running behind the scenes”
    - “discover patterns and make predictions in various”
    - “the core data mining and machine learning”
    - “with motivating applications from a variety of”
  - Train so that \( z_i \) of “hidden” words is are similar to \( z_i \) of surrounding words.
Word2Vec (Continuous Bag of Words)

- Continuous bag of words model probability of middle word ‘i’ as:

  \[ \prod_{j \in \text{surrounding words}} \frac{\exp(z_i^T z_j)}{\sum_{i=1}^{\text{#words}} \exp(z_i^T z_j)} \]

- We use gradient descent on negative logarithm of these probabilities:
  - Makes \( z_i^T z_j \) big for words appearing in same context (making \( z_i \) close to \( z_j \)).
  - Makes \( z_i^T z_j \) small for words not appearing together (makes \( z_i \) and \( z_j \) far).

- Once trained, you use these \( z_i \) as features for language tasks.
  - Tends to work much better than bag of words.
  - Allows you to get useful features of words from unlabeled text data.
Word2Vec (Skip-Gram)

• A common word2vec approaches (skip gram):
  – Each word ‘i’ is represented by a vector of real numbers $z_i$.
  – Training data: sentence fragments with “hidden” surrounding word:
    • “We introduce basic principles and techniques in”
    • “the fields of data mining and machine”
    • “tools behind the emerging field of data”
    • “techniques are now running behind the scenes”
    • “discover patterns and make predictions in various”
    • “the core data mining and machine learning”
    • “with motivating applications from a variety of”
  – Train so that $z_i$ of “hidden” words is similar to $z_i$ of surrounding words.
    • Uses same probability as continuous bag of words.
      – But denominator sums over all possible surrounding words (often just sample terms for speed).
Word2Vec Example

• MDS visualization of a set of related words:

• Distances between vectors might represent semantics.

http://sebastianruder.com/secret-word2vec
Word2Vec

• Subtracting word vectors to find related vectors.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>

Table 8 shows words that follow various relationships. We follow the approach described above: the relationship is defined by subtracting two word vectors, and the result is added to another word. Thus for example, Paris - France + Italy = Rome. As it can be seen, accuracy is quite good, although

• Word vectors for 157 languages here.

Summary

• **Multi-dimensional scaling** is a non-parametric latent-factor model.
• **Different MDS distances/losses/weights** usually gives better results.
• **Manifold**: space where local Euclidean distance is accurate.
  – Structured data like images often form manifolds in space.
• **t-SNE** is an MDS method focusing on matching small distances.
• **Word2vec**:
  – Latent-factor (continuous) representation of words.
  – Based on predicting word from its context (or context from word).

• Next time: deep learning.
Stochastic Gradient for SVD feature

• Common approach to fitting SVD feature is **stochastic gradient**.
• Previously you saw stochastic gradient for supervised learning:
  - Choose a random example ‘i’
  - Update parameters ‘w’ using gradient of example ‘i’
• Stochastic gradient for SVD feature (formulas as bonus):
  - Choose a random user ‘i’ and a random product ‘j’
  - Update $\beta_i$, $\beta_j$, $w_j$, $z_i$, and $w_j$ based on their gradient
  for this user-product
Objective: \( \frac{1}{2} \sum_{(i,j) \in R} (\hat{y}_{ij} - y_{ij})^2 \) with \( \hat{y}_{ij} = \beta + \beta_i + \beta_j + w^T x_{ij} + (w^T z_i \cdot r_{ij}) \)

Update based on random \((i,j)\):

\[
\begin{align*}
\beta &= \beta - \alpha r_{ij} \\
\beta_i &= \beta_i - \alpha r_{ij} \\
\beta_j &= \beta_j - \alpha r_{ij}
\end{align*}
\]

Updates are the same, but \(\beta\) is always updated while \(\beta_i\) and \(\beta_j\) are only updated for the specific user and product.

\[
\begin{align*}
w &= w - \alpha r_{ij} x_{ij} \quad \text{Updated every time.} \\
z_i &= z_i - \alpha r_{ij} w_j \\
w^j &= w^j - \alpha r_{ij} z_i \quad \text{Updated for specific user and product.}
\end{align*}
\]

(Adding regularization adds an extra term)
Tensor Factorization

• Tensors are higher-order generalizations of matrices:

Scalar \( \alpha = \sum_{1}^{d} \)  
Vector \( \alpha = [ \cdots ]_{d \times 1} \)  
Matrix \( A = \begin{bmatrix} \cdots \end{bmatrix}_{d \times d} \)  
Tensor \( A = \begin{bmatrix} \cdots \end{bmatrix}_{d \times d \times d} \)

• Generalization of matrix factorization is tensor factorization:

\[ y_{ijm} \sim \sum_{c=1}^{k} w_{jc} z_{ic} v_{mc} \]

• Useful if there are other relevant variables:
  • Instead of ratings based on \{user,movie\}, ratings based \{user,movie,group\}.
  • Useful if you have groups of users, or if ratings change over time.
Field-Aware Matrix Factorization

- **Field-aware factorization machines (FFMs):**
  - Matrix factorization with multiple $z_i$ or $w_c$ for each example or part.
  - You choose which $z_i$ or $w_c$ to use based on the value of feature.

- **Example from “click through rate” prediction:**
  - E.g., predict whether “male” clicks on “nike” advertising on “espn” page.
  - A previous matrix factorization method for the 3 factors used:
    - FFMs could use:
      - $w_{espn} w_{nike} + w_{espn} w_{male} + w_{nike} w_{male}$
      - $w^A_{espn} w^A_{nike} + w^p_{espn} w^p_{male} + w^G_{espn} w^G_{male}$
  - This approach has won some Kaggle competitions ([link](#)), and has shown to work well in production systems too ([link](#)).
Warm-Starting

• We’ve used data \{X,y\} to fit a model.
• We now have new training data and want to fit new and old data.
• Do we need to re-fit from scratch?
• This is the \textit{warm starting} problem.
  – It’s easier to warm start some models than others.
Easy Case: K-Nearest Neighbours and Counting

• K-nearest neighbours:
  – KNN just stores the training data, so just store the new data.

• Counting-based models:
  – Models that base predictions on frequencies of events.
  – E.g., naïve Bayes.
  – Just update the counts:
    \[
    p("vicelin" | \text{"spam"}) = \frac{\text{count of \{"vicelin\,spam\} in new and old data}}{\text{count of \"spam\" in new and old data}}
    \]
  – Decision trees with fixed rules: just update counts at the leaves.
Medium Case: L2-Regularized Least Squares

- **L2-regularized least squares** is obtained from linear algebra:
  \[ W = (X^T X + \lambda I)^{-1} X^T y \]
  - Cost is \( O(nd^2 + d^3) \) for ‘n’ training examples and ‘d’ features.

- Given one new point, we need to compute:
  - \( X^T y \) with one row added, which costs \( O(d) \).
  - Old \( X^T X \) plus \( x_i x_i^T \), which costs \( O(d^2) \).
  - Solution of linear system, which costs \( O(d^3) \).
  - So cost of adding ‘t’ new data point is \( O(td^3) \).

- With “matrix factorization updates”, can reduce this to \( O(td^2) \).
  - Cheaper than computing from scratch, particularly for large \( d \).
Medium Case: Logistic Regression

• We fit logistic regression by gradient descent on a convex function.

• With new data, convex function $f(w)$ changes to new function $g(w)$.

$$f(w) = \sum_{i=1}^{n} f_i(w) \quad g(w) = \sum_{i=1}^{n+1} f_i(w)$$

• If we don’t have much more data, ‘f’ and ‘g’ will be “close”.
  – Start gradient descent on ‘g’ with minimizer of ‘f’.
  – You can show that it requires fewer iterations.
Hard Cases: Non-Convex/Greedy Models

• For decision trees:
  – “Warm start”: continue splitting nodes that haven’t already been split.
  – “Cold start”: re-fit everything.

• Unlike previous cases, this won’t in general give same result as re-fitting:
  – New data points might lead to different splits higher up in the tree.

• Intermediate: usually do warm start but occasionally do a cold start.

• Similar heuristics/conclusions for other non-convex/greedy models:
  – K-means clustering.
  – Matrix factorization (though you can continue PCA algorithms).
Different MDS Cost Functions

- MDS default objective function with general distances/similarities:
  \[
  \hat{f}(Z) = \sum_{i=1}^{\hat{n}} \sum_{j=i+1}^{\hat{n}} d_3(d_2(z_i, z_j) - d_1(x_i, x_j))
  \]

- A possibility is “classic” MDS with \(d_1(x_i, x_j) = x_i^T x_j\) and \(d_2(z_i, z_j) = z_i^T z_j\).
  - We obtain PCA in this special case (centered \(x_i\), \(d_3\) as the squared L2-norm).
  - Not a great choice because it’s a linear model.
Different MDS Cost Functions

• **MDS** default objective function with **general distances/similarities**:

  \[
  f(\mathbf{Z}) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_3(d_2(z_i, z_j) - d_1(x_i, x_j))
  \]

• Another possibility: \(d_1(x_i, x_j) = ||x_i - x_j||_1\) and \(d_2(z_i, z_j) = ||z_i - z_j||\).

  – The \(z_i\) approximate the high-dimensional \(L_1\)-norm distances.

http://www.mdpi.com/1422-0067/15/7/12364/htm
Sammon’s Mapping

• Challenge for most MDS models: they focus on large distances.
  – Leads to “crowding” effect like with PCA.

• Early attempt to address this is Sammon’s mapping:
  – Weighted MDS so large/small distances are more comparable.
    \[ f(Z) = \sum_{i=1}^{n} \sum_{j \neq i} \left( \frac{d(Z_i, Z_j) - d(x_i, x_j)}{d(x_i, x_j)} \right)^2 \]
  – Denominator reduces focus on large distances.
Sammon’s Mapping

• Challenge for most MDS models: they focus on large distances.
  – Leads to “crowding” effect like with PCA.

• Early attempt to address this is Sammon’s mapping:
  – Weighted MDS so large/small distances are more comparable.

http://www.mdpi.com/1422-0067/15/7/12364/htm
Geodesic Distance on Manifolds

• Consider data that lives on a low-dimensional “manifold”.
  – With usual distances, PCA/MDS will not discover non-linear manifolds.
• We need geodesic distance: the distance *through* the manifold.

http://www.biomedcentral.com/content/pdf/1471-2105-13-S7-S3.pdf
ISOMAP

- ISOMAP is a latent-factor model for visualizing data on manifolds:
  - Find "neighbours" of each point
  - Represent points and neighbours as a weighted graph
  - Approximate geodesic distance between points by shortest path through graph
  - ISOMAP $z_i$ values in 10 or 20
  - Run MDS with these approximate geodesic distances
  - $D = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 3 & 1 & 0 \\ 3 & 2 & 0 & 1 \end{bmatrix}$
ISOMAP

• **ISOMAP** can “unwrap” the roll:
  – Shortest paths are approximations to geodesic distances.

• Sensitive to having the right graph:
  – Points off of manifold and gaps in manifold cause problems.
Constructing Neighbour Graphs

• Sometimes you can define the graph/distance without features:
  – Facebook friend graph.
  – Connect YouTube videos if one video tends to follow another.

• But we can also convert from features \( x_i \) to a “neighbour” graph:
  – Approach 1 ("epsilon graph"): connect \( x_i \) to all \( x_j \) within some threshold \( \varepsilon \).
    • Like we did with density-based clustering.

  – Approach 2 ("KNN graph"): connect \( x_i \) to \( x_j \) if:
    • \( x_j \) is a KNN of \( x_i \) OR \( x_i \) is a KNN of \( x_j \).

  – Approach 2 ("mutual KNN graph"): connect \( x_i \) to \( x_j \) if:
    • \( x_j \) is a KNN of \( x_i \) AND \( x_i \) is a KNN of \( x_j \).
Converting from Features to Graph

- Add edge if \( \|x_i - x_j\| \leq 0.3 \)
- Add edge if \( i \) is 5-NN of \( j \) or \( j \) is 5-NN of \( i \)
- Add edge if \( i \) and \( j \) are kNNs of each other.
ISOMAP

• **ISOMAP** is latent-factor model for visualizing data on manifolds:
  
  1. **Find the neighbours** of each point.
     - Usually “k-nearest neighbours graph”, or “epsilon graph”.
  
  2. **Compute edge weights:**
     - Usually distance between neighbours.
  
  3. **Compute weighted shortest path** between all points.
     - Dijkstra or other shortest path algorithm.
  
  4. **Run MDS** using these distances.

Does t-SNE always outperform PCA?

- Consider 3D data living on a 2D hyper-plane:
  - PCA can perfectly capture the low-dimensional structure.
  - T-SNE can capture the local structure, but can “twist” the plane.
    - It doesn’t try to get long distances correct.
Graph Drawing

• A closely-related topic to MDS is graph drawing:
  – Given a graph, how should we display it?
  – Lots of interesting methods: https://en.wikipedia.org/wiki/Graph_drawing
Bonus Slide: Multivariate Chain Rule

- Recall the univariate chain rule:
  \[ \frac{d}{dw} [ f(g(w)) ] = f'(g(w)) g'(w) \]

- The multivariate chain rule:
  \[ \nabla [ f(g(w)) ] = f'(g(w)) \nabla g(w) \]

- Example:
  \[ \nabla \left[ \frac{1}{2} (w^T x_i - y_i)^2 \right] \]
  \[ = \nabla [ f(g(w)) ] \]
  \[ \text{with } g(w) = w^T x_i - y_i \]
  \[ \nabla g(w) = x_i \]
  \[ = \nabla [ f(g(w)) ] = r_i x_i \]
  \[ = (w^T x_i - y_i) x_i \]
Bonus Slide: Multivariate Chain Rule for MDS

• General MDS formulation:

\[
\arg \min_{Z \in \mathbb{R}^{n \times k}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} g \left( d_1(x_i, x_j), d_2(z_i, z_j) \right)
\]

• Using multivariate chain rule we have:

\[
\nabla_{z_i} g \left( d_1(x_i, x_j), d_2(z_i, z_j) \right) = g' \left( d_1(x_i, x_j), d_2(z_i, z_j) \right) \nabla_{z_i} d_2(z_i, z_j)
\]

• Example: If \( d_1(x_i, x_j) = \|x_i - x_j\| \) and \( d_2(z_i, z_j) = \|z_i - z_j\| \) and

\[
g(d_1, d_2) = \frac{1}{2} (d_1 - d_2)^2
\]

\[
\nabla_{z_i} g \left( d_1(x_i, x_j), d_2(z_i, z_j) \right) = \left( d_1(x_i, x_j) - d_2(z_i, z_j) \right) \left[ -\frac{2(z_i - z_j)}{2 \|z_i - z_j\|^2} \right] \nabla_{z_i} d_2(z_i, z_j)
\]

Assuming \( z_i \neq z_j \)

(move distances closer) (how distance changes in \( z \)-space)
Multiple Word Prototypes

• What about homonyms and polysemy?
  – The word vectors would need to account for all meanings.

• More recent approaches:
  – Try to cluster the different contexts where words appear.
  – Use different vectors for different contexts.
Multiple Word Prototypes

http://www.socher.org/index.php/Main/ImprovingWordRepresentationsViaGlobalContextAndMultipleWordPrototypes