Machine Learning & Data Mining
CS/CNS/EE 155

Lecture 14:
Embeddings
Past Two Lectures

• Dimensionality Reduction
• Clustering

• Latent Factor Models
  – Learn low-dimensional representation of data
This Lecture

• Embeddings
  – Generalization of Latent-Factor Models

• Warm-up: Locally-Linear Embeddings

• Probabilistic Sequence Embeddings
  – Playlist embeddings
  – Word embeddings
Embedding

• Learn a representation U
  – Each column $u$ corresponds to data point

• Semantics encoded via $d(u,u')$
  – Distance between points
    \[ d(u,u') = \|u - u'\|^2 \]
  – Similarity between points
    \[ d(u,u') = u^T u' \]

Generalizes Latent-Factor Models
Locally Linear Embedding

- Given: \( S = \{ x_i \}_{i=1}^{N} \)

- Learn \( U \) such that local linearity is preserved
  - Lower dimensional than \( x \)
  - “Manifold Learning”

Any neighborhood looks like a linear plane

https://www.cs.nyu.edu/~roweis/lle/
Approach

• Define relationship of each x to its neighbors

• Find a lower dimensional u that preserves relationship
Locally Linear Embedding

• Create $B(i)$
  
  – $B$ nearest neighbors of $x_i$
  
  – **Assumption:** $B(i)$ is approximately linear
  
  – $x_i$ can be written as a convex combination of $x_j$ in $B(i)$

\[
S = \left\{ x_i \right\}_{i=1}^{N}
\]

\[
x_i \approx \sum_{j \in B(i)} W_{ij} x_j
\]

\[
\sum_{j \in B(i)} W_{ij} = 1
\]

https://www.cs.nyu.edu/~roweis/lle/
Locally Linear Embedding

Given Neighbors \( B(i) \), solve local linear approximation \( W \):

\[
\arg\min_W \sum_i \left\| x_i - \sum_{j \in B(i)} W_{ij} x_j \right\|^2
\]

\[
\left\| x_i - \sum_{j \in B(i)} W_{ij} x_j \right\|^2 = \left\| \sum_{j \in B(i)} W_{ij} (x_i - x_j) \right\|^2
\]

\[
= \left( \sum_{j \in B(i)} W_{ij} (x_i - x_j) \right)^T \left( \sum_{j \in B(i)} W_{ij} (x_i - x_j) \right)
\]

\[
= \sum_{j \in B(i)} \sum_{k \in B(i)} W_{ij} W_{ik} C_{jk}^i
\]

\[
= W_{i,*}^T C_i W_{i,*}
\]

https://www.cs.nyu.edu/~roweis/lle/
Locally Linear Embedding

Given Neighbors $B(i)$, solve local linear approximation $W$:

$$\arg\min_W \sum_i \left\| x_i - \sum_{j \in B(i)} W_{ij} x_j \right\|^2 = \arg\min_W \sum_i W_{i,*}^T C_i^* W_{i,*} \quad \sum_{j \in B(i)} W_{ij} = 1$$

$$C_{jk}^i = (x_i - x_j)^T (x_i - x_k)$$

- Every $x_i$ is approximated as a convex combination of neighbors
  - How to solve?
Lagrange Multipliers

\[ \text{argmin} \ L(w) \equiv w^T Cw \]

s.t. \( |w| = 1 \)

\[ \nabla_{w_j} |w| \begin{cases} 
-1 & \text{if } w_j < 0 \\
+1 & \text{if } w_j > 0 \\
[-1, +1] & \text{if } w_j = 0
\end{cases} \]

\[ \exists \lambda \geq 0 : (\partial_w L(y, w) \in \lambda \nabla_w |w|) \land (|w| = 1) \]

Solutions tend to be at corners!

http://en.wikipedia.org/wiki/Lagrange_multiplier
Solving Locally Linear Approximation

Lagrangian:

\[ L(W, \lambda) = \sum_i \left( W^T_i C_i W_i^* - \lambda_i \left( \mathbf{i}^T W_i^* - 1 \right) \right) \]

\[ \sum_j W_{ij} = \mathbf{i}^T W_i^* \]

\[ \partial_{W_{ij}} L(W, \lambda) = 2 C_i W_j^* - \lambda_i \mathbf{i} \]

\[ W_{ij} = \frac{\lambda_i}{2} \left( C^i \right)^{-1} \mathbf{1} \propto \left( C^i \right)^{-1} \mathbf{1} \]

\[ W_{ij} \propto \sum_{k \in B(i)} \left( C^i \right)^{-1}_{jk} \]

\[ W_{ij} = \frac{\sum_{k \in B(i)} \left( C^i \right)^{-1}_{jk}}{\sum_{l \in B(i)} \sum_{m \in B(i)} \left( C^i \right)^{-1}_{lm}} \]
Locally Linear Approximation

- Invariant to:
  - Rotation: $Ax_i \approx \sum_{j \in B(i)} AW_{ij} x_j$
  - Scaling: $5x_i \approx \sum_{j \in B(i)} 5W_{ij} x_j$
  - Translation: $x_i + x' \approx \sum_{j \in B(i)} W_{ij} (x_j + x')$

$$x_i \approx \sum_{j \in B(i)} W_{ij} x_j$$

$$\sum_{j \in B(i)} W_{ij} = 1$$
Story So Far: Locally Linear Embeddings

Given Neighbors B(i), solve local linear approximation W:

\[
\min_{W} \sum_{i} \left\| x_i - \sum_{j \in B(i)} W_{ij} x_j \right\|^2 = \min_{W} \sum_{i} W_{i,*}^T C^i W_{i,*} \quad \sum_{j \in B(i)} W_{ij} = 1
\]

Solution via Lagrange Multipliers:

\[
W_{ij} = \frac{\sum_{k \in B(i)} \left( C^i \right)_{jk}^{-1}}{\sum_{l \in B(i)} \sum_{m \in B(i)} \left( C^i \right)_{lm}^{-1}}
\]

\[
C^i_{jk} = (x_i - x_j)^T (x_i - x_k)
\]

• Locally Linear Approximation

https://www.cs.nyu.edu/~roweis/lle/
Recall: Locally Linear Embedding

- Given: \[ S = \{ x_i \}_{i=1}^{N} \]

- Learn U such that local linearity is preserved
  - Lower dimensional than x
  - “Manifold Learning”

https://www.cs.nyu.edu/~roweis/lle/
Dimensionality Reduction
(Learning the Embedding)

Given local approximation $W$, learn lower dimensional representation:

$$\arg\min_U \sum_i \left\| u_i - \sum_{j \in B(i)} W_{ij} u_j \right\|^2$$

- Find low dimensional $U$
  - Preserves approximate local linearity

https://www.cs.nyu.edu/~roweis/lle/
Given local approximation $W$, learn lower dimensional representation:

$$\arg\min_U \sum_i \left\| u_i - \sum_{j \in B(i)} W_{ij} u_j \right\|^2$$

$$UU^T = I_K$$

$$\sum_i u_i = \hat{0}$$

• Rewrite as:

$$\arg\min_U \sum_{ij} M_{ij} (u_i^T u_j) \equiv \text{trace} \left( UMU^T \right)$$

$$M_{ij} = 1_{[i=j]} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj}$$

$$M = (I_N - W)^T (I_N - W)$$

Symmetric positive semidefinite

https://www.cs.nyu.edu/~roweis/lle/
Given local approximation $W$, learn lower dimensional representation:

$$\arg\min_u \sum_{ij} M_{ij}(u_i^T u_j) \equiv \text{trace}(UMU^T)$$

- Suppose $K=1$

$$\arg\min_u \sum_{ij} M_{ij}(u_i^T u_j) \equiv \text{trace}(uMu^T)$$

$$= \arg\max_u \text{trace}(uM^+u^T)$$

- By min-max theorem
  - $u = \text{principal eigenvector of } M^+$

$$UU^T = I_K$$

$$\sum_i u_i = 0$$

$$uu^T = 1$$

http://en.wikipedia.org/wiki/Min-max_theorem
Recap: Principal Component Analysis

\[ M = V \Lambda V^T \]

- Each column of \( V \) is an Eigenvector
- Each \( \lambda \) is an Eigenvalue (\( \lambda_1 \geq \lambda_2 \geq \ldots \))

\[ M^+ = V \Lambda^+ V^T \]

\[ M M^+ = V \Lambda \Lambda^+ V^T = V_{1:2} V_{1:2}^T = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \]
Given local approximation $W$, learn lower dimensional representation:

$$\arg\min_U \sum_{ij} M_{ij}(u_i^T u_j) \equiv \text{trace}(UMU^T)$$

$$UU^T = I_K$$

• $K=1$:
  - $u = \text{principal eigenvector of } M^+$
  - $u = \text{smallest non-trivial eigenvector of } M$
    - Corresponds to smallest non-zero eigenvalue

• General $K$
  - $U = \text{top } K \text{ principal eigenvectors of } M^+$
  - $U = \text{bottom } K \text{ non-trivial eigenvectors of } M$
    - Corresponds to bottom $K \text{ non-zero eigenvalues}$

https://www.cs.nyu.edu/~roweis/lle/
http://en.wikipedia.org/wiki/Min-max_theorem
Recap: Locally Linear Embedding

• Generate nearest neighbors of each $x_i$, $B(i)$

• Compute Local Linear Approximation:

$$\arg\min_w \sum_i \left\| x_i - \sum_{j \in B(i)} W_{ij} x_j \right\|^2$$

$$\sum_{j \in B(i)} W_{ij} = 1$$

• Compute low dimensional embedding

$$\arg\min_u \sum_i \left\| u_i - \sum_{j \in B(i)} W_{ij} u_j \right\|^2$$

$$UU^T = I_K$$

$$\sum_i u_i = \vec{0}$$
Results for Different Neighborhoods

(K=2)

https://www.cs.nyu.edu/~roweis/lle/gallery.html
Probabilistic Sequence Embeddings
Example 1: **Playlist Embedding**

- **Users generate song playlists**
  - Treat as training data

- **Can we learn a probabilistic model of playlists?**
Example 2: Word Embedding

- People write natural text all the time
  - Treat as training data

- Can we learn a probabilistic model of word sequences?
Probabilistic Sequence Modeling

• Training set:
  \[ S = \{ s_1, \ldots, s_{|S|} \} \]
  \[ D = \{ p_i \}_{i=1}^{N} \]
  \[ p_i = \left< p_i^1, \ldots, p_i^{N_i} \right> \]

  Songs, Words  \hspace{1cm}  Playlists, Documents  \hspace{1cm}  Sequence Definition

• **Goal:** Learn a Markov model of sequences:
  \[ P(p_i^j \mid p_i^{j-1}) \]

• What is the form of \( P \)?
First Try: Probability Tables

| P(s|s') | s_1  | s_2  | s_3  | s_4  | s_5  | s_6  | s_7  | s_start |
|--------|------|------|------|------|------|------|------|---------|
| s_1    | 0.01 | 0.03 | 0.01 | 0.11 | 0.04 | 0.04 | 0.01 | 0.05    |
| s_2    | 0.03 | 0.01 | 0.04 | 0.03 | 0.02 | 0.01 | 0.02 | 0.02    |
| s_3    | 0.01 | 0.01 | 0.01 | 0.07 | 0.02 | 0.02 | 0.05 | 0.09    |
| s_4    | 0.02 | 0.11 | 0.07 | 0.01 | 0.07 | 0.04 | 0.01 | 0.01    |
| s_5    | 0.04 | 0.01 | 0.02 | 0.17 | 0.01 | 0.01 | 0.10 | 0.02    |
| s_6    | 0.01 | 0.02 | 0.03 | 0.01 | 0.01 | 0.01 | 0.01 | 0.08    |
| s_7    | 0.07 | 0.02 | 0.01 | 0.01 | 0.03 | 0.09 | 0.03 | 0.01    |
First Try: Probability Tables

| P(s|s') | s_1  | s_2  | s_3  | s_4  | s_5  | s_6  | s_7  | s_{start} |
|--------|------|------|------|------|------|------|------|-----------|
| s_1    | 0.01 | 0.03 | 0.01 | 0.11 | 0.04 | 0.04 | 0.01 | 0.05      |
| s_2    | 0.03 | 0.01 | 0.04 | 0.03 | 0.02 | 0.01 | 0.02 | 0.02      |
| s_3    | 0.01 | 0.01 | 0.01 | 0.07 | 0.02 | 0.02 | 0.05 | 0.09      |
| s_4    | 0.02 | 0.11 | 0.07 | 0.01 | 0.07 | 0.04 | 0.01 | 0.01      |
| s_5    |      |      |      |      |      |      |      |           |
| s_6    |      |      |      |      |      |      |      |           |
| s_7    |      |      |      |      |      |      |      |           |

#Parameters = O(|S|^2) !!!
(worse for higher-order sequence models)
Second Try: Hidden Markov Models

\[ P(p_i, z) = P(End | z^{N_i}) \prod_{j=1}^{N_i} P(z^j | z^{j-1}) \prod_{j=1}^{N_j} P(p_i^j | z^j) \]

- \( P(z^j | z^{j-1}) \) • #Parameters = \( O(K^2) \)

- \( P(p_i^j | z^j) \) • #Parameters = \( O(|S|K) \)

- Total = \( O(K^2) + O(|S|K) \)
Problem with Hidden Markov Models

\[ P(p_i, z) = P(\text{End} \mid z^{N_i}) \prod_{j=1}^{N_i} P(z^j \mid z^{j-1}) \prod_{j=1}^{N_j} P(p_i^j \mid z^j) \]

- Need to reliably estimate \( P(s \mid z) \)

\[ S = \{s_1, \ldots, s_{|S|}\} \quad D = \{p_i\}_{i=1}^N \quad p_i = \left\langle p_i^1, \ldots, p_i^{N_i} \right\rangle \]

- Hard to do!
Outline for Sequence Modeling

• Playlist Embedding
  – Distance-based embedding

• Word Embedding (word2vec)
  – Inner-product embedding
    – https://code.google.com/archive/p/word2vec/

• Compare the two approaches
Markov Embedding (Distance)

\[ P(s | s') \propto \exp \left\{ -\| u_s - v_{s'} \|^2 \right\} \]

\[ P(s | s') = \frac{\exp \left\{ -\| u_s - v_{s'} \|^2 \right\}}{\sum_{s''} \exp \left\{ -\| u_{s''} - v_{s'} \|^2 \right\}} \]

- “Log-Radial” function
  - (my own terminology)

\[ u_s: \text{entry point of song } s \]
\[ v_s: \text{exit point of song } s \]

Log-Radial Functions

Each ring defines an equivalence class of transition probabilities

\[ P(s \mid s') = \frac{\exp\left\{-\left\|u_s - v_{s'}\right\|^2\right\}}{\sum_{s''} \exp\left\{-\left\|u_{s''} - v_{s'}\right\|^2\right\}} \]

2K parameters per song
2 |S|K parameters total
Learning Problem

\[ S = \{s_1, \ldots, s_{|S|}\} \quad D = \{p_i\}_{i=1}^N \quad p_i = \langle p_i^1, \ldots, p_i^{N_i} \rangle \]

- **Songs**
- **Playlists**
- **Playlist Definition** (each \( p_j^i \) corresponds to a song)

**Learning Goal:**

\[
\text{argmax}_{U,V} \prod_i P(p_i) = \prod_i \prod_j P(p_i^j | p_i^{j-1})
\]

\[
P(s | s') = \frac{\exp\left\{ -\|u_s - v_{s'}\|^2 \right\}}{\sum_{s''} \exp\left\{ -\|u_{s''} - v_{s'}\|^2 \right\}} = \frac{\exp\left\{ -\|u_s - v_{s'}\|^2 \right\}}{Z(s')}
\]

Minimize Neg Log Likelihood

\[
\arg\max_{U,V} \prod_i \prod_j P(p^j_i \mid p^{j-1}_i) = \arg\min_{U,V} \sum_i \sum_j -\log P(p^j_i \mid p^{j-1}_i)
\]

• Solve using gradient descent
  – Random initialization

• Normalization constant hard to compute:
  – Approximation heuristics

  • See paper

  \[
P(s \mid s') = \frac{\exp\left\{-\|u_s - v_{s'}\|^2\right\}}{Z(s')}
\]

Story so Far: Playlist Embedding

• Training set of playlists
  – Sequences of songs

• Want to build probability tables \( P(s \mid s') \)
  – But a lot of missing values, hard to generalize directly
  – Assume low-dimensional embedding of songs

\[
P(s \mid s') = \frac{\exp\left\{-\|u_s - v_{s'}\|^2\right\}}{\sum_{s''} \exp\left\{-\|u_{s''} - v_{s'}\|^2\right\}} = \frac{\exp\left\{-\|u_s - v_{s'}\|^2\right\}}{Z(s')}
\]
Simpler Version

- Dual point model:
  \[ P(s \mid s') = \frac{\exp\left\{ -\|u_s - v_{s'}\|^2 \right\}}{Z(s')} \]

- Single point model:
  \[ P(s \mid s') = \frac{\exp\left\{ -\|u_s - u_{s'}\|^2 \right\}}{Z(s')} \]
  - Transitions are symmetric
  - (almost)
  - Exact same form of training problem
Visualization in 2D

**Simpler version:**
**Single Point Model**

\[
P(s | s') = \frac{\exp \left\{ -\| u_s - u_{s'} \|^2 \right\}}{Z(s')}
\]

Single point model is easier to visualize

Sampling New Playlists

• Given partial playlist:

\[ p = \langle p^1, \ldots, p^j \rangle \]

• Generate next song for playlist \( p^{j+1} \)

  – Sample according to:

\[
P(s \mid p^j) = \frac{\exp\left\{ -\|u_s - v_{p^j}\|^2 \right\}}{Z(p^j)}
\]

Dual Point Model

\[
P(s \mid p^j) = \frac{\exp\left\{ -\|u_s - u_{p^j}\|^2 \right\}}{Z(p^j)}
\]

Single Point Model

Demo

http://jimi.ithaca.edu/~dturnbull/research/lme/lmeDemo.html
What About New Songs?

• Suppose we’ve trained $U$:

$$P(s | s') = \frac{\exp\left\{-\|u_s - u_{s'}\|^2\right\}}{Z(s')}$$

• What if we add a new song $s'$?
  – No playlists created by users yet...
  – Only options: $u_{s'} = 0$ or $u_{s'} = \text{random}$
    • Both are terrible!
    • “Cold-start” problem
Song & Tag Embedding

• Songs are usually added with tags
  – E.g., indie rock, country
  – Treat as features or attributes of songs

• How to leverage tags to generate a reasonable embedding of new songs?
  – Learn an embedding of tags as well!

\[ S = \{s_1, \ldots, s_{|S|}\} \quad \text{Songs} \]
\[ D = \{p_i\}_{i=1}^{N} \quad \text{Playlists} \]
\[ p_i = \langle p_i^1, \ldots, p_i^{N_i} \rangle \quad \text{Playlist Definition} \]
\[ T = \{T_1, \ldots, T_{|S|}\} \quad \text{Tags for Each Song} \]

\textbf{Learning Objective:}
\[
\arg\max_{U,A} P(D \mid U)P(U \mid A,T)
\]

\textbf{Same term as before:}
\[
P(D \mid U) = \prod_i P(p_i \mid U) = \prod_i \prod_j P(p_i^j \mid p_i^{j-1}, U)
\]

\textbf{Song embedding \(\approx\) average of tag embeddings:}
\[
P(U \mid A,T) = \prod_s P(u_s \mid A,T_s) \propto \prod_s \exp \left\{ -\lambda \left\| u_s - \frac{1}{|T_s|} \sum_{t \in T_s} A_t \right\|^2 \right\}
\]

\textbf{Solve using gradient descent:}

The simplest NLP model is the n-gram models from natural language processing (NLP).

To evaluate our method and the embeddings it produces, we use a common NLP benchmark dataset: the Million Song Dataset (MSD). For each song, we treat every tag as a word and a short 1-hot vector indicating whether the tag is present in the song. The representations found by these models are then visualized in figure 2.

...
Revisited: What About New Songs?

• No user has $s'$ added to playlist
  – So no evidence from playlist training data:

$$s' \text{ does not appear in } D = \left\{ p_i \right\}_{i=1}^{N}$$

• Assume new song has been tagged $T_s'$
  – The $u_{s'} = \text{average of } A_t \text{ for tags } t \text{ in } T_{s'}$
  – Implication from objective:

$$\arg\max_{U,A} P(D|U)P(U|A,T)$$
Switching Gears: Word Embeddings

• Given a large corpus
  – Wikipedia
  – Google News

• Learn a word embedding to model sequences of words (e.g., sentences)

https://code.google.com/archive/p/word2vec/
Switching Gears: Inner Product Embeddings

• Previous: capture semantics via distance

\[ P(s \mid s') = \frac{\exp\left\{-\|u_s - v_{s'}\|^2\right\}}{\sum_{s''} \exp\left\{-\|u_{s''} - v_{s'}\|^2\right\}} \]

• Can also capture semantics via inner product

\[ P(s \mid s') = \frac{\exp\left\{u_s^T v_{s'}\right\}}{\sum_{s''} \exp\left\{u_{s''}^T v_{s'}\right\}} \]

Basically a latent-factor model!
Log-Linear Embeddings

\[ P(s \mid s') = \frac{\exp\{u_s^T v_{s'}\}}{\sum_{s''} \exp\{u_{s''}^T v_{s'}\}} \]

2K parameters per song
2|S|K parameters total

Each projection level onto the green line defines an equivalence class
Learning Problem (Version 1)

\[ S = \{s_1, \ldots, s_{|S|}\} \quad \text{Words} \]
\[ D = \{p_i\}_{i=1}^N \quad \text{Sentences} \]
\[ p_i = \left\langle p_i^1, \ldots, p_i^{N_i} \right\rangle \quad \text{Sentence Definition} \]
\[ \text{(Each } p^j \text{ is a word)} \]

Learning Goal:

\[
\arg \max_{U,V} \prod_i P(p_i) = \prod_i \prod_j P(p_i^j \mid p_i^{j-1})
\]

\[
P(s \mid s') = \frac{\exp \left\{ u_s^T v_{s'} \right\}}{\sum_{s''} \exp \left\{ u_{s''}^T v_{s'} \right\} \ Z(s')}
\]
Skip-Gram Model (word2vec)

- Predict probability of any neighboring word

\[
\text{argmax}_{U,V} \prod_i \prod_j \prod_{k \in [-C,C] \setminus 0} P(p_{i+k}^j \mid p_i^j)
\]

\[
P(s \mid s') = \frac{\exp\{u_s^T v_{s'}\}}{\sum_{s''} \exp\{u_{s''}^T v_{s'}\}} = \frac{\exp\{u_s^T v_{s'}\}}{Z(s')}
\]

https://code.google.com/archive/p/word2vec/
Skip-Gram Model (word2vec)

- Predict probability of any neighboring word

\[
\text{argmax}_{U,V} \prod_i \prod_j \prod_{k \in [-C,C] \setminus 0} P(p_{i+j+k} | p_i^j)
\]

What are benefits of Skip-Gram model?

https://code.google.com/archive/p/word2vec/
Intuition of Skip-Gram Model

• “The dog jumped over the fence.”
• “My dog ate my homework.”
• “I walked my dog up to the fence.”

Example sentences

\[ \arg \max_{U,V} \prod_i \prod_j \prod_{k \in [-C, C] \setminus 0} P(p_{i+j+k} \mid p_i^j) \]

• Distribution of neighboring words more peaked
• Distribution of further words more diffuse
• Capture everything in a single model
Dimensionality Reduction

• What dimensionality should we choose U, V?
  – E.g., what should K be?

\[ P(s \mid s') = \frac{\exp\left\{ u_s^T v_{s'} \right\}}{\sum_{s''} \exp\left\{ u_{s''}^T v_{s'} \right\}} \]

• K = |S|^2 implies we can memorize every word pair interaction
• Smaller K assumes words lie in lower-dimensional space
  – Easier to generalize across words
• Larger K can overfit
Example 1

- $V_{\text{Czech}} + V_{\text{currency}} \approx V_{\text{koruna}}$

<table>
<thead>
<tr>
<th>Czech + currency</th>
<th>Vietnam + capital</th>
<th>German + airlines</th>
<th>Russian + river</th>
<th>French + actress</th>
</tr>
</thead>
<tbody>
<tr>
<td>koruna</td>
<td>Hanoi</td>
<td>airline Lufthansa</td>
<td>Moscow</td>
<td>Juliette Binoche</td>
</tr>
<tr>
<td>Check crown</td>
<td>Ho Chi Minh City</td>
<td>carrier Lufthansa</td>
<td>Volga River</td>
<td>Vanessa Paradis</td>
</tr>
<tr>
<td>Polish zolty</td>
<td>Viet Nam</td>
<td>flag carrier Lufthansa</td>
<td>upriver</td>
<td>Charlotte Gainsbourg</td>
</tr>
<tr>
<td>CTK</td>
<td>Vietnamese</td>
<td>Lufthansa</td>
<td>Russia</td>
<td>Cecile De</td>
</tr>
</tbody>
</table>

• \( \text{E.g., } \mathbf{v}_{\text{France}} - \mathbf{v}_{\text{Paris}} + \mathbf{v}_{\text{Italy}} \approx \mathbf{v}_{\text{Rome}} \)

<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>

Example 3

- 2D PCA projection of countries and cities:
Aside: Embeddings as Features

• Use the learned u (or v) as features

• E.g., linear models for classification:

\[ h(x) = \text{sign} \left( w^T \phi(x) \right) \]

Can be word identities or word2vec representation!
Training word2vec

• Train via gradient descent

\[
\arg\min_{U,V} \sum_{i} \sum_{j} \sum_{k \in [-C,C] \setminus 0} -\log P(p_{i}^{j+k} | p_{i}^{j})
\]

Sequences

Tokens in each Sequence

Skip Length

\[
P(s | s') = \frac{\exp\{u_{s}^{T}v_{s'}\}}{\sum_{s''} \exp\{u_{s'}^{T}v_{s''}\}} = \frac{\exp\{u_{s}^{T}v_{s'}\}}{Z(s')}
\]

Denominator expensive!

https://code.google.com/archive/p/word2vec/
Hierarchical Approach
(Probabilistic Decision Tree)

- Decision tree of paths
- Leaf node = word
- Choose each branch independently

Hierarchical Approach
(Probabilistic Decision Tree)

\[ P(s_1 | s') = P(B | A, s') P(s_1 | B, s') \]

\[ P(s_2 | s') = P(B | A, s') P(s_2 | B, s') \]

\[ P(s_3 | s') = P(C | A, s') P(s_3 | C, s') \]

\[ P(s_4 | s') = P(C | A, s') P(s_4 | C, s') \]

Hierarchical Approach
(Probabilistic Decision Tree)

\[
P(B \mid A, s) = \frac{1}{1 + \exp\left\{-u_{BC}^T v_s\right\}} = \frac{1}{1 + \exp\left\{u_{CB}^T v_s\right\}}
\]

\[
P(C \mid A, s) = \frac{1}{1 + \exp\left\{-u_{CB}^T v_s\right\}} = \frac{1}{1 + \exp\left\{u_{BC}^T v_s\right\}}
\]

\[u_{BC} = -u_{CB}\]
Hierarchical Approach
(Probabilistic Decision Tree)

\[ P(s_1 \mid B, s) = \frac{1}{1 + \exp\{-u_{12}^T v_s\}} = \frac{1}{1 + \exp\{u_{21}^T v_s\}} \]

\[ P(s_2 \mid B, s) = \frac{1}{1 + \exp\{-u_{21}^T v_s\}} = \frac{1}{1 + \exp\{u_{12}^T v_s\}} \]

\[ u_{12} = -u_{21} \]
Hierarchical Approach
(Probabilistic Decision Tree)

• Compact formula:

\[ P(s | s') = \prod_{m} P(n_{m,s} | n_{m-1,s}, s) \]

Levels in tree

Internal node at level m on path to leaf node s
Training Hierarchical Approach

- Train via gradient descent (same as before!)

\[
\arg\min_{U,V} \sum_{i} \sum_{j} \sum_{k \in [-C,C] \setminus 0} -\log P(p_{i}^{j+k} | p_{i}^{j})
\]

Complexity = \(O(\log_2(|S|))!\)

\[
P(s | s') = \prod_{m} P(n_{m,s} | n_{m-1,s}, s)
\]

https://code.google.com/archive/p/word2vec/
Summary: Hierarchical Approach

• Each word has a corresponding:
  – One \( v_s \)
  – \( \log_2(|S|) \) u’s!

• Target factors u’s are shared across words
  – Total number of U is still \( O(|S|) \)

• Previous use cases unchanged
  – They all used \( v_s \)
  – Vector subtraction, use as features for CRF, etc.
Recap: Embeddings

• **Given**: Training Data
  – Care about some property of training data
    • Markov Chain
    • Skip-Gram

• **Goal**: learn low dim representation
  – “Embedding”
  – Geometry of embedding captures property of interest
    • Either by distance or by inner-product
Visualization Semantics

Inner-Product Embeddings
Similarity measured via dot product
Rotational semantics
Can interpret axes
Can only visualize 2 axes at a time

Distance-Based Embedding
Similarity measured via distance
Clustering/locality semantics
Cannot interpret axes
Can visualize many clusters simultaneously
Next Week

• Recent Applications Lectures

• Latent Factor Models

• Deep Generative Models