Non-Standard Datenbanken und Data Mining

From Clustering to Embedding

Prof. Dr. Ralf Möller
Universität zu Lübeck
Institut für Informationssysteme
Übersicht

- Semistrukturierte Datenbanken (JSON, XML) und Volltextsuche
- Information Retrieval
- Mehrdimensionale Indexstrukturen
- Cluster-Bildung
- **Einbettungstechniken**
- First-n-, Top-k-, und Skyline-Anfragen
- Probabilistische Datenbanken, Anfragebeantwortung, Top-k-Anfragen und Open-World-Annahme
- Probabilistische Modellierung, Bayes-Netze, Anfragebeantwortungsalgorithmen, Lernverfahren,
- Temporale Datenbanken und das relationale Modell,
- Probabilistische Temporale Datenbanken
- SQL: neue Entwicklungen (z.B. JSON-Strukturen und Arrays), Zeitreihen (z.B. TimeScaleDB)
- Stromdatenbanken, Prinzipien der Fenster-orientierten inkrementellen Verarbeitung
- Approximationstechniken für Stromdatenverarbeitung, Stream-Mining
- Probabilistische raum-zeitliche Datenbanken und Stromdatenverarbeitungssysteme: Anfragen und Indexstrukturen, Raum-zeitliches Data Mining, Probabilistische Skylines
- Von NoSQL- zu NewSQL-Datenbanken, CAP-Theorem, Blockchain-Datenbanken
Word-Word Associations in Document Retrieval

Recap bag-of-words approaches
- LSI: Documents as vectors, dimension reduction

Words are not independent of each other
- Word similarity measures
- Extend query with similar words automatically
- Extend query with most frequent followers/predecessors
- Insert words in anticipated gaps in a string query

Need to represent some aspects of word semantics
Approaches for Representing Word Semantics

**Beyond bags of words**

**Distributional Semantics (Count)**
- Used since the 90’s
- Sparse word-context PMI/PPMI matrix
- Decomposed with SVD

**Word Embeddings (Predict)**
- Inspired by deep learning
- **word2vec** *(Mikolov et al., 2013)*
- **GloVe** *(Pennington et al., 2014)*

Underlying Theory: **The Distributional Hypothesis** *(Harris, ’54; Firth, ’57)*
“Similar words occur in similar contexts”

https://www.tensorflow.org/tutorials/word2vec
https://nlp.stanford.edu/projects/glove
References

• Harris 54

• Firth 57

• Micholov et al. 13

• Pennington et al. 14
Point(wise) Mutual Information: PMI

- Measure of association used in information theory and statistics
  
  \[ \text{pmi}(x; y) \equiv \log \frac{p(x, y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)} \]

- Positive PMI: \( \text{PPMI}(x, y) = \max( \text{pmi}(x, y), 0 ) \)

- Quantifies the discrepancy between the probability of their coincidence given their joint distribution and their individual distributions, assuming independence

- Finding collocations and associations between words

- Countings of occurrences and co-occurrences of words in a text corpus can be used to approximate the probabilities \( p(x) \) or \( p(y) \) and \( p(x, y) \) respectively

[Wikipedia]
PMI – Example

Counts of pairs of words getting the most and the least PMI scores in the first 50 millions of words in Wikipedia (dump of October 2015)

Filtering by 1,000 or more co-occurrences.

The frequency of each count can be obtained by dividing its value by 50,000,952. (Note: natural log is used to calculate the PMI values in this example, instead of log base 2)
Applications of PMI Data

- Extend query with most frequent followers/predecessors
- Insert words in anticipated gaps in a string query

## PMI – Co-occurrence Matrix

### Add-2 Smoothed Count\((w, context)\)

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>pineapple</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>digital</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>information</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

### PPMI\((w, context)\)

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>-</td>
<td>-</td>
<td>2.25</td>
<td>-</td>
<td>2.25</td>
</tr>
<tr>
<td>pineapple</td>
<td>-</td>
<td>-</td>
<td>2.25</td>
<td>-</td>
<td>2.25</td>
</tr>
<tr>
<td>digital</td>
<td>1.66</td>
<td>0.00</td>
<td>-</td>
<td>0.00</td>
<td>-</td>
</tr>
<tr>
<td>information</td>
<td>0.00</td>
<td>0.57</td>
<td>-</td>
<td>0.47</td>
<td>-</td>
</tr>
</tbody>
</table>
1. Clustering Approach to Word Semantics

Clustering vectors to visualize similarity in co-occurrence matrices (Rohde et al. 2005)

Use whatever clustering algorithm you prefer to determine ”related” words

Application: Extend query with related words automatically
Apply SVD-based Dimension Reduction

- principal components: Word context vectors
- principal axes: Number of clusters
2. Embedding Approaches to Word Semantics

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word
Represent the meaning of words – word2vec

- 2 basic structural models:
  - **Continuous Bag of Words (CBOW)**: use a window of words to predict the middle word
  - **Skip-gram (SG)**: use a word to predict the surrounding ones in window.
Word2vec – Continuous Bag of Word

- E.g. “The cat <sat> on floor”
  - Window size = 2
Index of cat in vocabulary: 1

Input layer

Hidden layer

Output layer

one-hot vector

sat one-hot vector
We must learn $W$ and $W'$

$W_{V \times N}$

$W'_{N \times V}$

N will be the size of word vector
Input layer

\( W_{V \times N}^T \times x_{cat} = v_{cat} \)

\[
\begin{bmatrix}
0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \ldots & \ldots & 3.2 \\
0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 3.6 & \ldots & \ldots & 6.1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots \\
0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 2.0 & \ldots & \ldots & 1.2
\end{bmatrix}
\times
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
= 
\begin{bmatrix}
2.4 \\
2.6 \\
\vdots \\
1.8
\end{bmatrix}

Output layer

\( W_{V \times N}^T \times x_{on} = v_{on} \)

\( \hat{v} = \frac{v_{cat} + v_{on}}{2} \)

Hidden layer

N-dim

V-dim

sat

V-dim

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

\ldots

0
Input layer

\[ W_{V \times N}^T \times x_{on} = v_{on} \]

\[
\begin{pmatrix}
0.1 & 2.4 & 1.6 & \mathbf{1.8} & 0.5 & 0.9 & \ldots & \ldots & \ldots & 3.2 \\
0.5 & 2.6 & 1.4 & \mathbf{2.9} & 1.5 & 3.6 & \ldots & \ldots & \ldots & 6.1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\
0.6 & 1.8 & 2.7 & \mathbf{1.9} & 2.4 & 2.0 & \ldots & \ldots & \ldots & 1.2
\end{pmatrix}
\]

\[ \times \begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\vdots \\
1 \\
0 \\
0 \\
0 \\
0 \\
\vdots \\
0
\end{pmatrix} = \begin{pmatrix}
1.8 \\
2.9 \\
\vdots \\
1.9
\end{pmatrix} \]

Output layer

\[ W_{V \times N}^T \times x_{cat} = v_{cat} \]

\[ \hat{v} = \frac{v_{cat} + v_{on}}{2} \]

Hidden layer

\[ N \text{-dim} \]

V-cat

\[ x_{cat} \]

V-on

\[ x_{on} \]
\[
\hat{y} = \text{softmax}(z)
\]

\[
W'_{N \times V} \times \hat{v} = z
\]

N will be the size of word vector
A logistic function or logistic curve is a common "S" shape (sigmoid curve), with equation:

\[ f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \]

where

- \( e \) is the natural logarithm base (also known as Euler's number),
- \( x_0 \) is the x-value of the sigmoid's midpoint,
- \( L \) is the curve's maximum value, and
- \( k \) is the steepness of the curve.\(^{[1]}\)
The **softmax function**, or **normalized exponential function**, is a generalization of the **logistic function** that "squashes" a $K$-dimensional vector $\mathbf{z}$ of arbitrary real values to a $K$-dimensional vector $\sigma(\mathbf{z})$ of real values in the range $[0, 1]$ that add up to 1. The function is given by

$$\sigma : \mathbb{R}^K \rightarrow [0, 1]^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$ 

In **probability theory**, the output of the softmax function can be used to represent a **categorical distribution** – that is, a **probability distribution** over $K$ different possible outcomes.
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$

$$W'_{N \times V} \times \hat{v} = z$$

$$\hat{y} = softmax(z)$$

$V$-dim

$N$-dim

$V$-dim

$N$ will be the size of word vector
We can consider either $W$ or $W'$ as the word's representation.
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

\[ d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\| \|w_x\|} \]

man:woman :: king:?

+ king [ 0.30 0.70 ]
- man [ 0.20 0.20 ]
+ woman [ 0.60 0.30 ]

queen [ 0.70 0.80 ]
Word Analogies
What is word2vec?

- word2vec is **not** a single algorithm
- It is a **software package** for representing words as vectors, containing:
  - Two distinct models
    - CBoW
    - **Skip-Gram** (SG)
  - Various training methods
    - **Negative Sampling** (NS)
    - Hierarchical Softmax
  - A rich preprocessing pipeline
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words
Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little wampimuk hiding in the tree.

“word2vec Explained…”
Goldberg & Levy, arXiv 2014
Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little *wampimuk* hiding in the tree.

“word2vec Explained…”
Goldberg & Levy, arXiv 2014
Marco saw a **furry little wampimuk** hiding **in** the tree.

<table>
<thead>
<tr>
<th>words</th>
<th>contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>wampimuk</td>
<td>furry</td>
</tr>
<tr>
<td>wampimuk</td>
<td>little</td>
</tr>
<tr>
<td>wampimuk</td>
<td>hiding</td>
</tr>
<tr>
<td>wampimuk</td>
<td>in</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ D \text{ (data)} \]

“word2vec Explained…”
Goldberg & Levy, arXiv 2014
Skip-Grams with Negative Sampling (SGNS)

- SGNS finds a vector $\vec{w}$ for each word $w$ in our vocabulary $V_W$
- Each such vector has $d$ latent dimensions (e.g. $d = 100$)
- Effectively, it learns a matrix $W$ whose rows represent $V_W$
- **Key point:** it also derives a similar auxiliary matrix $C$ of context vectors
- In fact, each word has two embeddings

\[ \begin{align*}
\text{w:} \text{wampimuk} &= \langle -3.1, 4.15, 9.2, -6.5, \ldots \rangle \\
\neq & \\
\text{c:} \text{wampimuk} &= \langle -5.6, 2.95, 1.4, -1.3, \ldots \rangle
\end{align*} \]
Skip-Grams with Negative Sampling (SGNS)

“word2vec Explained…”
Goldberg & Levy, arXiv 2014
### Skip-Grams with Negative Sampling (SGNS)

- **Maximize:** \( \sigma(\vec{w} \cdot \vec{c}) \)
  - \( c \) was **observed** with \( w \)

<table>
<thead>
<tr>
<th><strong>Words</strong></th>
<th><strong>Contexts</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>wampimuk</td>
<td>furry</td>
</tr>
<tr>
<td>wampimuk</td>
<td>little</td>
</tr>
<tr>
<td>wampimuk</td>
<td>hiding</td>
</tr>
<tr>
<td>wampimuk</td>
<td>in</td>
</tr>
</tbody>
</table>

“word2vec Explained…”
Goldberg & Levy, arXiv 2014
Skip-Grams with Negative Sampling (SGNS)

- **Maximize:** $\sigma(\vec{w} \cdot \vec{c})$
  - $c$ was **observed** with $w$

<table>
<thead>
<tr>
<th>words</th>
<th>contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>wampimuk</td>
<td>furry</td>
</tr>
<tr>
<td>wampimuk</td>
<td>little</td>
</tr>
<tr>
<td>wampimuk</td>
<td>hiding</td>
</tr>
<tr>
<td>wampimuk</td>
<td>in</td>
</tr>
</tbody>
</table>

- **Minimize:** $\sigma(\vec{w} \cdot \vec{c}')$
  - $c'$ was **hallucinated** with $w$

<table>
<thead>
<tr>
<th>words</th>
<th>contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>wampimuk</td>
<td>Australia</td>
</tr>
<tr>
<td>wampimuk</td>
<td>cyber</td>
</tr>
<tr>
<td>wampimuk</td>
<td>the</td>
</tr>
<tr>
<td>wampimuk</td>
<td>1985</td>
</tr>
</tbody>
</table>

“word2vec Explained…”
Goldberg & Levy, arXiv 2014
Skip-Grams with Negative Sampling (SGNS)

- “Negative Sampling”
- SGNS samples $k$ contexts $c'$ at random as negative examples
- “Random” = unigram distribution

$$P(c) = \frac{\#c}{|D|}$$

- ** Spoiler:** Changing this distribution has a significant effect
What is SGNS learning?

- Take SGNS’s embedding matrices ($W$ and $C$)
What is SGNS learning?

• Take SGNS’s embedding matrices ($W$ and $C$)
• Multiply them
• What do you get?

$W \cdot V_d \cdot V_e$

“Neural Word Embeddings as Implicit Matrix Factorization”
Levy & Goldberg, NIPS 2014
What is SGNS learning?

- A $V_W \times V_C$ matrix
- Each cell describes the relation between a specific word-context pair

$$\vec{w} \cdot \vec{c} = ?$$

“Neural Word Embeddings as Implicit Matrix Factorization”
Levy & Goldberg, NIPS 2014
What is SGNS learning?

- Levy&Goldberg [2014] proved that for large enough $d$ and enough iterations …
- … one obtains the word-context PMI matrix

\[
W_d V_c = \sum_j M_i^j P(M^j)
\]

"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014
What is SGNS learning?

- Levy&Goldberg [2014] **proved** that for large enough $d$ and enough iterations …
- … one obtains the word-context PMI matrix …
- shifted by a global constant

$$Opt (\overrightarrow{w} \cdot \overrightarrow{c}) = PMI(w, c) - \log k$$
What is SGNS learning?

- SGNS is doing something very similar to the older approaches

- SGNS factorizes the traditional word-context PMI matrix

- So does SVD!

- GloVe factorizes a similar word-context matrix
But embeddings are still better, right?

- Plenty of evidence that embeddings outperform traditional methods
  - “Don’t Count, Predict!” (Baroni et al., ACL 2014)
  - GloVe (Pennington et al., EMNLP 2014)

- How does this fit with our story?
The Big Impact of “Small” Hyperparameters

- word2vec & GloVe are more than just algorithms…

- Introduce new hyperparameters

- May seem minor, but make a big difference in practice
New Hyperparameters

• **Preprocessing** (word2vec)
  – Dynamic Context Windows
  – Subsampling
  – Deleting Rare Words

• **Postprocessing** (GloVe)
  – Adding Context Vectors

• **Association Metric** (SGNS)
  – Shifted PMI
  – Context Distribution Smoothing
Dynamic Context Windows

Marco saw a furry little wampimuk hiding in the tree.
Dynamic Context Windows

saw a furry little wampimuk hiding in the tree
Dynamic Context Windows

saw a furry little wampimuk hiding in the tree

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2vec</td>
<td>\frac{1}{4}</td>
<td>\frac{2}{4}</td>
<td>\frac{3}{4}</td>
<td>\frac{4}{4}</td>
<td>\frac{4}{4}</td>
<td>\frac{3}{4}</td>
<td>\frac{2}{4}</td>
<td>\frac{1}{4}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe</td>
<td>\frac{1}{4}</td>
<td>\frac{1}{3}</td>
<td>\frac{1}{2}</td>
<td>\frac{1}{1}</td>
<td>\frac{1}{1}</td>
<td>\frac{1}{2}</td>
<td>\frac{1}{3}</td>
<td>\frac{1}{4}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggressive</td>
<td>\frac{1}{8}</td>
<td>\frac{1}{4}</td>
<td>\frac{1}{2}</td>
<td>\frac{1}{1}</td>
<td>\frac{1}{1}</td>
<td>\frac{1}{2}</td>
<td>\frac{1}{4}</td>
<td>\frac{1}{8}</td>
</tr>
</tbody>
</table>

The Word-Space Model (Sahlgren, 2006)
Adding Context Vectors

- SGNS creates word vectors $\vec{w}$
- SGNS creates auxiliary context vectors $\vec{c}$
  - So do GloVe and SVD
Adding Context Vectors

- SGNS creates word vectors $\vec{w}$
- SGNS creates auxiliary context vectors $\vec{c}$
  - So do GloVe and SVD

- Instead of just $\vec{w}$
- Represent a word as: $\vec{w} + \vec{c}$

- Introduced by Pennington et al. (2014)
- Only applied to GloVe
Context Distribution Smoothing

- SGNS samples $c' \sim P$ to form **negative** $(w, c')$ examples

- Our analysis assumes $P$ is the unigram distribution

$$P(c) = \frac{\#c}{\sum_{c' \in V_C} \#c'}$$
Context Distribution Smoothing

- SGNS samples $c' \sim P$ to form negative $(w, c')$ examples

- Our analysis assumes $P$ is the unigram distribution

- In practice, it’s a smoothed unigram distribution

$$P^{0.75}(c) = \frac{(\#c)^{0.75}}{\sum_{c' \in V_C} (#c')^{0.75}}$$

- This little change makes a big difference
Context Distribution Smoothing

• We can **adapt** context distribution smoothing to PMI!

• Replace $P(c)$ with $P^{0.75}(c)$:

$$PMI^{0.75}(w, c) = \log \frac{P(w, c)}{P(w) \cdot P^{0.75}(c)}$$

• Consistently improves PMI on **every task**

• **Always use Context Distribution Smoothing!**
Represent the meaning of sentence/text

- Paragraph vector (2014, Quoc Le, Mikolov)
  - Extend word2vec to text level
  - Also two models: add paragraph vector as the input
Don’t Count, Predict! [Baroni et al., 2014]

- "word2vec is better than count-based methods"

- **Hyperparameter settings** account for most of the reported gaps

- Embeddings do not really outperform count-based methods

- No unique conclusion available
The Contributions of Word Embeddings

Novel Algorithms
(objective + training method)
• Skip Grams + Negative Sampling
• CBOW + Hierarchical Softmax
• Noise Contrastive Estimation
• GloVe
• ...

New Hyperparameters
(preprocessing, smoothing, etc.)
• Subsampling
• Dynamic Context Windows
• Context Distribution Smoothing
• Adding Context Vectors
• ...

What’s really improving performance?