Semantische Ähnlichkeit mit distributionellen Methoden: word embeddings

Katja Markert, mit einigen Folien von Dan Jurafsky und Ines Rehbein

Institut für Computerlinguistik
Uni Heidelberg
markert@cl.uni-heidelberg.de

January 9, 2023

Bisher und heute

1. Bisher: Lexika und Wortnetze
2. Bisher: Semantische Relationen
3. Jetzt: Wortähnlichkeit
4. Jetzt: distributionelle Methoden zur Wortrepräsentation und Bestimmung von Wortähnlichkeiten $\rightarrow$ **keine manuell kreiierte Ressourcen nötig**

Übersicht

Übersicht

Semantische Wortähnlichkeit: die Aufgabe

Welche der folgenden Wortpaare sind am ähnlichsten:

1. bank — fund
2. bank — enter
3. bank — man
4. bank — theorem

- **Synonymie:** Binäre Relation, WordNet synsets
- **Ähnlichkeit:** Graduell, schwächer, nicht direkt in WordNet enthalten
- **Ähnlichkeit:** eigentlich zwischen senses, nicht Worten
  - $bank_1$ ähnlich zu $fund_3$
  - $bank_2$ ähnlich zu $slope_5$
- Lexikalische Methoden meist auf senses, **(statische) distributionelle** auf Wortebene

Ähnlichkeit vs. Relationiertheit

- Ähnlichkeit: Auto, Fahrrad, Traktor
- Relationiertheit: Auto— Benzin, Auto — Lenkrad, Auto— Fahrer, Auto — lenken, Auto — verschmutzend

Verwandte Unterscheidung:

- **Paradigmatische Relationen:** (oft) austauschbare Einheiten. Teilen viele Eigenschaften. Inkludiert Synonyme (Auto, Automobil), Ko-Hyponyme (Auto/Traktor), Hyperonyme (Auto — Gefährt), Hyponyme (Auto — SUV), leider oft auch Antonyme
- **Syntagmatische Relationen:** Worte, die (häufig) in einer Äußerung aufeinander folgen.

Warum Wortähnlichkeit?

- Generierung
- Thesauruserstellung
- Entdeckung von Plagiaten
- Sprachtests (TOEFL)
- Historische Linguistik und Digital Humanities
- Insgesamt Verwendung von Embeddings: in allen Anwendungen in NLP (Machine Translation, Summarization . . .)

Beispiel: Historische Semantik

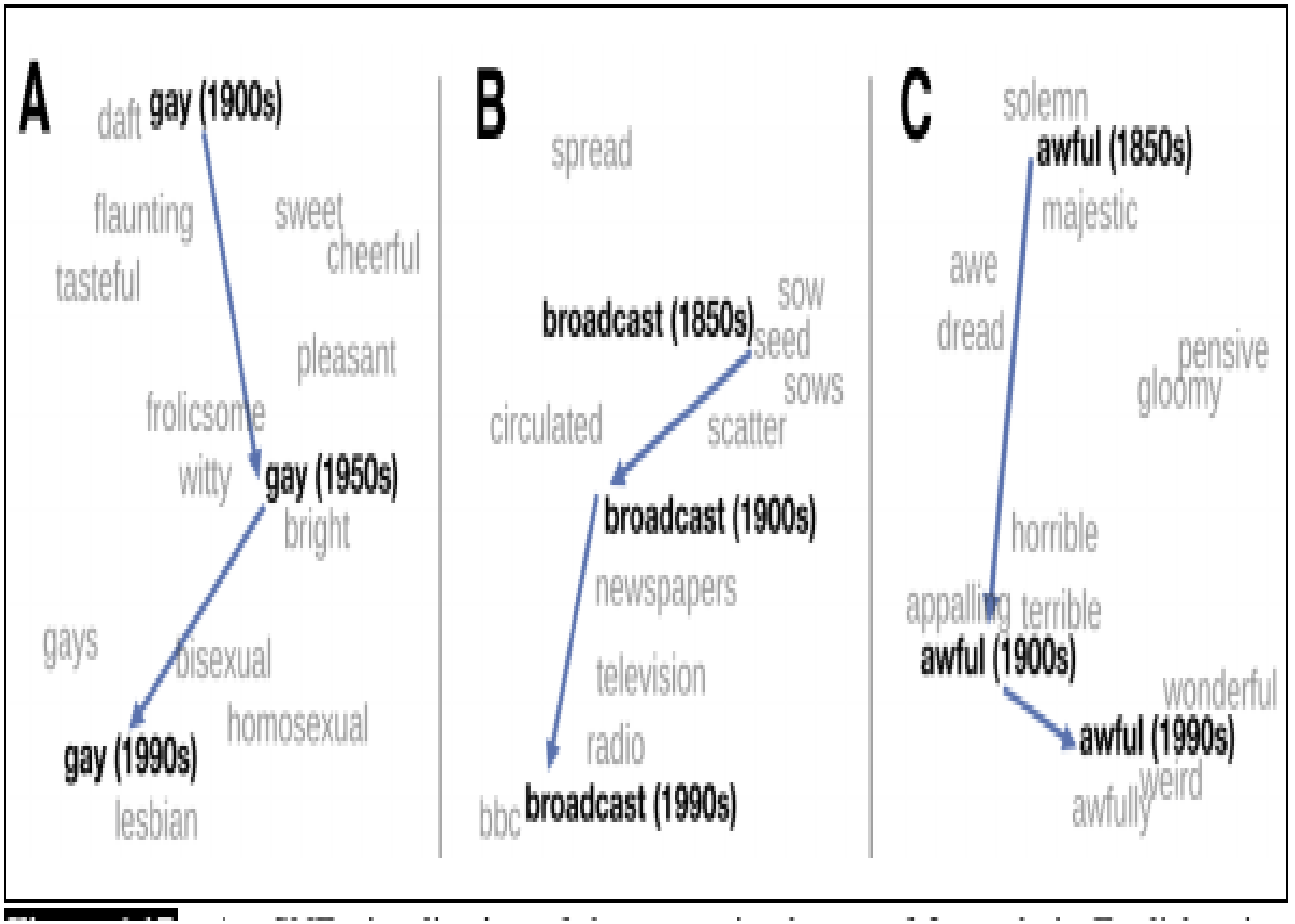


Bild aus Hamilton et al. 2016

Zwei Arten von Algorithmen für Ähnlichkeitsbestimmung

Lexikonbasiert (veraltet)

- Benutzung der Hierarchie
- Benutzung der glosses/Definitionen

Distributionell

- “You shall know a word by the company it keeps” (Firth)
- Distributional Hypothesis: Linguistische Einheiten mit gleicher Verteilung haben ähnliche Bedeutung
- Vektorbasiert: lineare Algebra, logistische Regression

Übersicht

Vektorraummodell

Embedding

Darstellung eines Wortes aus einem Vokabular V als Vektor im $\mathbb{R}^n$

$$f : V \rightarrow \mathbb{R}^n$$

Worte sind sich ähnlich, wenn sie im Vektorraum nah beieinander liegen

Worte als Vektoren: Versuch 1

One-Hot-Encoding:

1. Sammle das Vokabular V und ordne es in einer Liste: $\{aakvard, abnormal, abuse, abusive \ldots\ldots zebra, zoo\}$
2. Stelle jedes Wort mit einem Vektor im $\mathbb{R}^{|V|}$ da, der eine Eins hat bei seinem Index in der Vokabularliste und Nullen sonst

$$\overrightarrow{aakvard} = (1,0,0,0 \ldots 0,0)$$

$$\overrightarrow{abnormal} = (0,1,0,0 \ldots 0,0)$$

$$\overrightarrow{abuse} = (0,0,1,0 \ldots 0,0)$$

$$\overrightarrow{abusive} = (0,0,0,1, \ldots 0,0)$$

$$\overrightarrow{zebra} = (0,0,0,0, \ldots 1,0)$$

$$\overrightarrow{zoo} = (0,0,0,0, \ldots 0,1)$$

Löst dies unser Ähnlichkeitsproblem?

Worte als Vektoren: Versuch 2

Beispiel aus vorheriger Vorlesung (Osgood et al, 1957) mit 3 Dimensionen

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

- Stellt nur Konnotationen dar
- Vektoren zu kurz, um alle Bedeutungsaspekte zu erfassen
- Brauchen manuelle Annotation, Befragungen

Dritter und letzter Versuch: Distributionelles Vektorraummodell

You shall know a word by the company it keeps (Firth,57).

- Basiert auf distributioneller Hypothese
  - Ich trinke *Maruinho*.
  - Der *Maruinho* ist sehr stark.
  - *Maruinho* ist aus Weizen gebrannt.
  - Wenn er *Vodka* trinkt, ist er traurig.
  - *Vodka* ist ein starker Schnaps, den ich gerne trinke und oft trinke.
  - *Vodka* kann sogar aus Kartoffeln gebrannt werden.
  - *Vodka* mag ich nicht, da er mir zu stark ist.

Dritter und letzter Versuch: Distributionelles Vektorraummodell

You shall know a word by the company it keeps (Firth,57).

- Basiert auf distributioneller Hypothese
  - Ich trinke *Maruinho*.
  - Der *Maruinho* ist sehr stark.
  - *Maruinho* ist aus Weizen gebrannt.
  - Wenn er *Vodka* trinkt, ist er traurig.
  - *Vodka* ist ein starker Schnaps, den ich gerne trinke und oft trinke.
  - *Vodka* kann sogar aus Kartoffeln gebrannt werden.
  - *Vodka* mag ich nicht, da er mir zu stark ist.

Wort-Wort-Matrix (Term-Term-Matrix): Kunstbeispiel

- Ich trinke *Maruinho*.
- Der *Maruinho* ist sehr stark.
- *Maruinho* ist aus Weizen gebrannt.
- Wenn ich *Vodka* trinke, ist er traurig.
- *Vodka* ist ein starker Schnaps, den ich gerne trinke und oft trinke.
- *Vodka* kann sogar aus Kartoffeln gebrannt werden.
- *Vodka* mag ich nicht, da er mir zu stark ist.

	ich	trinke	stark	Weizen	gebr.	Schnaps	traurig	Kart.	Σ
Maruinho	1	1	1	1	1	0	0	0	5
Vodka	3	3	2	0	1	1	1	1	12
Σ	4	4	3	1	2	1	1	1	17

Terminologie am Kunstbeispiel

	ich	trinke	stark	Weizen	gebr.	Schnaps	traurig	Kart.	Σ
Maruinho	1	1	1	1	1	0	0	0	5
Vodka	3	3	2	0	1	1	1	1	12
Σ	4	4	3	1	2	1	1	1	17

- **Zielwörter**: Vodka, Maruinho
- Kontextwörter: Hier 8
- Kookkurrenzvektor in den Zeilen: Embedding in den $\mathbb{R}^n$
- Assoziationsmaß bestimmt Vektoreinträge: Hier Häufigkeit
- Kontextfenster: hier ein Satz
- 5: Anzahl der Paare, an denen *Mourinho* beteiligt ist
- 17: Anzahl aller Paare ($\neq$ Korpusgröße)

Wort-Wort-Matrix: Beispiel 2 (real world)

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- englische Wikipedia
- 4 Zielwörter
- +/- 4 Wörter als Kontextfenster
- hochdimensional (10, 000+) und sparse

Beispiel aus Jurafsky und Martin, 3rd edition, Kapitel 6

Beispiel 2 Visualisierung

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	**3325**	**3982**	378	5	13	...

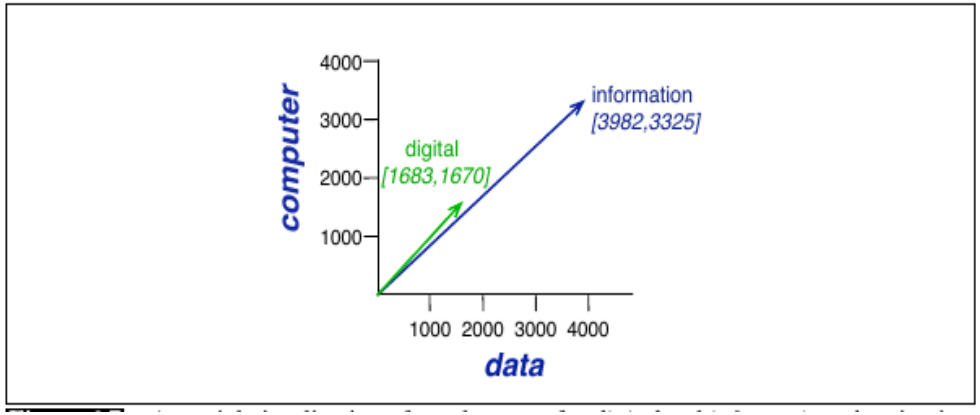


Bild aus Jurafsky und Martin, Kapitel 6, Figure 6.7

- Zwei Dimensionen/Kontextwörter: computer, data
- Zwei Zielwörter: digital, information

Vorteile von Worten als Vektoren

- Alle Methoden aus der linearen Algebra stehen uns zur Verfügung
- Input von Methoden aus dem Maschinellen Lernen brauchen Zahlen
- Distributionelle Vektorraummodelle können **ohne jede Vorannotation** aus großen Textkorpora erstellt werden
- Kookkurrenzvektor $\approx$ hochdimensionale Zusammenfassung des Verhaltens des Zielwortes
- Solange wir für alle Vektordarstellungen das gleiche Kontextvokabular, das gleiche Assoziationsmaß sowie die gleiche Fenstergröße nehmen, sind die embeddings verschiedener Wörter vergleichbar (da in den gleichen Vektorraum eingebettet): implizit in Matrix

Übersicht

Parameter

Schritte zur Konstruktion eines Vektorraummodels:

1. Wähle Korpus
2. Wähle Zielwörter
3. Wähle **Kontextworte** c_i, z.B. die 10,000 häufigsten Worte im Korpus.
4. Bestimme **Fenstergröße**, z. B. 5 (5 Worte links und 5 rechts).
5. Für jedes Zielwort w, bestimme **Kookkurrenzvektor** mithilfe eines **Assoziationsmaßes** von Ziel- zu Kontextwörtern (Beispiel: Kookkurrenzhäufigkeit)
6. Berechne mit **Ähnlichkeitsmaß**, ob Vektoren nah beiander liegen

Parameter 1, 2 und 3: Wähle Korpus sowie Zielworte und Kontextworte

- Korpusgröße relevant, auch Domäne
- Meist eine $n \times n$ Matrix (also Zielwörter = Kontextwörter)
- Benutze nur die häufigsten 10K bis 50K Wörter. Warum?
- Evtl. eliminiere Funktionswörter. Warum?
- Wörter oder Lemmas?
- *Sparse* matrices

Parameter 4: Fenstergröße

Ähnlichste Worte zu *car* (BNC)

2-Wort-Fenster	30-Wort-Fenster
van	drive
vehicle	park
truck	bonnet
motorcyle	windscreen
driver	hatchback
motor	headlight

Tendenz für kleine Fenster: paradigmatische Assoziationen

Tendenz für große Fenster: syntagmatische Assoziationen

Parameter 5: Assoziationsmaße

Mit Zielwort w_j und Kontextwort c_i

Assoc	$assoc(w_j, c_i)$
$assoc_{bin}$	1, wenn mindestens einmal w_j mit c_i vorkommt, 0 sonst
$assoc_{freq}$	$f(w_j, c_i)$
$assoc_{pmi}$	$\log \frac{p(w_j, c_i)}{p(w_j) \cdot p(c_i)}$
...	...

Für ein Wort w und Kontextworte $c_1, \ldots c_n$ können wir nun ein embedding konstruieren mit dem gewähltem Assoziationsmaß konstruieren:

$$\vec{w} = (assoc(w, c_1), assoc(w, c_2), assoc(w, c_3) \ldots, assoc(w, c_n))$$

Parameter 5 an Beispiel 2: $assoc_{freq}$

	computer	data	result	pie	sugar	Σ
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
Σ	4997	5673	473	512	61	11716

Reduziert auf 5 Kontextwörter

Parameter 5 an Beispiel 2: $assoc_{bin}$

	computer	data	result	pie	sugar	Σ
cherry	1	1	1	1	1	5
strawberry	0	0	1	1	1	3
digital	1	1	1	1	1	5
information	1	1	1	1	1	5
Σ	3	3	4	4	4	18

Zu viel Informationsverlust.

Parameter 5: (Positive) PMI als Assoziationsmaß

	computer	data	result	pie	sugar	Σ
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
Σ	4997	5673	473	512	61	11716

cherry kommt viel öfter mit *pie* vor als *strawberry*. Warum? → Lösung: (P)PMI

Parameter 5: PPMI

- **Pointwise mutual information.**

$$PMI(w_j, c_i) = log_2 \frac{p(w_j, c_i)}{p(w_j) \cdot p(c_i)}$$

- Setze $PMI(w_j, c_i) = 0$, wenn $f(w_j, c_i) = 0$
- Wahrscheinlichkeiten werden mit MLE geschätzt
- Verbesserung: **Positive pointwise mutual information**. Warum?

$$PPMI(w_j, c_i) = max(PMI(w_j, c_i), 0)$$

.

Parameter 5: PPMI an Beispiel 2

	computer	data	result	pie	sugar	Σ
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
Σ	4997	5673	473	512	61	11716

$$p(w = cherry, c = pie) = \frac{442}{11716} = 0.0377$$
$$p(w = cherry) = \frac{486}{11716} = 0.0414$$
$$p(c = pie) = \frac{512}{11716} = 0.0437$$
$$ppmi(cherry, pie) = \log_2 \frac{0.0377}{0.0414 \cdot 0.0437} = 4.38$$

$$p(w = strawberry, c = pie) = \frac{60}{11716}$$
$$p(w = strawberry) = \frac{80}{11716}$$
$$p(c = pie) = \frac{512}{11716}$$
$$ppmi(strawberry, pie) = 4.10$$

Parameter 5: PPMI an Beispiel 2

	computer	data	result	pie	sugar	Σ
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
Σ	4997	5673	473	512	61	11716

$$p(w = cherry, c = pie) = \frac{442}{11716} = 0.0377$$
$$p(w = cherry) = \frac{486}{11716} = 0.0414$$
$$p(c = pie) = \frac{512}{11716} = 0.0437$$
$$ppmi(cherry, pie) = \log_2 \frac{0.0377}{0.0414 \cdot 0.0437} = 4.38$$

$$p(w = strawberry, c = pie) = \frac{60}{11716}$$
$$p(w = strawberry) = \frac{80}{11716}$$
$$p(c = pie) = \frac{512}{11716}$$
$$ppmi(strawberry, pie) = 4.10$$

Parameter 5: Vollständige PPMI-Tabelle für Beispiel 3

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	**0**	**0**	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

- **Manche Nullen** kommen von Ursprungsnullfrequenzen
  PMI(*strawberry,computer*) = $\log_2 \frac{0}{p(strawberry) \cdot p(computer)} = 0$ per Definitionem
- Manche kommen von (anderem) negativem PMI:
  PMI(*cherry,computer*) = -6.7
- PPMI zu hoch für seltene Kontextwörter. Daher modifiziere

$$PPMI_\alpha(w_i, c_j) = max(\log_2 \frac{p(w_i, c_j)}{p(w_i) \cdot p_\alpha(c_j)}, 0)$$

mit

$$p_\alpha(c_j) = \frac{f(c_j)^\alpha}{\sum_c f(c)^\alpha}$$

Oft mit $\alpha = 0.75$.

Parameter 6: Möglichkeit I: Euklidische Metrik

Euklidische Metrik für zwei Vektoren $\vec{v}, \vec{w} \in \mathbb{R}^n$

$$d_2(\vec{v}, \vec{w}) := \sqrt{\sum_{i=1}^{n} (v_i - w_i)^2}$$

Dies entspricht der geometrischen Interpretation des "Luftlinienabstands" im $\mathbb{R}^2$ (oder $\mathbb{R}^3$):

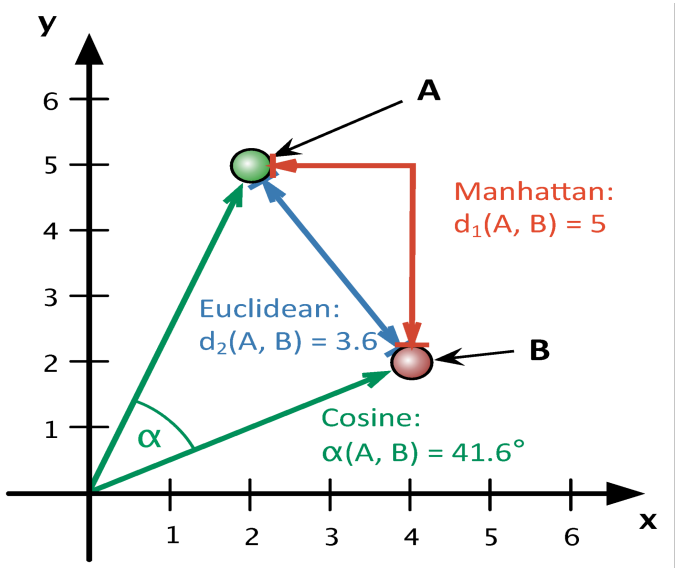


Bild (abgerufen 2020) von `http://dh2016.adho.org/static/data/290.html`

Parameter 6: Euklidischer Abstand am (reduzierten) Beispiel 2

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$d_2(\overrightarrow{cherry}, \overrightarrow{digital}) = \sqrt{(442-5)^2 + (8-1683)^2 + (2-1670)^2} = 2403.9$$

$$d_2(\overrightarrow{cherry}, \overrightarrow{info}) = \sqrt{(442-5)^2 + (8-3982)^2 + (2-3325)^2} = 5198.6$$

$$d_2(\overrightarrow{digital}, \overrightarrow{info}) = \sqrt{(5-5)^2 + (3982-1683)^2 + (3325-1670)^2} = 2832.7$$

Vorteile? Probleme?

Parameter 6: Euklidische Distanz am Beispiel 2

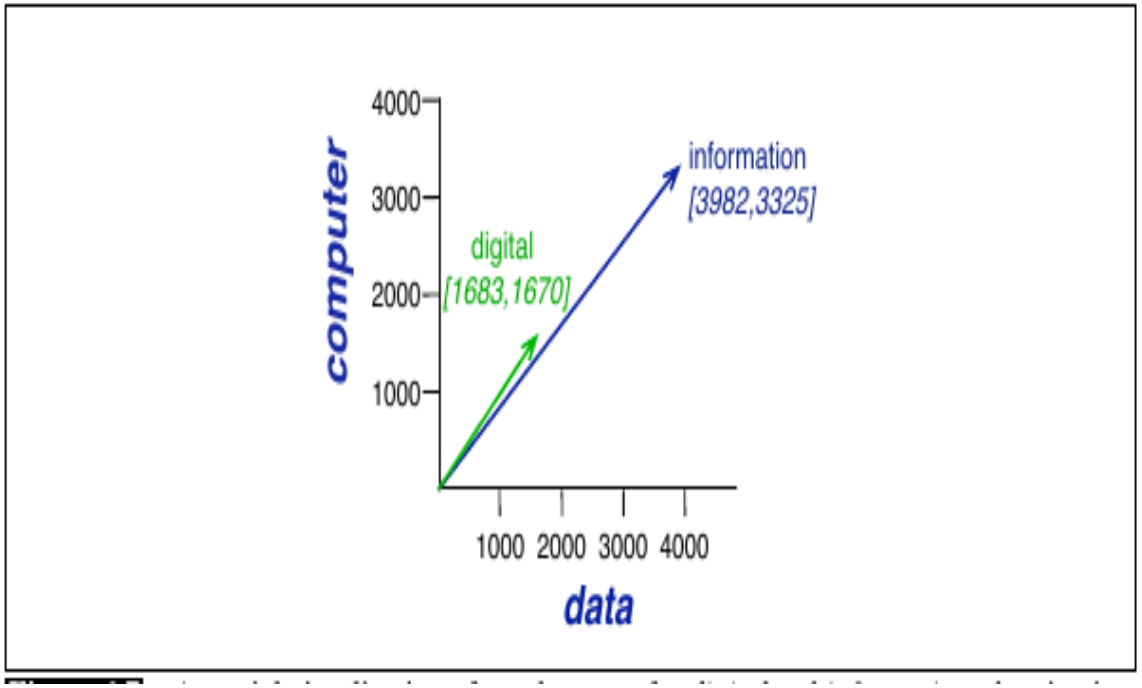


Bild aus Jurafsky und Martin, Kapitel 6, Figure 6.7

Parameter 6: Möglichkeit II: Basierend auf Skalarprodukt

Das **Skalarprodukt** zweier Vektoren im $\mathbb{R}^n$ ist (algebraisch) definiert als eine Abbildung $\cdot : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ mit $\vec{v} \cdot \vec{u} := \sum_{i=1}^{n} v_i \cdot u_i$

Beispiel im $\mathbb{R}^3$:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\overrightarrow{cherry} \cdot \overrightarrow{digital} = 442 \cdot 5 + 8 \cdot 1683 + 2 \cdot 1670$$

- Skalarprodukt groß, wenn hohe Werte in denselben Dimensionen
- Skalarprodukt klein, wenn Nullen in verschiedenen Dimensionen
- Symmetrisch
- Also gutes Ähnlichkeitsmaß
- Abhängigkeit von Vektorlänge!

Geometrische Definition des Skalarproduktes

Das **Skalarprodukt** zweier Vektoren im $\mathbb{R}^n$ ist (geometrisch) definiert als $\vec{v} \cdot \vec{u} := ||\vec{v}||_2 \cdot ||\vec{u}||_2 \cdot \cos\alpha$

wobei α der Winkel zwischen $\vec{v}$ und $\vec{u}$ ist.

- Beweis der Äquivalenz der beiden Definitionen:
  `https://en.wikipedia.org/wiki/Dot_product`

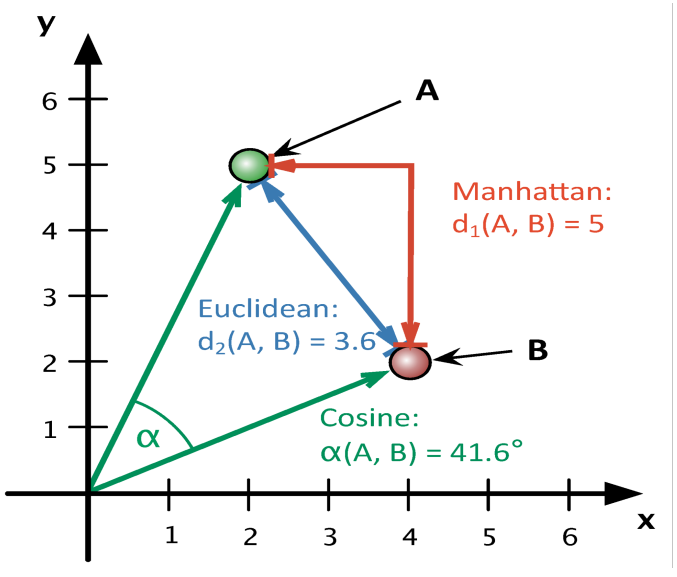


Bild (abgerufen 2020) von
`http://dh2016.adho.org/static/data/290.html`

Normalisiertes Skalarprodukt = cosine

Aus der geometrischen Definition des Skalarproduktes folgt:

$$cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}||_2 \cdot ||\vec{v}||_2} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

- Abstrahiert von Länge der Vektoren (Wie?)
- Werte zwischen -1 und +1. Für PPMI und frequenzbasierte Vektoren: zwischen 0 und 1
- Am höchsten (=1) , wenn Winkel zwischen Vektoren am kleinsten: Vektoren zeigen genau in die gleiche Richtung (sind parallel)
- Am niedrigsten (=-1), wenn in gegensätzliche Richtung
- Null, wenn orthogonal

Parameter 6: Berechnung der Kosinusähnlichkeit am Beispiel 2

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$cos(\overrightarrow{cherry}, \overrightarrow{digital}) = \frac{442 \cdot 5 + 8 \cdot 1683 + 2 \cdot 1670}{\sqrt{442^2 + 8^2 + 2^2} \cdot \sqrt{5^2 \cdot 1683^2 + 1670^2}} = 0.018$$

$$cos(\overrightarrow{cherry}, \overrightarrow{information}) = 0.017$$

$$\cos(\overrightarrow{digital}, \overrightarrow{information}) = 0.996$$

Parameter 6: Geometrische Interpretation am Beispiel 2

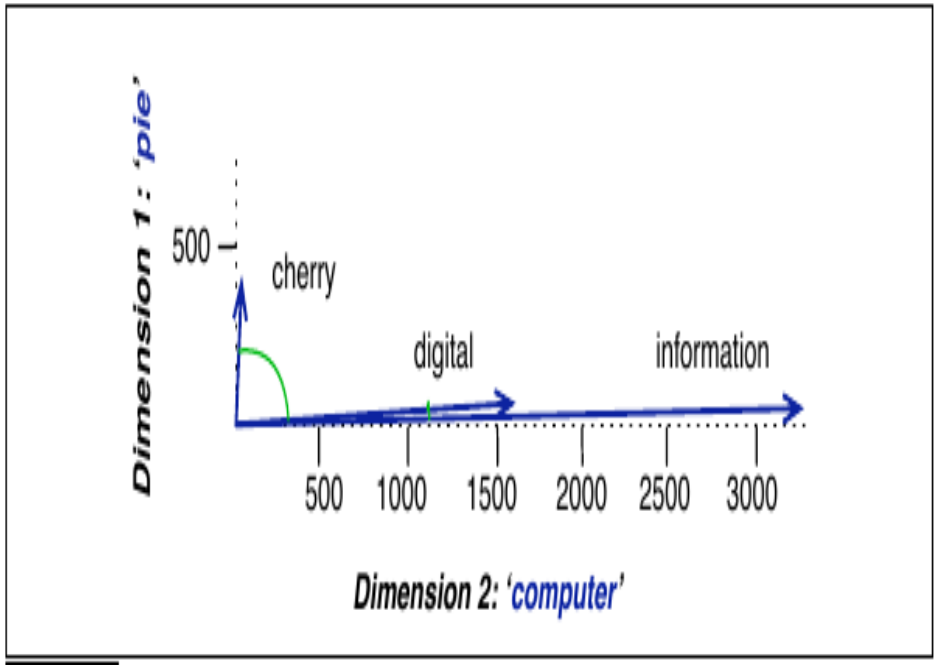


Bild aus Jurafsky und Martin, 3rd ed., Kapitel 6, Figure 6.8

- Am höchsten (=1) , wenn Winkel zwischen Vektoren am kleinsten: Vektoren zeigen genau in die gleiche Richtung (sind parallel)
- Null, wenn orthogonal

Zwischenfazit

- Vektormodelle sind eine einfache Möglichkeit, Wortähnlichkeiten zu berechnen. Benötigen keine manuell kreiierten Ressourcen.
- Setzen der Parameter: Fenstergröße, Vokabular, Ähnlichkeitsmaß, Assoziationsmaß
- Oft gute Settings: 10K bis 50K häufigste Worte in kleinem Fenster (+/- 1-5), (S)PPMI, Kosinusähnlichkeit
- Kosinusähnlichkeit wird auch weiter verwendet bei den dichten Embeddings, also unabhängig von der Berechnung der Embeddings

Übersicht

Probleme mit jetziger Methode zur Vektordarstellung

- Hochdimensional und speicherintensiv
- Sparse
- Overfitting durch zu viele Parameter mit geringen “counts”
- Zu viele fälschlich unterschiedliche Dimensionen:
  - *Sputnik* kommt mit *Kosmonaut* vor
  - *Apollo17* kommt mit *Astronaut* vor
  - *Kosmonaut* und *Astronaut* sind unterschiedliche Dimensionen. Damit lässt sich die “Ahnlichkeit zwischen *Sputnik* und *Apollo17* schwer fassen.
- Neue Worte, Sätze und Dokumente evtl. schwer integrierbar
- Daher weiterführend: dichte (und kürzere) Wortvektoren durch **Embeddings mit neuronalen Netzen**

Idee: Dont' count, predict

- Anstatt Kookkurrenzen zu zählen, **sage** die umgebenden Worte für jedes Wort im Korpus vorher

- Vorteile:
  - Niedrigdimensional (100-300, evtl. 10-1000) und dicht
  - **neue** Sätze und Dokumente leicht integrierbar
- Nachteil: Dimensionen **nicht leicht interpretierbar**

$$\text{Rauhaardackel} = \begin{bmatrix} 0.348 \\ 0.758 \\ 0.399 \\ 0.183 \\ -0.252 \\ -0.234 \\ 0.434 \\ -0.215 \end{bmatrix}$$

Wir behandeln nur Word2vec Skipgram with negative sampling (SGNS)

Zwei Arten von word2vec

- Continuous Bag Of Words (CBOW)
- Skipgram

Geg. Korpus

- iteriere über alle Worte und entweder
  - benutze Kontextworte, um ein target word vorherzusagen (CBOW), oder
  - **oder benutze derzeitiges target word um Kontextworte vorherzusagen (Skipgram)**

Skipgram – Intuition

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c_1	c_2	c_3	c_4	w_t	c_5	c_6	c_8

- target word an Position $t = 5$
- Fenstergröße L gegeben: hier $L = 2$, zwei rechts, zwei links

1. Iteriere über alle Wörter im Korpus
2. Für jede Position t, sei w_t das center/target word und alle anderen in Kontextgrösse L um w_t sind Kontext
3. Berechne für jedes Wort im Vokabular, wie wahrscheinlich es ist, dass es ein Kontextwort von w_t ist
4. Ändere Parameter so, dass diese Wahrscheinlichkeiten maximal werden
5. Behalte gelernte Parameter als embeddings

Classifier with Self-Supervision

Skipgram – Intuition

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c_1	c_2	c_3	c_4	w_t	c_5	c_6	c_8

- target word an Position $t = 5$
- Fenstergröße L gegeben: hier $L = 2$, zwei rechts, zwei links

1. Iteriere über alle Wörter im Korpus
2. Für jede Position t, sei w_t das center/target word und alle anderen in Kontextgrösse L um w_t sind Kontext
3. Berechne für jedes Wort im Vokabular, wie wahrscheinlich es ist, dass es ein Kontextwort von w_t ist
4. Ändere Parameter so, dass diese Wahrscheinlichkeiten maximal werden
5. Behalte gelernte Parameter als embeddings

Classifier with Self-Supervision

Skipgram – Intuition

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c_1	c_2	c_3	c_4	w_t	c_5	c_6	c_8

- target word an Position $t = 5$
- Fenstergröße L gegeben: hier $L = 2$, zwei rechts, zwei links

1. Iteriere über alle Wörter im Korpus
2. Für jede Position t, sei w_t das center/target word und alle anderen in Kontextgrösse L um w_t sind Kontext
3. Berechne für jedes Wort im Vokabular, wie wahrscheinlich es ist, dass es ein Kontextwort von w_t ist
4. Ändere Parameter so, dass diese Wahrscheinlichkeiten maximal werden
5. Behalte gelernte Parameter als embeddings

Classifier with Self-Supervision

Skipgram – Intuition

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c_1	c_2	c_3	c_4	w_t	c_5	c_6	c_8

- target word an Position $t = 5$
- Fenstergröße L gegeben: hier $L = 2$, zwei rechts, zwei links

1. Iteriere über alle Wörter im Korpus
2. Für jede Position t, sei w_t das center/target word und alle anderen in Kontextgrösse L um w_t sind Kontext
3. Berechne für jedes Wort im Vokabular, wie wahrscheinlich es ist, dass es ein Kontextwort von w_t ist
4. Ändere Parameter so, dass diese Wahrscheinlichkeiten maximal werden
5. Behalte gelernte Parameter als embeddings

Classifier with Self-Supervision

Skipgram – Intuition

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c_1	c_2	c_3	c_4	w_t	c_5	c_6	c_8

- target word an Position $t = 5$
- Fenstergröße L gegeben: hier $L = 2$, zwei rechts, zwei links

1. Iteriere über alle Wörter im Korpus
2. Für jede Position t, sei w_t das center/target word und alle anderen in Kontextgrösse L um w_t sind Kontext
3. Berechne für jedes Wort im Vokabular, wie wahrscheinlich es ist, dass es ein Kontextwort von w_t ist
4. Ändere Parameter so, dass diese Wahrscheinlichkeiten maximal werden
5. Behalte gelernte Parameter als embeddings

Classifier with Self-Supervision

Skipgram – Intuition

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c_1	c_2	c_3	w_t	c_4	c_5	c_6	c_7

$w_t = \textit{Maus}$, welche Wörter des Vokabulars kommen wahrscheinlich im jeweiligen Kontext vor?

Vokabular:

klein	lila
grau	Eiskunstlauf
Käse	schwimmen
Katze	Europa

Vorhersage als Klassifikation

Die	kleine	graue	Maus	frißt	den	leckeren	Käse
c	c	c_1	c_2	w	c_3	c_4	c

Von Prädiktion zu binärer Klassifikation:

$$p(+|w,c)$$

$$p(-|w,c) = 1 - p(+|w,c)$$

Berechnung von p

Grundidee

Wahrscheinlichkeit von $p(+|w,c)$ hoch, wenn die beiden Wörter w und c sich ähnlich sind

1. Approximiere Ähnlichkeit durch Skalarprodukt ($p(+|w,c) \approx w \cdot c$))
2. Verwandle Skalarprodukt mit Sigmoidfunktion in Wahrscheinlichkeit (siehe logistische Regression)

$$p(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+e^{-cw}}$$

sowie

$$p(-|w,c) = 1 - p(+|w,c) = 1 - \sigma(c \cdot w) = \sigma(-c \cdot w) = \frac{1}{1+e^{cw}}$$

Generalisiere zu mehreren Wörtern in einem Kontextfenster der Größe L

Grundidee

- Gleiche Wahrscheinlichkeitsverteilung per target word w
- Alle Kontextworte sind voneinander unabhängig

$$p(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log p(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Zwischenfazit

- Binärer Klassifizierer, der Vorkommenswahrscheinlichkeit aus Wortähnlichkeit ableitet
- Um diese Wahrscheinlichkeit zu berechnen, brauchen wir die Wortvektoren (einmal als Kontextvektor und einmal als Targetvektor): $2|V|$ Vektoren mit Dimension d

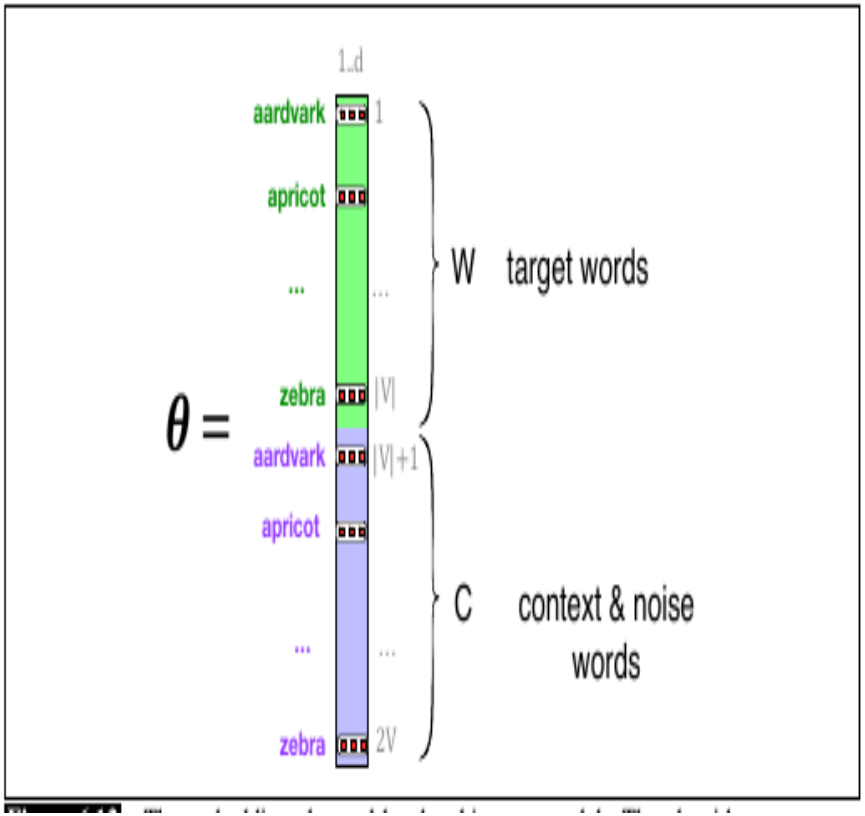


Bild aus Jurafsky und Martin, 3rd Edition, Kapitel 6, Figure 6.13

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus

Die kleine graue Maus frißt den leckeren Käse

Die kleine graue Maus frißt den leckeren Käse

Die kleine graue Maus frißt den leckeren Käse

Die kleine graue Maus frißt den leckeren Käse

Train samples

(Die,	kleine)
(Die,	graue)
(kleine,	Die)
(kleine,	graue)
(kleine,	Maus)
(graue,	Die)
(graue,	kleine)
(graue,	Maus)
(graue,	frißt)
(Maus,	kleine)
(Maus,	graue)
(Maus,	frißt)
(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Positive Samples (w, c_{pos}): leicht

Corpus								Train samples	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Die,	kleine)
								(Die,	graue)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(kleine,	Die)
								(kleine,	graue)
								(kleine,	Maus)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(graue,	Die)
								(graue,	kleine)
								(graue,	Maus)
								(graue,	frißt)
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

(http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model)

Trainingsdaten: Negative Samples?

Grundidee

Für ein positives Paar (w, c_{pos}) kreiiere negative Beispiele oder noise words (w, c_{neg}) zufällig mit $c_{neg} \in V$ mit $c_{neg} \neq w$ sowie $c_{neg} \neq c_{pos}$ sowie alle c_{neg} verschieden. (Warum?)

- pro positives Beispiel, kreiiere k negative Beispiele mit Parameter k
- Wähle noise words c_{neg} nach der Unigramverteilung hoch $\alpha \rightarrow$ $p_\alpha(c_{neg}) = f(c_{neg})^\alpha / Z$
  (weniger häufige Worte werden etwas häufiger gewählt)

Trainingsdaten: Negative Samples?

Grundidee

Für ein positives Paar (w, c_{pos}) kreiiere negative Beispiele oder noise words (w, c_{neg}) zufällig mit $c_{neg} \in V$ mit $c_{neg} \neq w$ sowie $c_{neg} \neq c_{pos}$ sowie alle c_{neg} verschieden. (Warum?)

- pro positives Beispiel, kreiiere k negative Beispiele mit Parameter k
- Wähle noise words c_{neg} nach der Unigramverteilung hoch $\alpha \rightarrow$ $p_\alpha(c_{neg}) = f(c_{neg})^\alpha / Z$
  (weniger häufige Worte werden etwas häufiger gewählt)

Negative Samples: Beispiel

Corpus

Die kleine graue **Maus** frißt den leckeren Käse

Train sample

(Maus, kleine)
(Maus, graue)
(Maus, frißt)
(Maus, den)

4 positive Beispiele. Mit $k = 2$ könnten wir 8 negative Beispiele bekommen:

(Maus,ich)	(Maus,sieben)
(Maus,Diagramm)	(Maus, wenn)
(Maus, immer)	(Maus, Tisch)
(Maus, Zeitung)	(Maus, wo)

Negative Samples: Beispiel

Corpus								Train sample	
Die	kleine	graue	Maus	frißt	den	leckeren	Käse	(Maus,	kleine)
								(Maus,	graue)
								(Maus,	frißt)
								(Maus,	den)

4 positive Beispiele. Mit $k = 2$ könnten wir 8 negative Beispiele bekommen:

(Maus,ich)	(Maus,sieben)
(Maus,Diagramm)	(Maus, wenn)
(Maus, immer)	(Maus, Tisch)
(Maus, Zeitung)	(Maus, wo)

Negative Samples: Impakt von k

1. Bessere Schätzung von der Verteilung der negativen Beispiele: größeres k heisst mehr Daten und bessere Schätzung
2. k ist ein Prior für die Wahrscheinlichkeit von positiven Beispiele: größeres k macht negative Beispiele wahrscheinlicher

Negative Samples: Auswahl

Wähle noise words nach der Unigramverteilung hoch $\alpha \rightarrow$ $p_\alpha(c_{neg}) = f(c_{neg})^\alpha / Z$:

$$p_\alpha(c_{neg}) = \frac{f(c_{neg})^\alpha}{\sum_{w'} f(w')^\alpha}$$

Beispiel mit $\alpha = 0.75$ und zwei Ereignissen von 100 Beobachtungen, wobei a 99mal vorkommt und b 1mal.

$$p_{0.75}(a) = \frac{99^\alpha}{99^\alpha + 1^\alpha} = 0.969$$

$$p_{0.75}(b) = \frac{1^\alpha}{99^\alpha + 1^\alpha} = 0.031$$

Ziel des Trainierens und Loss Funktion

1. Maximiere die Ähnlichkeiten von $(w, c_{pos}) \rightarrow$ beobachtete Wortepaare haben dann ähnliche embeddings
2. Minimiere die Ähnlichkeiten von $(w, c_{neg}) \rightarrow$ nicht beobachtete Wortpaare sind im Raum verteilt

Als Loss Function für ein positives Paar mit seinen k negativen Paaren:

$$L_{CE} = -\log[p(+|w, c_{pos}) \prod_{i=1}^{k} p(-|w, c_{neg_i})]$$

Umformulierung der Loss Function

$$
\begin{aligned}
L_{CE} &= -\log[p(+|w, c_{pos}) \prod_{i=1}^{k} p(-|w, c_{neg_i})] \\
&= -[\log p(+|w, c_{pos}) + \sum_{i=1}^{k} \log p(-|w, c_{neg_i})] \\
&= -[\log p(+|w, c_{pos}) + \sum_{i=1}^{k} \log(1 - p(+|w, c_{neg_i}))] \\
&= -[\log \sigma(w \cdot c_{pos}) + \sum_{i=1}^{k} \log \sigma(-w \cdot c_{neg_i})]
\end{aligned}
$$

- Maximiere die Wahrscheinlichkeiten gesehener Paare
- Minimiere die Wahrscheinlichkeiten nicht gesehener Paare, bzw maximiere die Wahrscheinlichkeiten, dass negative samples nicht gesehen werden

Trainieren von word2vec embeddings: Initialisierung

- Wähle Korpus
- Wähle Vokabular V (z.B 50,000 häufigsten Wörter)
- Wähle Kontextfenster f, Vektorlänge n, Anzahl der negative samples k, Parameter α
- Wir starten mit zufällig initialisierten $W \in \mathbb{R}^{|V| \times n}$ und $C \in \mathbb{R}^{|V| \times n}$ Matrizen. Initialisierung um 0 herum.
- Jedes Wort des Vokabulars hat zwei Vektoren: Zeile in W und Zeile in C

Trainieren von Word2Vec: die Trainingsphase

1. Gehe durch das ganze Trainingskorpus Position für Position mit einem "sliding window" von $2f+1$. Target word w in der Mitte.
2. Für jedes Kontextwort c_{pos} im sliding window um w:
   - Bilde positives Sample (w, c_{pos}) und schaue die passenden Vektoren $w^{(t)}$ in W und $c_{pos}^{(t)}$ in C nach
   - Wähle k negative Samples, bilde entsprechend negative Samples (w, c_{neg}) und schaue ebenfalls deren Vektoren nach.
   - Update die Vektoren wie folgt mit Learning Rate η

$$c_{pos}^{(t+1)} = c_{pos}^{(t)} - \eta \frac{\partial L_{CE}}{\partial c_{pos}} = c_{pos}^{(t)} - \eta[\sigma(c_{pos}^{(t)} \cdot w) - 1]w$$

$$c_{neg}^{(t+1)} = c_{neg}^{(t)} - \eta \frac{\partial L_{CE}}{\partial c_{neg}} = c_{neg}^{(t)} - \eta[\sigma(c_{neg}^{(t)} \cdot w)]w$$

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial L_{CE}}{\partial w} = w^{(t)} - \eta[(\sigma(c_{pos} \cdot w^{(t)}) - 1)c_{pos} + \sum_{i=1}^{k} (\sigma(c_{neg_i} \cdot w^{(t)})c_{ne}$$

3. Gehe zurück nach 1 (zwischen 3 und 50 Epochen)
4. Benutze w (oder $w + c$) als embedding eines Wortes.

Trainieren von Word2Vec: die Trainingsphase

1. Gehe durch das ganze Trainingskorpus Position für Position mit einem "sliding window" von $2f+1$. Target word w in der Mitte.
2. Für jedes Kontextwort c_{pos} im sliding window um w:
   - Bilde positives Sample (w, c_{pos}) und schaue die passenden Vektoren $w^{(t)}$ in W und $c_{pos}^{(t)}$ in C nach
   - Wähle k negative Samples, bilde entsprechend negative Samples (w, c_{neg}) und schaue ebenfalls deren Vektoren nach.
   - Update die Vektoren wie folgt mit Learning Rate η

$$c_{pos}^{(t+1)} = c_{pos}^{(t)} - \eta \frac{\partial L_{CE}}{\partial c_{pos}} = c_{pos}^{(t)} - \eta[\sigma(c_{pos}^{(t)} \cdot w) - 1]w$$

$$c_{neg}^{(t+1)} = c_{neg}^{(t)} - \eta \frac{\partial L_{CE}}{\partial c_{neg}} = c_{neg}^{(t)} - \eta[\sigma(c_{neg}^{(t)} \cdot w)]w$$

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial L_{CE}}{\partial w} = w^{(t)} - \eta[(\sigma(c_{pos} \cdot w^{(t)}) - 1)c_{pos} + \sum_{i=1}^{k} (\sigma(c_{neg_i} \cdot w^{(t)})c_{ne}$$

3. Gehe zurück nach 1 (zwischen 3 und 50 Epochen)
4. Benutze w (oder $w + c$) als embedding eines Wortes.

Trainieren von Word2Vec: die Trainingsphase

1. Gehe durch das ganze Trainingskorpus Position für Position mit einem "sliding window" von $2f+1$. Target word w in der Mitte.
2. Für jedes Kontextwort c_{pos} im sliding window um w:
   - Bilde positives Sample (w, c_{pos}) und schaue die passenden Vektoren $w^{(t)}$ in W und $c_{pos}^{(t)}$ in C nach
   - Wähle k negative Samples, bilde entsprechend negative Samples (w, c_{neg}) und schaue ebenfalls deren Vektoren nach.
   - Update die Vektoren wie folgt mit Learning Rate η

$$c_{pos}^{(t+1)} = c_{pos}^{(t)} - \eta \frac{\partial L_{CE}}{\partial c_{pos}} = c_{pos}^{(t)} - \eta[\sigma(c_{pos}^{(t)} \cdot w) - 1]w$$

$$c_{neg}^{(t+1)} = c_{neg}^{(t)} - \eta \frac{\partial L_{CE}}{\partial c_{neg}} = c_{neg}^{(t)} - \eta[\sigma(c_{neg}^{(t)} \cdot w)]w$$

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial L_{CE}}{\partial w} = w^{(t)} - \eta[(\sigma(c_{pos} \cdot w^{(t)}) - 1)c_{pos} + \sum_{i=1}^{k} (\sigma(c_{neg_i} \cdot w^{(t)})c_{ne}$$

3. Gehe zurück nach 1 (zwischen 3 und 50 Epochen)
4. Benutze w (oder $w + c$) als embedding eines Wortes.

Skipgram mit negative Sampling Fazit I

Grundidee I

Eventuell komplexe Wortvoraussage wird ersetzt durch binäre Klassifikation

Grundidee II

Positive samples direkt aus Korpus. Generiere negative Samples zufällig nach modifizierter Unigramverteilung

Grundidee III

Dann genügt logistische Regression mit Stochastic Gradient Descent, die positive und negative samples klassifiziert. Wir brauchen kein komplexes Netzwerk.

Grundidee IV

Klassifikation selbst interessiert uns nicht, aber die gelernten Gewichte sind Embeddings

Skipgram mit negative Sampling Fazit I

Grundidee I

Eventuell komplexe Wortvoraussage wird ersetzt durch binäre Klassifikation

Grundidee II

Positive samples direkt aus Korpus. Generiere negative Samples zufällig nach modifizierter Unigramverteilung

Grundidee III

Dann genügt logistische Regression mit Stochastic Gradient Descent, die positive und negative samples klassifiziert. Wir brauchen kein komplexes Netzwerk.

Grundidee IV

Klassifikation selbst interessiert uns nicht, aber die gelernten Gewichte sind Embeddings

Skipgram mit negative Sampling Fazit I

Grundidee I

Eventuell komplexe Wortvoraussage wird ersetzt durch binäre Klassifikation

Grundidee II

Positive samples direkt aus Korpus. Generiere negative Samples zufällig nach modifizierter Unigramverteilung

Grundidee III

Dann genügt logistische Regression mit Stochastic Gradient Descent, die positive und negative samples klassifiziert. Wir brauchen kein komplexes Netzwerk.

Grundidee IV

Klassifikation selbst interessiert uns nicht, aber die gelernten Gewichte sind Embeddings

Skipgram mit negative Sampling Fazit I

Grundidee I

Eventuell komplexe Wortvoraussage wird ersetzt durch binäre Klassifikation

Grundidee II

Positive samples direkt aus Korpus. Generiere negative Samples zufällig nach modifizierter Unigramverteilung

Grundidee III

Dann genügt logistische Regression mit Stochastic Gradient Descent, die positive und negative samples klassifiziert. Wir brauchen kein komplexes Netzwerk.

Grundidee IV

Klassifikation selbst interessiert uns nicht, aber die gelernten Gewichte sind Embeddings

Skipgram mit negative Sampling Fazit II

1. Worte, die häufig zusammen vorkommen, erhalten ähnliche embeddings
2. Das Wunder: auch Worte, die nicht zusammen vorkommen, können eng beieinander im Vektorraum landen (*cosmonaut,astronaut*). Warum?

Skipgram mit negative Sampling Fazit III

1. neue Sätze/Dokumente leicht integrierbar: passe Wortvektoren einfach an, startend von schon gelernten Matrizen
2. Negative Sampling: wir müssen in jedem Schritt nur Parameter für wenige Kontextvektoren ($k+1$) updaten, nicht für alle Worte!
3. statisch (= 1 Vektor pro Wort) und nicht kontextualisierbar $\rightarrow$ weiterführend kontextualisierte Embeddings wie Elmo, Bert

Skipgram mit negative Sampling Fazit III

1. neue Sätze/Dokumente leicht integrierbar: passe Wortvektoren einfach an, startend von schon gelernten Matrizen
2. Negative Sampling: wir müssen in jedem Schritt nur Parameter für wenige Kontextvektoren ($k+1$) updaten, nicht für alle Worte!
3. statisch (= 1 Vektor pro Wort) und nicht kontextualisierbar $\rightarrow$ weiterführend kontextualisierte Embeddings wie Elmo, Bert

Auch hier sind viele Parameter wichtig

- Fenstergröße für positive samples (oft +/-5)
- Anzahl negativer Samples k (meist zwischen 5 und 15)
- Verwendung welcher Vektoren: w oder c oder einer Kombi?
- Vektordimensionen zwischen 100 und 300 (50 und 1000)
- Algemeine Parameter in logistic regression: learning rate, Epochenanzahl
- Weitere Variationen: Subsampling häufiger Worte bei den positive samples, Distribution der negative samples (meist $\alpha = 0.75$)
- Meist werden die Vektoren normalisiert ($\rightarrow$ Einheitsvektoren) , bevor sie weiter verwendet werden
- Mehr hierzu in Embeddings-Seminar

Ressources

- Demo: `http://epsilon-it.utu.fi/wv_demo/`
- Prätrainierte 300-dimensionale Vektoren für das Englische auf GoogleNews: `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit`

Übersicht

Datensätze

Menschliche Ähnlichkeitsnormen. Zum Beispiel: WordSim353.
`http://www.gabrilovich.com/resources/data/wordsim353/wordsim353.html`

Wort1	Wort2	Rating
tiger	cat	7.35
tiger	tiger	10.00
drug	abuse	6.85
bread	butter	6.19
cup	coffee	6.58
cup	object	3.69
king	cabbage	0.23
king	queen	8.58
king	rook	5.92

353 Wortpaare aus 437 word types, von 13-16 Raters auf 0-10 Skala

Was fällt auf?

Datensätze

SimLex 999: Auch Adjektive und Verben. Nur Ähnlichkeit, nicht Relationiertheit.
`https://www.cl.cam.ac.uk/~fh295/simlex.html`

bread	cheese	1.95
cup	spoon	2.02
cup	jar	5.13
vanish	disappear	9.8

999 Paare auf 0-10 Skala, konkrete und abstrakte Worte.

Auch `https://multisimlex.com` für 12 weitere Sprachen sowie cross-lingual

Wie evaluiert man mit diesen Ähnlichkeits-Datensätzen?

1. Ordne menschliches Datensatz nach Ähnlichkeit
2. Bestimme Ähnlichkeit nach Sprachmodell und ordne

Wort1	Wort2	Mensch	Emb.	Mensch Rang	Emb. Rang	d_i	d_i^2
tiger	tiger	10.0	1	1	1	0	0
football	soccer	9.03	0.73	2	2	0	0
journey	voyage	8.19	0.68	3	3	0	0
book	paper	7.46	0.36	4	5	-1	1
movie	popcorn	6.19	0.44	5	4	1	1
drink	ear	1.31	0.09	6	7	1	1
professor	cucumber	0.31	0.06	7	8	1	1
king	cabbage	0.23	0.12	8	6	2	4

3. Berechne Spearman Rank Correlation:

$$\rho = 1 - \frac{6 \cdot \sum_i d_i^2}{n(n^2-1)} = 1 - \frac{6 \cdot 8}{8 \cdot 63} = 0.904$$

Spearman Rank Eigenschaften

- Nur für monotone Beziehungen zwischen zwei Zufallsvariablen
- Beschreibt, wie nah die Beziehung an perfekter Monotonie ist
- Zwischen -1 und 1
- 1 perfekt monoton steigend, -1 perfekt monoton fallende Beziehung, 0 keine Beziehung
- **Vorsicht**: Die vereinfachte Formel auf der leltzten Folie gilt nur, wenn alle der n items für beide Variablen verschiedene Werte annehmen. Sonst hat man “geteilte Ränge” und wir müssten anders berechnen

Resultate mit Spearman Rank (indikativ)

Auf WordSim 353:

Literatur	Modell	Korpus	Spearman
Agirre et al (2009)	Sparse Matrix mit χ^2	1.6 Tera	65
Levy & Goldberg (2014)	SPPMI mit k=15	1.5B	68.7
Levy & Goldberg (2014)	word2vec	1.5B	66.6
Socher (2016)	Glove	42B	75.9

$$SPPMI_k(w_j, c_i) = max(PMI(w_j, c_i) - \log_2 k, 0)$$

.

Auf Simlex-999: word2vec 0.37. Jetzige beste Modelle über 0.75

Resultate variieren stark mit Vektorlänge, Korpusgröße, Parametern.

Aus R. Socher's Folien für CS224d (2016) https://cs224d.stanford.edu/lectures/CS224d-Lecture3.pdf
sowie

Agirre et al (2009), Levy & Goldberg (2014)

sowie

Kiela et al 2015

Analogien mit Wortvektoren: Parallelogrammmethode

a is to b what a* is to ?

e.g. apple is to tree what grape is to ?

$$\tilde{b*} = argmax_x s(x, a* - a + b)$$

- Similarity: Cosine
- Exkludiere Inputworte und morphologische Variationen
- Evaluation: Accuracy

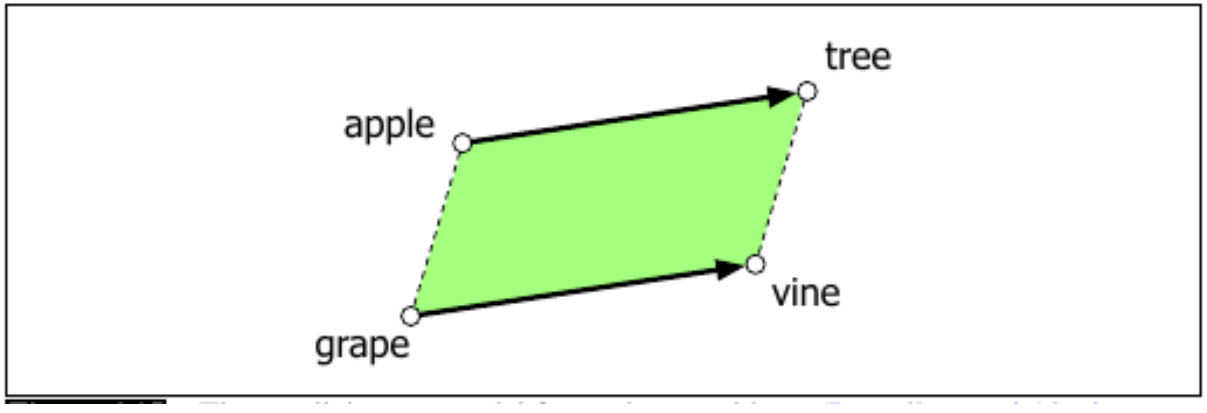
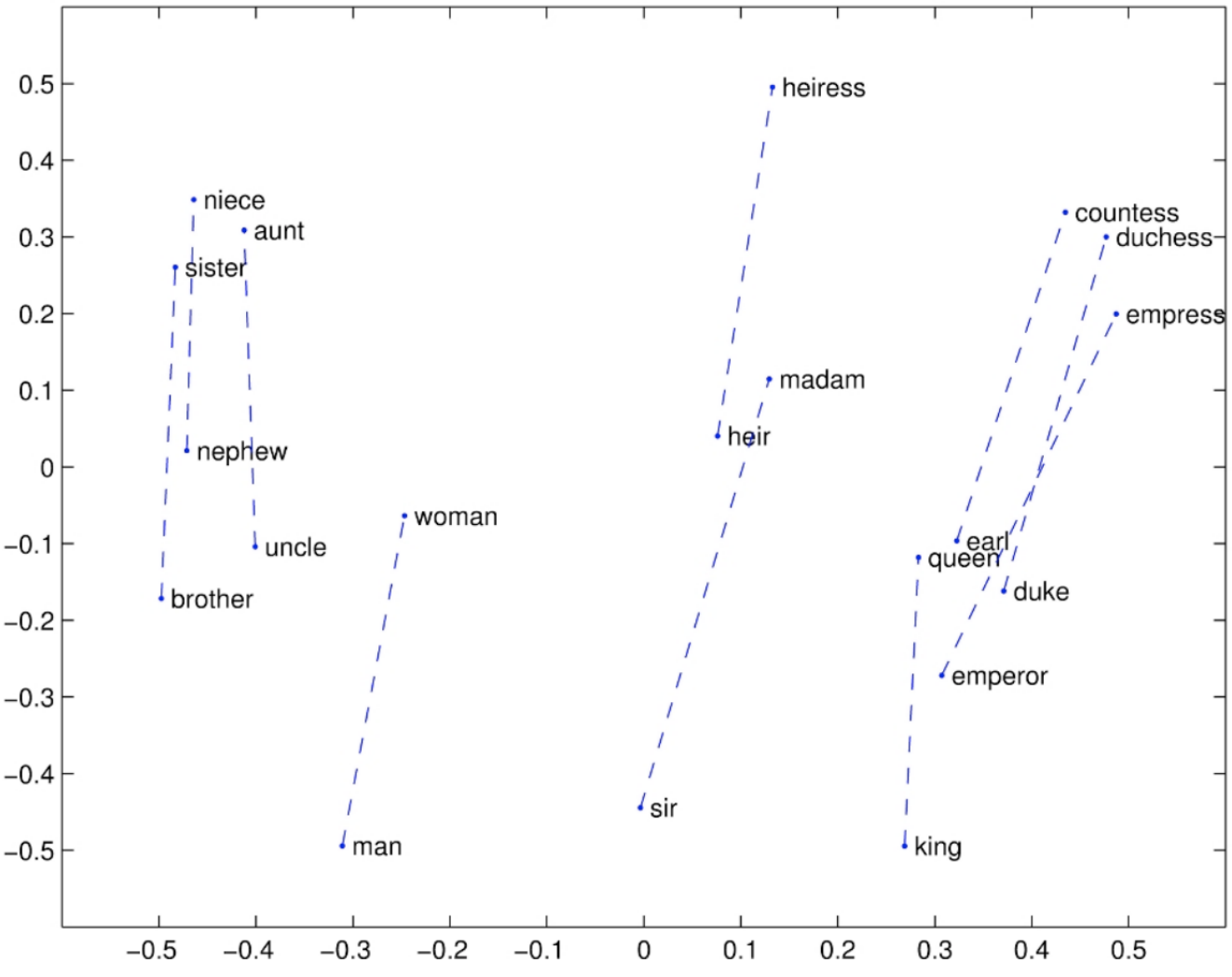


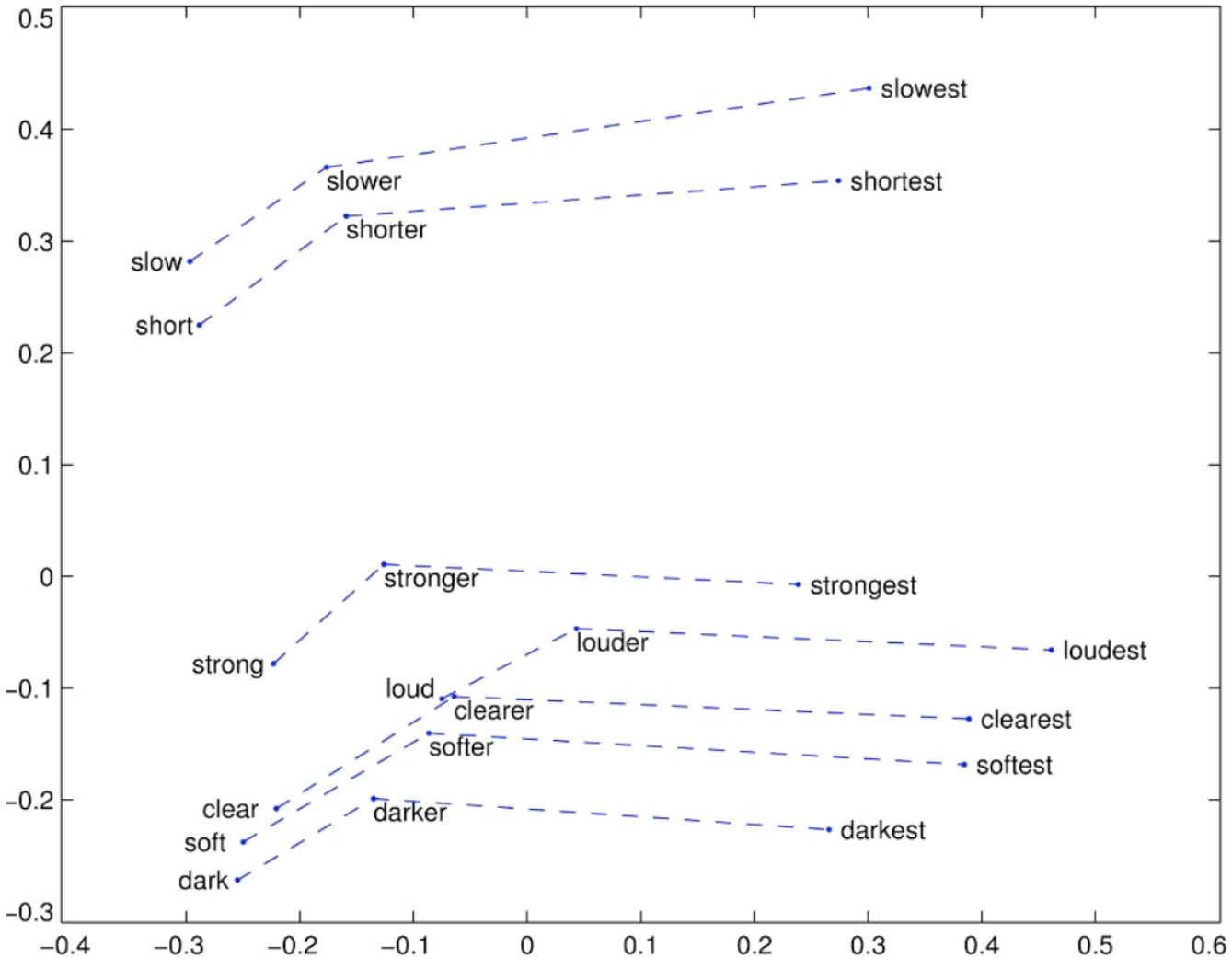
Bild aus Jurafsky und Martin, 3 Edition, Kapitel 6, Figure 6.15

Wortanalogien: Glove Beispiele



R. Socher's slides für CS224d (2016) https://cs224d.stanford.edu/lectures/CS224d-Lecture3.pdf

Wortanalogien: Glove Beispiele



R. Socher's slides für CS224d (2016) https://cs224d.stanford.edu/lectures/CS224d-Lecture3.pdf

Analogiedatensätze: GoogleAnalogy

19, 544 Syntaktische/morphologische, semantische und enzyklopädische Analogiefragen von

`http://download.tensorflow.org/data/questions-words.txt`

(Mikolov et al. 2013)

semantisch

uncle aunt boy girl

uncle aunt father mother

son daughter groom bride

prince princess nephew niece

...

Analogiedatensätze: GoogleAnalogy

19, 544 Syntaktische/morphologische, semantische und enzyklopädische Analogiefragen von

`http://download.tensorflow.org/data/questions-words.txt`

(Mikolov et al. 2013)

enzyklopädisch

Abuja Nigeria Accra Ghana
Abuja Nigeria Algiers Algeria
Abuja Nigeria Ankara Turkey
Abuja Nigeria Apia Samoa
Abuja Nigeria Asmara Eritrea
...

Analogiedatensätze: GoogleAnalogy

19, 544 Syntaktische/morphologische, semantische und enzyklopädische Analogiefragen von

`http://download.tensorflow.org/data/questions-words.txt`

(Mikolov et al. 2013)

morphologisch

bad worst big biggest

Albania Albanian Denmark Danish

dance dancing debug debugging

obvious obviously slow slowly

...

Resultate auf GoogleAnalogy für Original Skipgram Modell

Resultate in Akkuratheit (Prozent)

# Dim	Korpus	Sem	Syn	Insgesamt
300	783m	45.6	52.2	49.2
300	1.6B	52.2	55.1	53.8
600	783m	56.7	54.5	55.5
1000	6B	66.1	65.1	65.6

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean (2013): Efficient estimation of word representations in vector space. CoRR, abs/1301.3781

Evaluation Fazit

- Häufigste Evaluationen: Wortähnlichkeit und Analogien
- Man sollte möglichst viele verschiedene Testdaten und Tasks benutzen (seltene Worte, morphologisch komplexe, Relationiertheit vs. Ähnlichkeit)
- Ergebnisse hängen stark von Parameterwahl und Korpus ab
- Beste Resultate werden erst erzielt, wenn man aus “sparse” Vektoren dichte macht (via Latent semantic analysis oder bei neuralen word embeddings)
- Vorteil: funktioniert ohne manuelle Thesauruserstellung
- Nachteil: Vermischung von Relationen: relatedness, Ähnlichkeit etc.
- Nachteil: Sense-Unterscheidung wird ignoriert

Evaluation Fazit

- Häufigste Evaluationen: Wortähnlichkeit und Analogien
- Man sollte möglichst viele verschiedene Testdaten und Tasks benutzen (seltene Worte, morphologisch komplexe, Relationiertheit vs. Ähnlichkeit)
- Ergebnisse hängen stark von Parameterwahl und Korpus ab
- Beste Resultate werden erst erzielt, wenn man aus “sparse” Vektoren dichte macht (via Latent semantic analysis oder bei neuralen word embeddings)
- Vorteil: funktioniert ohne manuelle Thesauruserstellung
- Nachteil: Vermischung von Relationen: relatedness, Ähnlichkeit etc.
- Nachteil: Sense-Unterscheidung wird ignoriert

Übersicht

Referenzen: Distributionelle Methoden mit sparse vectors

- * Jurafsky und Martin, 3rd edition, Kapitel 6.
- Bullinaria, J. A., & Levy, J. P. (2012). Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and SVD. Behavior research methods, 44(3), 890-907.
- Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Pasca, M., & Soroa, A. (2009). A study on similarity and relatedness using distributional and wordnet-based approaches. In Naacl, 2009.

Literatur Word2vec

Die Darstellung lehnt sich eng an an Jurafsky und Martin, Introduction to Natural Language Processing, Edition 3, Kapitel 6.

Weitere essentielle Referenzen:

- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean (2013): Efficient estimation of word representations in vector space. CoRR, abs/1301.3781
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean (2013): Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, pages 3111–3119.
- Tomas Mikolov, Wen-Tau Yih, Geoffrey Zweig (2013): Linguistic regularieis in continuous space word representations. In: Proceedings of the 2013 conference of the North American Chapter of the Association for Computational Linguistics: Human language technologies. 2013, pages 746-751.
- Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In Advances in neural information processing systems (pp. 2177-2185).
- Kiela et al (2015): Specializing word embeddings for similarity or relatedness. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015. S. 2044-2048.

Toolkits und Implementationen

- Mit sparse Matrizen DISCO (2009)
  `http://www.linguatools.de/disco/disco_en.html`
- Word2vec:
  `https://code.google.com/archive/p/word2vec/`
- Glove: `http://nlp.stanford.edu/projects/glove/`

Referenzen: Mathematischer Hintergrund

- Gerd Fischer: Lineare Algebra. Eine Einführung für Studienanfänger
- Serie von Videos von 3Blue1Brown. Startet hier: `https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitg`

Referenzen: Evaluation, Anwendungen

- WordSim 353
  - Finkelstein et al. (2002): Placing Search in Context: The Concept Revisited, ACM Transactions on Information Systems, 20(1):116-131, January 2002.
  - Herunterladbar unter: `http://www.gabrilovich.com/resources/data/wordsim353/wordsim353.html`
- SimLex-999
  - Hill et al. (2015): SimLex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation. 2014. Computational Linguistics. 2015.
  - `https://fh295.github.io/simlex.html`
- Google Analogy: `http://download.tensorflow.org/data/questions-words.txt`
- Lastra-Diaz et al (2019): A reproducible survey on word embeddings and ontology-based methods for word similarity: Linear combinations outperform the state of the art. In Engineering Applications of Artificial Intelligence, 2019.
- Hamilton et al (2016): Diachronic word embeddings reveal statistical laws of semantic change. In *ACL, 2016*