Rechercheaufgabe

July 31, 2020

TODOS:

- Übersicht über alle Verfahren geben in kapitel 3 (Tabelle?!)
- überprüfen Formulierung
- untertitel der grafik bei LSA in pdf überprüfen

Inhaltsverzeichnis

[6]:
```
import gensim
from gensim.models import Word2Vec
import gensim.downloader as gensim_downloader

import matplotlib.pyplot as plt
#%matplotlib inline
#%config InlineBackend.figure_format = 'svg'
from nltk import word_tokenize
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import urllib.request
```

```
from utils import preprocess_text, plot_word_embeddings
```

[ ]:
```
text_w2v = ["das kind sagt, dass es feuerwehrmann sein möchte",
            "der erwachsene sagt, dass er lieber kein feuerwehrmann sein␣
 ↪möchte",
            "das kind fragt, was der erwachsene denn lieber sein möchte",
            "der erwachsene sagt, dass er lieber polizist sein möchte"]
#text_w2v = [word_tokenize(sen) for sen in text_w2v]
df = pd.DataFrame(text_w2v, columns = ["text"])
```

[7]:
```
%%time
corpus = pd.read_csv("../corpora/small_amazon_reviews_electronic.csv")
corpus["review"] = corpus.review.apply(lambda x: preprocess_text(x))
texts = [word_tokenize(row["review"]) for idx, row in corpus.iterrows()]
```

```
CPU times: user 1min 1s, sys: 1.14 s, total: 1min 2s
Wall time: 1min 2s
```

[ ]:
```
url = 'http://cloud.devmount.de/d2bc5672c523b086/german.model'
urllib.request.urlretrieve(url, "german_model.bin")
```

1 Einführung

Das **Natural Language Processing** (kurz: NLP) befasst sich mit Methoden und Verfahren zur maschinellen Verarbeitung von natürlicher Sprache in Form von Worten, Texten oder ganzen Korpora. Bevor jedoch NLP Verfahren wie die Textklassifikation oder das Topic Modelling auf die Textdaten angewendet werden können, müssen diese in eine Darstellungsweise umgewandelt werden, mit der die Verfahren arbeiten können. Die rohen Textdaten werden daher in **Vektoren**, die aus Zahlen bestehen, umgewandelt. Dieser Vorgang nennt sich **Vektorisierung**. Ein Wort wie "Baum" kann dadurch als Vektor aufgefasst werden. Natürlich können auch andere Features aus den Texten als Vektoren dargestellt werden; so ist es auch möglich, einzelne Buchstaben, Phrasen, Sätze, Segmente oder ganze Texte als Features aus den Textdaten zu extrahieren und diese zu vektorisieren. In der folgenden Übersicht werden jedoch vorwiegend Wörter als Features verwendet.

1.1 Das Bag-of-Words Modell

Das wohl einfachste Verfahren zur Darstellung von Wörtern als Vektoren ist das **Bag-of-Words** Modell. Wörter werden hier als eindimensionale Vektoren (= einfache Zahlen) dargestellt, wobei jedes individuelle Wort einen individuellen eindimensionalen Vektor (auch: **Index**) zugeordnet bekommt. Die Zuordnungen jedes einzigartigen Wortes zu seinem Vektor werden in einem *Vokabular* gespeichert. Nun können mithilfe dieses Vokabulars auch ganze Sätze oder sogar Texte dargestellt werden. Dafür wird für jeden Satz/Text ein Vektor gebildet, der die gleiche Länge wie das Vokabular hat. Jedem Eintrag des Vektors wird anhand des Vokabulars ein Wort zugeordnet. Der Satz/Text wird dann als Vektor aus **absoluten Termhäufigkeiten** dargestellt, wo an jeder Stelle, an dem ein Wort aus dem Vokabular in dem Text vorkommt, die Häufigkeit des Wortes in dem jeweiligen

Satz/Text steht und an jeder anderen Stelle eine 0, da es kein einziges Mal vorkommt.Section **??** Dies soll im Folgenden anhand eines Code-Beispiels erläutert werden. Zuerst wird das Vokabular aller Texte dargestellt, bei dem die Wörter einem Index zugeordnet werden (es wird ab 0 gezählt). Danach werden die vektorisierten Sätze/Texte angezeigt.

1 Dies ist nur eine Möglichkeit, die Häufigkeit eines Wortes beim Bag-of-Words Modell darzustellen. Eine weitere Möglichkeit wären binäre Häufigkeiten, bei denen das Vorkommen eines Wortes mit einer 1 und die Abwesenheit eines Wortes mit einer 0 gekennzeichnet werden. Um häufigen Wörtern in den Dokumenten weniger Gewicht zu geben, da diese meist einen geringeren Informationsgehalt besitzen, ist es auch möglich, das Bag-of-Words Modell in der Kombination mit dem TF-IDF Maß aus dem Bereich des Information Retrievals zu verwenden, bei dem die Häufigkeit von Worten skaliert wird.

[ ]:
```
text = ["ich gehe nachher zur bank, um etwas geld zu holen",
        "ich möchte mich kurz auf die bank setzen",
        "um mir etwas zu essen zu holen, stand ich von der bank auf",
        "auf der hölzernen bank neben der bank liegt noch geld"]

vectorizer = CountVectorizer()
vector = vectorizer.fit_transform(text)
print(vectorizer.vocabulary_)
```

[ ]:
```
print(vector.toarray())
```

1.2 Grenzen des Bag-of-Words Modells

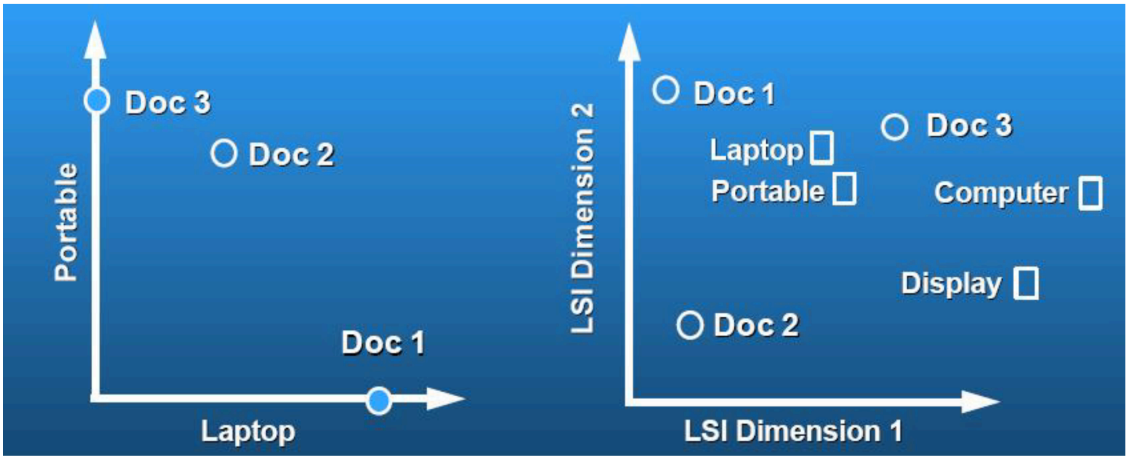
Aufgrund seiner Einfachheit ist das Bag-of-Words Modell leicht verständlich und sehr schnell umsetzbar. Es hat jedoch eine Reihe an Nachteilen, von deinen einige im Folgenden kurz erläutert werden:

- **Keine Informationen über Reihenfolge der Wörter.** Beim Bag-of-Words Modell wird jegliche Information über die Reihenfolge der Wörter verworfen, der Kontext eines Wortes bleibt unberücksichtigt. Dies wird auch durch den Namen dieses Modells deutlich: Die Bezeichnung "bag" (deutsch: Sack) soll darauf hinweisen, dass alle Informationen über die Struktur oder Reihenfolge der Wörter im Dokument verworfen werden, da sie metaphorisch in einen "Sack" geworfen werden. Die Reihenfolge lässt sich auch nicht im Nachhinein rekonstruieren. Insgesamt gehen somit sehr viele semantische Informationen verloren. Eine Lösung, bei der die Reihenfolge der Worte berücksichtigt werden kann, ist die Verwendung von **N-Grammen** oder **LSA** (Kapitel 2 TODO).
- **Spärlichkeit von Wortvektoren.** Umso mehr verschiedene Worte in den verwendeten Texten vorkommen, umso größer wird das Vokabular. Dies kann oft zu sehr spärlichen (engl. *sparse*) Wortvektoren führen. Besteht das Vokabular aus 500000 Worten, ein Text aber nur aus 50 verschiedenen Worten, sind nur 0.01% der Stellen des 500000 langen Wortvektors mit Einsen besetzt, der Rest nur mit Nullen. Dies führt dazu, dass eine große Menge an Rechenspeicher für die Verarbeitung der riesigen Matrizen benötigt wird. Weiterhin werden wenige Informationen in sehr großen Repräsentationsräumen benutzt, wodurch es für einige NLP Verfahren und Modelle problematisch ist, diese wenigen Informationen effizient zu nutzen. Eine Lösung bieten dichtbesetzte **Word Embeddings**, die in den nächsten Kapiteln behandelt werden.

- **Abbildung der Mehrdeutigkeit von Worten**. Wörter können trotz gleicher Schreibweise mehrere Bedeutungen haben, welche sich durch den Kontext des Wortes zeigen können. Dies wird durch das Bag-of-Words Modell nicht abgebildet. Eine mögliche Lösung wäre die Verwendung von **kontextabhängigen Word Embeddings** wie die **BERT-Embeddings** in Kapitel 3.6 (TODO).

2 LSA

Latent Semantic Analysis (kurz: LSA, auch: *Latent Semantic Indexing*) ist ein Verfahren aus dem Bereich des Information Retrievals aus dem Jahre 1990. Bei diesem Verfahren werden Dokumente und Terme (repräsentiert durch eine **Term-Dokument Matrix**) in einem latenten Raum abgebildet, der aus **Konzepten** (oder **Hauptkomponenten**) besteht. Dokumente, die ähnlich zueinander sind, d.h. aus ähnlichen Konzepten bestehen, werden in diesem Raum näher beieinander platziert. Dies wird durch die folgende Grafik deutlich.



Grafik von Susan Dumais, siehe Präsentation.

Das Ziel der LSA ist es, die Konzepte innerhalb der Dokumente zu finden. Dabei greift das Verfahren auf eine Technik der linearen Algebra zurück, der **Singulärwertzerlegung** (englisch: Singular Value Decomposition). Die Idee dabei ist, dass die Term-Dokument Matrix aus **Hauptdimensionen**, welche die wichtigen Konzepte der Dokumente beinhalten, und aus weniger aussagekräftigen Dimensionen mit unwichtigen Termen besteht. Mithilfe der Singulärwertzerlegung wird die originale Term-Dokument Matrix in drei Matrizen aufgeteilt, wobei die beiden äußeren Matrizen aus den linken bzw. rechten orthonormalen Eigenvektoren bestehen und die mittlere Matrix eine Diagonalmatrix ist, die die singulären Werte der Originalmatrix enthält. Mit Hilfe dieser Zerlegung kann eine **Approximation** der Originalmatrix mit einer kleiner dimensionierten Matrix erreicht werden. Die singulären Werte in der Diagonalmatrix sind nach ihrer Größe absteigend geordnet. Singulärwerte, die unter einem bestimmten Schwellenwert liegen, werden entfernt. Auch in den anderen Matrizen werden entsprechende Zeilen oder Spalten entfernt. Mit Hilfe der reduzierten Matrizen erhält man durch Matrixmultiplikation die optimale Approximation der Originalmatrix, die kleiner als die originale Term-Dokument Matrix ist, da Informationen aus den weniger aussagekräftigen Dimensionen verworfen wurden. Weiterhin werden bei der Dimensionsreduktion auch ähnliche Konzepte zusammengefasst, so werden z.B. Worte wie “Tür” und “Tor” in einem Konzept zusammengefasst.

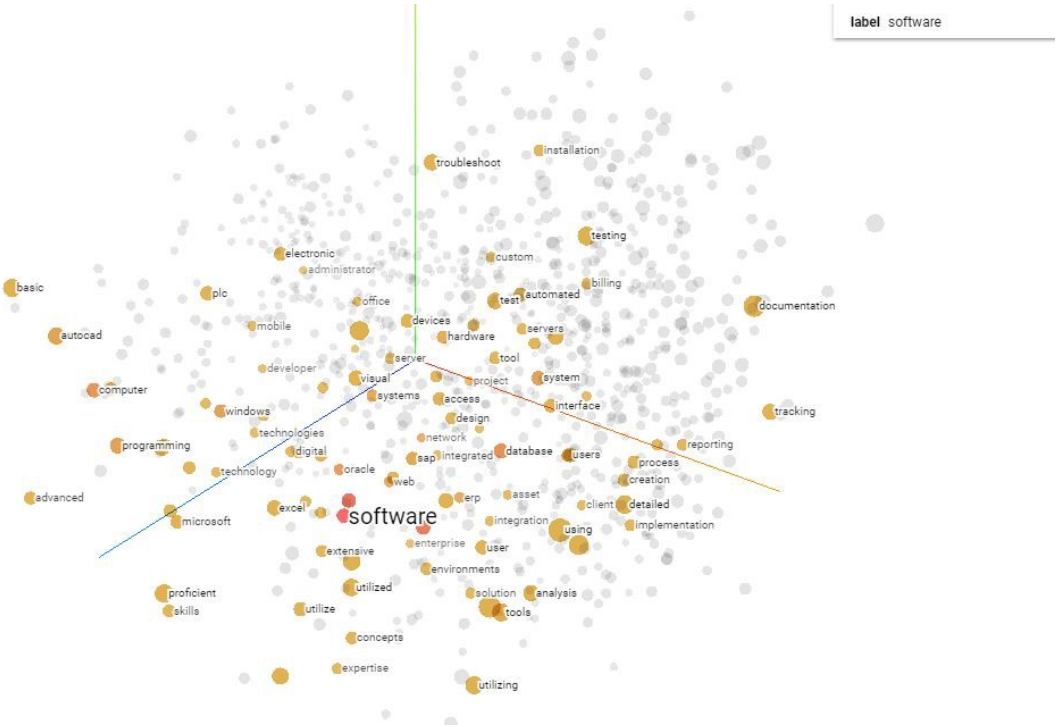
Der Vorteil der LSA ist, dass anders als beim Bag-of-Words Modell die **Semantik** der Dokumente wiedergegeben werden kann. Zudem werden **Synonyme** zusammengefasst. Probleme hat LSA jedoch mit der **Polysemie**. Zudem ist der Algorithmus sehr rechenaufwendig.

TODO: mehr?

3 Word Embeddings

3.1 Allgemeines

Word Embeddings (deutsch: Worteinbettungen) sind die Sammelbezeichnung für eine Reihe von Sprachmodellierungstechniken. Anders als beim Bag-of-Words Modell werden Wörter mit ähnlichen Bedeutungen ähnlich dargestellt, wobei der Kontext der Wörter berücksichtigt wird. Word Embeddings repräsentieren Wörter als Vektoren in einem multidimensionalen semantischen Raum. In diesem Raum werden Wörter, die ähnlich zueinander sind, näher beieinander platziert. Diesen Raum kann man sich etwa folgendermaßen vorstellen:



Die grundsätzliche Idee von Word Embeddings basiert auf der **Distributionellen Hypothese** von John Rupert Firth, die besagt, dass die Bedeutung eines Wortes durch sein Umfeld geprägt ist. Wörter, die einen ähnlichen Kontext besitzen, haben eine ähnliche Bedeutung. Anders als vorherige Vektorisierungsmethoden basieren Word Embeddings auf **Vorhersage-Modellen** (englisch: **prediction models**), indem Wörter durch Wahrscheinlichkeiten anstatt durch Häufigkeiten wie beim Bag-of-Words Modell oder bei der Latent Semantic Analysis dargestellt werden. Weiterhin sind die Wortvektoren von Word Embeddings anders als beim Bag-of-Words Modell **dichtbesetzt** (englisch: **dense**) und haben weitaus weniger Dimensionen (100-800 Dimensionen anstatt 100000-1000000, je nach der Größe des Vokabulars). Dadurch haben die Wortvektoren eine viel geringere Größe, bieten trotzdem eine effizientere und komplexere Darstellung der Wörter.

Eine weitere Besonderheit von Word Embeddings ist, dass es mit diesen möglich ist, eine Arithmetik mit Wörtern umzusetzen. So kann mit Wortvektoren "gerechnet" werden. Folgende Gleichungen sind mit Word Embeddings möglich:

```
König - Mann + Frau = Königin
```

```
London - Großbritannien + Deutschland = Berlin
```

Word Embeddings wurden ab 2013 durch die Einführung des Algorithmus **Word2Vec** populär, in den Jahren darauf folgten weitere Word Embedding Algorithmen wie **GloVe**, **FastText**, **ELMo** und **BERT**. In der folgenden Tabelle sind die wichtigsten Unterschiede der verschiedenen Embeddings zusammengefasst, genauere Erläuterungen der Embeddings befinden sich in den folgenden Kapiteln.

TODO: ausfüllen & mehr, gucken wie anders dargestellt werden kann

Word2Vec

Glove

FastText

ELMo

BERT

Entstehungsjahr

2013

2014

2016

2018

2018

Out of vocabulary Fehler

ja

ja

nein

nein

nein

Kontextsensitiv

nein

nein

nein

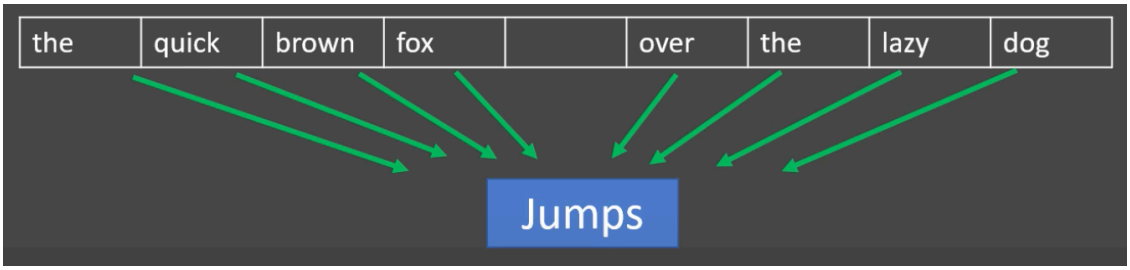
ja

ja

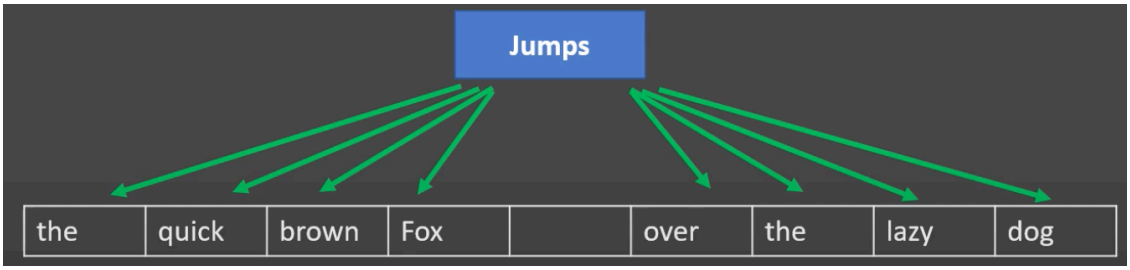
3.2 Word2Vec

Die Popularität von Word Embeddings ist vor allem **Word2Vec** geschuldet, welches 2013 von Tomas Mikolov und weiteren Mitgliedern von Google publiziert wurde. Word2Vec ist im eigentlichen Sinne kein alleinstehender Algorithmus, sondern besteht aus zwei Techniken: dem **Continous Bag of Words** (**CBOW**) und dem **Skip-gram Model**.

Continous Bag of Words versucht die Wahrscheinlichkeit eines Wortes oder einer Gruppe von Wörtern anhand eines gegebenen Kontext vorauszusagen:



Das **Skip-gram Model** funktioniert wie CBOW, nur anders herum. Das Modell versucht, anhand eines gegebenen Wortes den Kontext vorauszusagen:



Word2Vec nutzt wahlweise eine dieser Techniken, um aus rohen Textdaten mithilfe eines Neuronales Netzes Wortvektoren zu erstellen. Dabei kann Word2Vec bei vielen Implementierungen (z.b. bei Gensim) auf zwei Arten verwendet werden: Entweder werden bereits vortrainierte Embeddings geladen oder es werden eigene Embeddings auf eigenen Textdaten trainiert.

3.2.1 Vortrainierte Embeddings verwenden

Für die Demonstration der Nutzung von vortrainierten Word Embeddings wird ein deutsches Modell verwendet, welches auf Wikipedia- und Zeitungsartikeln trainiert wurde (Quelle). Als Framework wird **Gensim** verwendet.

[8]:
```
pre_w2v = gensim.models.KeyedVectors.load_word2vec_format('german_model.bin',
                                                          binary=True)
```

Im folgenden Code werden die 10 ähnlichsten Wörter zu “König” ausgegeben. Alle diese Wörter passen auch thematisch zu “König”, sie verbindet alle das Thema “royal” bzw. “Königshaus”.

[9]:
```
for t in pre_w2v.most_similar('Koenig', topn=5):
    print(f"{t[0]}: {np.around(t[1], decimals=3)}")
```

```
Prinz: 0.786
Koenigs: 0.736
Koenigin: 0.726
Jungkoenig: 0.705
Kaiser: 0.705
```

Auch das Rechenbeispiel aus Kapitel 3.1 TODO kann mit Word2Vec umgesetzt werden. Addiert man "König" mit "Frau" und subtrahiert "Mann", erhält man Begriffe zum Thema "Königin".

[13]:
```
for t in pre_w2v.most_similar(positive=['Koenig', 'Frau'],
                              negative=['Mann'], topn=5):
    print(f"{t[0]}: {np.around(t[1], decimals=3)}")
```

```
Koenigin: 0.752
Prinzessin: 0.715
Prinz: 0.688
Jungschuetzenkoenigin: 0.674
Majestaet: 0.659
```

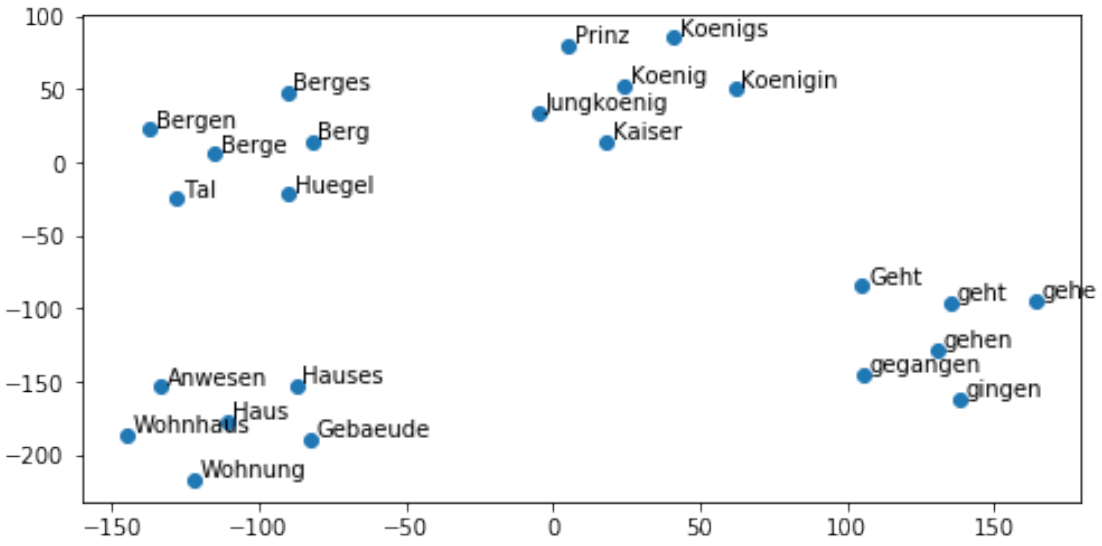
Weiterhin kann man auch ein Wort mit einer Liste von anderen Wörtern vergleichen und abfragen, welchem Wort aus der Liste das Wort am ähnlichsten ist oder man überprüft bei einer Liste von Wörtern, welches Wort nicht passt.

[20]:
```
most_similar = pre_w2v.most_similar_to_given('Banane', ['Koenig', 'Berg',
                                                        'Haus', 'Apfel'])
doesnt_match = pre_w2v.doesnt_match(['Koenig', 'Thron', 'Berg', 'Prinzessin'])
print(f"Welches Wort ist am ähnlichsten zu 'Banane'? -> {most_similar}")
print(f"Welches Wort nicht zu den anderen Wörtern? -> {doesnt_match}")
```

```
Welches Wort ist am ähnlichsten zu 'Banane'? -> Apfel
Welches Wort nicht zu den anderen Wörtern? -> Berg
```

Die Wortvektoren können mit verwandeten Wörtern (hier die 5 ähnlichsten Wörter) auch geplottet werden, wodurch die einzelnen Themen sehr gut sichtbar sind.

[12]:
```
wordlist = ['Haus', 'Berg', 'Koenig', 'gehen']
plot_word_embeddings(pre_w2v, wordlist, figsize=(8,4))
```



3.2.2 Eigene Embeddings trainieren

Es ist auch möglich, eigene Embeddings zu trainieren.

[ ]:
```
%%time
word2vec_cbow = Word2Vec(texts, min_count=1, size=100, window=5, sg=0)
word2vec_skipgram = Word2Vec(texts, min_count=1, size=100, window=5, sg=1)
```

[ ]:
```
word2vec.wv.most_similar('smartphone')
```

TODO: - Vorwissen? - Unterschied BOW: - dense -

[ ]:

[ ]:

[ ]:

3.3 Glove

TODO

3.4 FastText

TODO

3.5 ELMo

TODO

3.6 BERT

BERT steht für “Bidirectional Encoder Representations from Transformers” und brach bei seiner Veröffentlichung 2018 eine ganze Reihe von Rekorden für Aufgaben des Natural Language Processing.

[ ]: