Gensim

Topic modelling for humans

Einleitung

- Gensim ist eine Bibliothek für Python
  - semantische Analyse von Texten
  - suche nach Dokumenten mit semantischer Ähnlichkeit zur Anfrage

Philosophie

- Große Dokumenten Corpora sind ein Problem bezüglich ihrer Größe im Arbeitsspeicher
- Ziel: Gensim versucht alle Operationen möglichst resourcenschonend für den Rechner zu gestalten
- Große Datenstrukturen werden als Stream verwaltet
- Aufwendige Transformationen werden komprimiert abgespeichert
- vom der Datei bei Bedarf wieder gelesen

Gensim API

- Webseite
- Außerdem komplett über die Python help pages einsehbar

```
python
import gensim
help(gensim)                    # Biblotheksbeschreibung
help(gensim.corpora)               # Modulbeschreibung
help(gensim.corpora.Mmcorpus) # Klassenbeschreibung
...
```

Help(gensim)

```
DESCRIPTION
This package contains interfaces and functionality to
compute pair-wise document similarities within a
corpus of documents.

PACKAGE CONTENTS
    corpora (package)
    models (package)
    parsing (package)
    scripts (package)
    similarities (package)
    summarization (package)
    ...
```

Getting started

- Datei access

```
with open(filename, mode="r") as filestream:
    doc = filestream.readlines()
```

- Input: *Bag of words*

```
[[word_1, word_2, ..., word_n], # first document
 [word_1, word_2, ..., word_m]  # ...
 ...
 [word_1, word_2, ..., word_x]] # last document
```

- Gensim überlässt das File mapping dem Programmierer

```
filemapping = dict(enumerate(files))
{i_1 : filename_1,
 ...
 i_n : filename_n}
```

Dictionary

```
help(gensim.corpora)
help(gensim.corpora.Dictionary)
```

Output

```
class Dictionary(gensim.utils.SaveLoad,
                 collections.abc.Mapping)
|  Dictionary encapsulates the mapping between
|  normalized words and their integer ids.
|
|  The main function is `doc2bow`, which
|  converts a collection of words to its
|  bag-of-words representation: a list of
|  (word_id, word_frequency) 2-tuples.
|
|  Method resolution order:
|      Dictionary
|      gensim.utils.SaveLoad
|      ...
```

Corpus

- Der Corpus repräsentiert die Dokumente
- Einfaches Beispiel mit *doc2bow*:

```
corpus = []
for words in bag:
    corpus.append(my_dict.doc2bow(words))
```

- oder:

```
corpus = [my_dict.doc2bow(words) for words in bag]
```

- my_dict bildet die Wörter des Bags auf Zahlen ab

Corpus Fortgeschritten

```
class MyCorpus():
    self.path = ""
    def __init__(self, path):
        self.path = path
    def __iter__(self):
        with open(self.path) as file_stream:
            for doc in json.load(file_stream):
                yield my_dict.doc2bow(doc)
```

Gensim.corpora einige Beispiele

- `gensim.corpora.csvcorpus`
  - selbsterklärend
- `gensim.corpora.mmcorpus`
  - sparse matrix Format
  - jede nicht null Zelle wird mit Koordinaten abgespeichert
- `gensim.corpora.lowcorpus`
  - GibbsLda++ Format: Latent Dirichlet allocation
  - Dokumente als Wahrscheinlichkeitsverteilung der Topics
- `gensim.corpora.svmlightcorpus`
  - Format basierend auf Konzepten für Support Vector Machines
  - jede Zeile ist ein Trainingsdatensatz gefolgt von feature:value Paaren

Modelle

```
help(gensim.models)
help(gensim.models.tfidfmodel)
help(gensim.models.lsifmodel)
help(gensim.models.ldafmodel)
help(gensim.models.Word2Vec)
help(gensim.models.Doc2Vec)
```

Output

```
class TfidfModel(gensim.interfaces.TransformationABC)
 |  ...
 |
 |  The main methods are:
 |
 |  1. constructor, which calculates inverse document
 |     counts for all terms in the training corpus.
 |  2. the [] method, which transforms a simple count
 |     representation into the TfIdf space.
 |
 |  >>> tfidf = TfidfModel(corpus)
 |  >>> print(tfidf[some_doc])
 |  >>> tfidf.save('/tmp/foo.tfidf_model')
 |
 |  Model persistency is achieved via its load/save methods.
```

TFIDF

- Gensim: $w_{t,d} = tf(t,d) * log_2(\frac{N}{df(t)})$
  - tf und log_2 sind in Gensim austauschbar
- Folien: $w_{t,d} = (1 + log_{10}(tf(t,d)) * log(\frac{N}{df(t)})$
  - Bonus Aufgabe: Wer schafft es die Version der tfidf aus der Vorlesung in Gensim einzubauen?

Latent semantic Indexing

- akzeptiert *auch* einen durch tfidf gewichteten Corpus als Parameter
- erlaubt inkrementelles erweitern des Corpus um weitere Dokumente
- Unterstützt Corpora größer als RAM Beschränkungen es zulassen
- Streaming
- Distributed computing ohne große Anpassung des Codes

Latent Dirichlet Allocation

- Probabilistisches Modell
- kann inkrementell trainiert werden
- Cosinus Ähnlichkeit eher ungeeignet
  - besser: Hellinger Distanz
  - `gensim.mathutils.hellinger()`
  - Lässt sich nicht ohne weiteres in `Similarity` verwenden
- `sims = [(doc, hellinger(doc, query)) for doc in index]`

Anwendung

- Modelle sind Wrapper um den Corpus
- Das heißt, dass erst bei Anfrage das Modell *angewendet* wird

```
tfidf_corpus = models.TfidfModel(corpus)
```

- Die Models haben noch zusätzliche Parameter mit denen sich das Model weiter konfigurieren lässt

Suchanfragen

- `similarities`
- Berechnet die Cosinus Ähnlichkeit einer query zu einem Corpus
- Kann verschiedene `models` als Input bekommen
- Nicht Sinnvoll wenn Vektoren Wahrscheinlichkeitsverteilungen darstellen
- `Similarity`
  - erlaubt großen Index
  - wird in einzelne `shards` unterteilt für Speicher-Unabhängigkeit
- `SimilarityMatrix`
  - wird komplett in den Ram geladen

Anwendung

```
query_matrix =
    similarities.Similarity(model[corpus])
query = my_dict.doc2bow(["foo"])
query_model = model[query]
results = query_matrix[query_model]
```

Distributed Computing

- gensim lässt sich mit Pyro4 (python remote objects) auf ein Netzwerk aufteilen
- Es muss kein Code angepasst werden
- Man startet stattdessen Worker
- kommt mit wenig Netzwerk Kommunikation aus
- **Wichtig** Speedup kann allein schon durch Numpys Basic Linear Algebra library erreicht werden

Tips #1

- Dict key: value Paare

```
dict = {key : value}
dict[key] == value
```

- List (Typ egal)

```
list = [elem1, elem2, .., elem_n]
list[i] = elem_i
```

- Tuple (immutable)

```
touple = (elem1, elem2, ..., elem_n)
```

- Index an der Stelle i

```
enumerate(iterable) ==
    [(i_1, elem1), (i_2, elem2), ...,(i_n, elem_n)]
```

- Länge eines Iterables

```
len(iterable)
```

Tips #2

- String Funktionen
  - split(delimiter=" ") returnt liste getrennt an delimiter
  - lower() alles Kleinbuchstaben
- Sonderzeichen filtern

```
ignorechars = """[:.,;:!?"-()]\n"""
"bla!".replace("!", "")
```

- Sortieren

```
sorted(iterable[, cmp[, key[, reverse]]])
```

- Datei öffnen

```
with open(file[, mode[, encoding]]) as file_obj:
    file_obj.read() # Ganze Datei als String
    file_obj.readlines() # Liste von Zeilen
```

Vielen Dank!

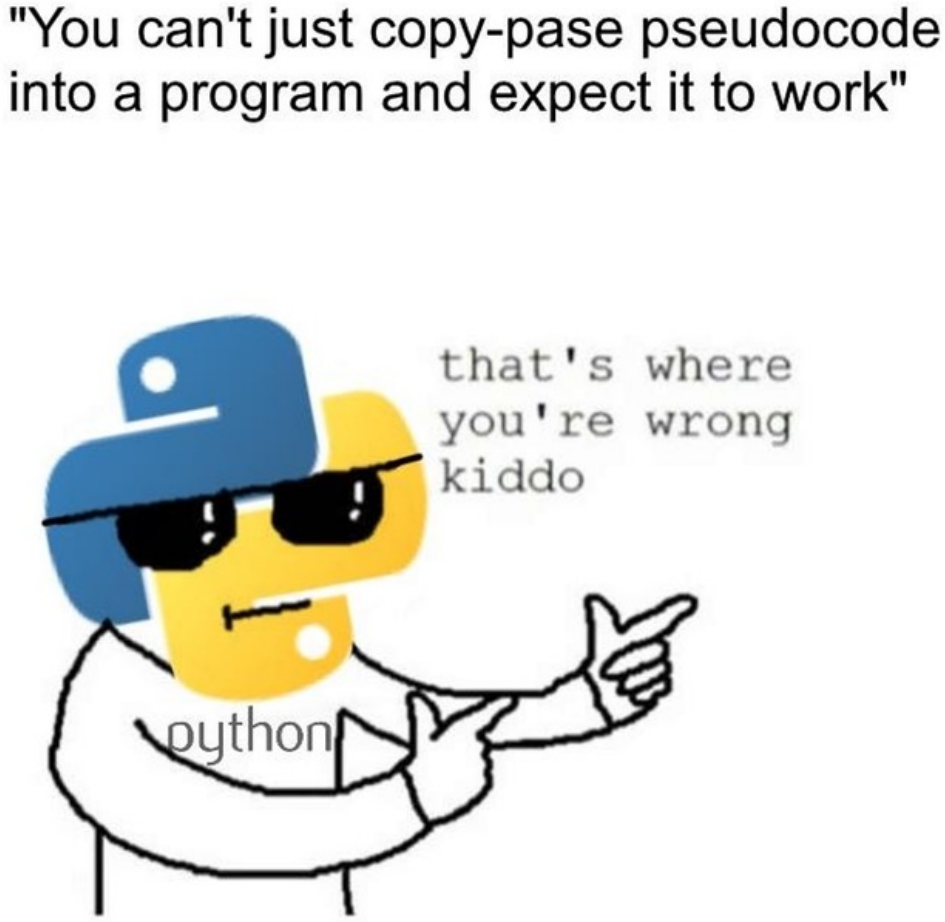

Figure 1: