Editor-Trick des Tages: Reveal active file in Explorer

Es ist praktisch, dass wir in VSCode ganze Ordner öffnen können und so bequem auf alle Dateien im Ordner zugreifen können. Manchmal braucht man aber den normalen Dateiexplorer, um z.B. Dateien an andere Orte zu verschieben oder andere Dateien zu öffnen, die im gleichen Ordner liegen, die aber nicht mit VSCode bearbeitet werden können.

Verwenden Sie den Shortcut ´Ctrl´ + `K` `R`, um ein Explorerfenster zu öffnen, in dem der Ordner der aktuellen Datei angezeigt wird. Wenn die Datei nicht gespeichert wurde, wird der Installationsordner von VSCode angezeigt.

Sprachdaten

Da die meisten von Ihnen Computerlinguistik studieren, spielen Sprachdaten in Ihrer weiteren Programmierlaufbahn eine wichtige Rolle. Wir haben schon in den letzten Wochen mit Sprachdaten gearbeitet:

- Wir haben Dateien geschrieben und gelesen.
- Wir haben Nutzereingaben verarbeitet.
- Wir haben mit regulären Ausdrücken komplexe Bedingungen für Sprachdaten geprüft.
- Wir haben Strings formatiert, um eigene Sprachausgaben zusammenzufügen.

Dabei haben wir uns typischerweise mit Einheiten in der Größenordnung einzelner Wörter beschäftigt.

N-Gramme

Es ist oft nützlich, Wörter nicht nur isoliert, sondern auch in ihren Kontexten innerhalb eines Textes oder einer Textsammlung zu betrachten. Beispielsweise kann es informativ sein, Einheiten aus je zwei **benachbarten** Wörtern zu analysieren. Solche Einheiten heißen **Bigramme**. Einheiten aus drei Wörtern bezeichnen wir als **Trigramme**; Einheiten aus jeweils nur einem Wort heißen **Unigramme**. Der allgemeine Begriff lautet **N-Gramm**. Es gibt auch Zeichen-N-Gramme, also Einheiten aus einem/zwei/drei/… Zeichen.

Anhand von Unigrammen können wir bereits einige Aussagen über einen Text machen: Was sind wichtige Themen (erkennbar an häufigen Unigrammen)? Kommen im Text Wörter vor, die in anderen Texten nicht vorkommen (***Hapaxe***)?

Aber Unigramme verlieren jeglichen Kontext und sind deshalb manchmal nicht zu gebrauchen. Hier ein Beispiel:

In [1]:

```
text1 = "Ich finde Katzen gut, aber Hunde nicht."
text2 = "Ich finde Hunde gut, aber Katzen nicht."

unigrams1 = set(text1.split())
unigrams2 = set(text2.split())

print(unigrams1)
print(unigrams2)
```

In natürlichen Sprachen interagieren Wörter miteinander, wenn sie in einem syntaktischen Zusammenhang stehen. Das Wort "nicht" in den beiden Texten im Beispiel hatte eine modifizierende Bedeutung, die sich jeweils nur auf Katzen oder nur auf Hunde bezog. Dass wir wissen, wie oft "nicht" in jedem der Texte vorkam, hilft uns also nicht dabei, herauszufinden, wie die Texte sich unterscheiden.

Um ein vollständiges Abbild des Textes zu erstellen, wäre es natürlich am informativsten, jeweils den gesamten Text (sozusagen als "X-Gramm") zu lesen… Aber dann können wir keine abgestuften Aussagen über Ähnlichkeit oder Unterschied zwischen den Texten machen. Mit Bigrammen erhalten wir folgendes Bild:

In [1]:

```
bigrams1 = {"Ich finde", "finde Katzen", "Katzen gut", "gut ,", ", aber", "aber Hunde",
"Hunde nicht", "nicht ."}
bigrams2 = {"Ich finde", "finde Hunde", "Hunde gut", "gut ,", ", aber", "aber Katzen",
"Katzen nicht", "nicht ."}

print(bigrams1.intersection(bigrams2))
print(bigrams1.difference(bigrams2))
```

Je größer die N-Gramme werden, desto mehr schrumpft die Anzahl der gemeinsamen N-Gramme zwischen den Texten. Hier die Trigramme:

In [1]:

```
trigrams1= {"Ich finde Katzen", "finde Katzen gut", "Katzen gut ,", "gut , aber", ", ab
er Hunde", "aber Hunde nicht", "Hunde nicht ."}
trigrams2= {"Ich finde Hunde", "finde Hunde gut", "Hunde gut ,", "gut , aber", ", aber
 Katzen", "aber Katzen nicht", "Katzen nicht ."}

print(trigrams1.intersection(trigrams2))
print(trigrams1.difference(trigrams2))
```

Warum sind N-Gramme hilfreich?

Texte in N-Gramme zu zerlegen, hilft uns also dabei, Ähnlichkeiten und Unterschiede zwischen Text A und Text B zu finden. Zusätzlich helfen uns N-Gramme, wenn wir in einem Satz wie dem folgenden vorhersagen wollen, welches Wort als nächstes kommt (Beispiel aus "Speech and Language Technology" von Jurafsky & Martin):

> Please turn your homework …

Wir könnten die Grammatik des Satzes genauestens analysieren, Wörterbucheinträge für jedes Wort konsultieren und dann schlussfolgern, dass nur eine Präposition als nächstes folgen kann, und zwar genauer: die Präposition "in". Das würde aber sehr aufwendige Analysen erfordern für eine Aufgabe, die wir als Menschen intuitiv lösen können, und für die Maschinen auch nur ein bisschen Statistik kennen müssen.

In den Worten von Jurafsky und Martin (https://web.stanford.edu/~jurafsky/slp3/3.pdf):

> Hopefully, most of you concluded that a very likely word is in, or possibly over, but probably not refrigerator or the. (…) We will formalize this intuition by introducing models that assign a probability to each possible next word. The same models will also serve to assign a probability to an entire sentence. Such a model, for example, could predict that the following sequence has a much higher probability of appearing in a text:
>
> > *all of a sudden I notice three guys standing on the sidewalk*
>
> than does this same set of words in a different order:
>
> > *on guys all I of notice sidewalk three a sudden standing the*

Die Einsatzmöglichkeiten von N-Grammen sind vielseitig: Bei Jurafsky & Martin werden Spracherkennung, Handschrifterkennung, Rechtschreibkorrektur und automatische Übersetzung als Beispiele genannt. Allgemein sind N-Gramme eine wichtige Methode zum Modellieren der Verteilung von Wörtern oder Zeichen.

Wenn in einem deutschsprachigen Text die Zeichenfolge shc vorkommt, können wir mit hoher Sicherheit vermuten, dass dort eigentlich sch stehen sollte.

Die Wahrscheinlichkeiten können auch zur Einordnung eines Textes in bestimmte Kategorien verwendet werden. Beispielsweise unterscheiden sich verschiedene Sprachen sehr stark in den Buchstabenkombinationen, die jeweils möglich sind: Texte, in denen she häufiger ist als sch, sind wahrscheinlich eher auf Englisch verfasst.

Das NLTK:

Was ist das?

Das *Natural Language Tool Kit* ist ein großes Modul für Python, dass speziell dafür entwickelt wurde um Python-Programme zur Verarbeitung von natürlicher Sprache zu schreiben. Es besteht aus mehreren Teilmodulen und enthält u.a. große Textsammlungen, sowie spezialisierte Funktionen zu syntaktischen, semantischen und morphologischen Analyse von Sprachdaten.

Zusätzlich zur Dokumentation ist für das NLTK auch ein Buch frei online verfügbar, das Einblicke in den Funktionsumfang des NLTK bietet und gleichzeitig auch versucht grundlegendes Wissen über Python zu vermitteln (https://www.nltk.org/book/ (https://www.nltk.org/book/)).

Auf Grund des großen Umfangs, und der Art der enthaltenen Funktionalitäten ist das NLTK nicht Teil der normalen Python-Distribution, kann aber nachträglich über die Kommandozeile installiert werden (http://www.nltk.org/install.html (http://www.nltk.org/install.html)). Starten Sie dafür die Kommandozeile mit Administratorrechten:

1. Suchen Sie im Startmenü nach "cmd"
2. Starten Sie Kommandozeile mit der Tastenkombionation `STRG + SHIFT + ENTER`
3. In der Kommandozeile rufen sie den folgenden Befehl auf: `pip install nltk`

Bevor wir beginnen können mit den Funktionen aus dem NLTK-Modul zu arbeiten müssen wir es importieren, wie wir es bei anderen Modulen auch tun.

Zusätzlich, weil das Modul so groß ist und sich aus mehreren Teilmodulen zusammensetzt müssen wir uns überlegen, welche Teilpakete wir tatsächlich benutzen und evtl. noch herunterladen müssen.

In [1]:

```
import nltk
```

Auf Texte aus dem Internet zugreifen

Das NLTK bringt eine Auswahl an Texten mit, mit denen wir arbeiten können. Diese werden aber nicht automatisch bei der Installation des NLTK auf ihrem Rechner gespeichert. Auch um Texte herunterzuladen können Sie den eingebauten Download-Manager des NLTK nutzen:

In [1]:

```
import nltk
nltk.download()  # Startet den eingebauten Download-Manager
# In der GUI können sie die "Book" collection auswählen und herunterladen.
from nltk.book import *

whale_book_1 = text1[:100]
print(whale_book_1)
```

Auch im Internet findet sich gutes Rohmaterial zur Verarbeitung mit Hilfe von Python. Viele ältere Texte ohne Copyright finden Sie, wenn Sie nach dem Titel des Texts + "plain text" suchen:

`"Alice in Wonderland" plain text` ergibt: http://www.umich.edu/~umfandsf/other/ebooks/alice30.txt (http://www.umich.edu/~umfandsf/other/ebooks/alice30.txt)

Sie können den Text entweder mit über den Browser abspeichern oder die URL benutzen um in Python mit Hilfe des Requests-Moduls auf den Text zuzugreifen:

In [1]:

```
import requests
text_from_url = requests.get("http://www.umich.edu/~umfandsf/other/ebooks/alice30.txt")

for line in text_from_url:
  print(line)
```

Type/Token Unterscheidung & Tokenization

Types & Tokens

Wenn wir Texte maschinell auswerten, ist oft die Rede von Tokens und Types. Dabei bezeichnet **Token** eine beliebige Gruppe von Zeichen , die durch Leerzeichen von anderen Gruppen abgegrenzt ist, und auch Interpunktion. Tokens eines Texts sind nicht immer gleichbedeutend mit den Wörtern in einem Text. Beispielsweise werden nicht alle zusammengesetzten Substantive im Englischen auch zusammengeschrieben. Auch möchten wir nicht immer Interpunktionszeichen als Wörter zählen.

Für den Anfang könnte man versuchen die Tokens eines Texts einfach dadurch bestimmen, indem man den Text an den Leerzeichen aufsplittet.

In [1]:

```
my_text1 = """This is a book about Natural Language Processing. By "natural language" w
e mean a language that is used for everyday communication by humans; languages like Eng
lish, Hindi or Portuguese. In contrast to artificial languages such as programming lang
uages and mathematical notations, natural languages have evolved as they pass from gene
ration to generation, and are hard to pin down with explicit rules. We will take Natura
l Language Processing — or NLP for short — in a wide sense to cover any kind of compute
r manipulation of natural language. At one extreme, it could be as simple as counting w
ord frequencies to compare different writing styles. At the other extreme, NLP involves
"understanding" complete human utterances, at least to the extent of being able to give
useful responses to them."""

tokens_v1 = my_text1.lower().split()
print(sorted(tokens_v1))
```

Mit **Type** bezeichnet man ein Element aus der Menge der Zeichengruppen aus denen eine Text besteht. Die Menge der Types ist nicht gleichbedeutend mit der Menge der Vokabeln in einem Text, da beispielsweise jede Flektionsform eines Worts einem Type entspricht. Basierend auf der Liste der Tokens, können wir uns mit Hilfe der Set-Funktion grob an die Tokenmenge annähern.

In [1]:

```
my_text1 = """This is a book about Natural Language Processing. By "natural language" w
e mean a language that is used for everyday communication by humans; languages like Eng
lish, Hindi or Portuguese. In contrast to artificial languages such as programming lang
uages and mathematical notations, natural languages have evolved as they pass from gene
ration to generation, and are hard to pin down with explicit rules. We will take Natura
l Language Processing — or NLP for short — in a wide sense to cover any kind of compute
r manipulation of natural language. At one extreme, it could be as simple as counting w
ord frequencies to compare different writing styles. At the other extreme, NLP involves
"understanding" complete human utterances, at least to the extent of being able to give
useful responses to them."""

tokens_v1 = my_text1.lower().split()

types_v1 = set(tokens_v1)
print(types_v1)
```

Tokenization

Mit **Tokenization** wird die Generierung von Token-Listen eines Texts bezeichnet. Wie wir oben gesehen haben, ist diese Aufgabe schwieriger als zunächst gedacht. Jedoch ist ein guter Algorithmus zu Tokenisierung notwendig für die weitere Analyse, wie schon die Erstellung der Typenmenge zeigt. Wir haben bereits einige Werkzeuge kennengelernt, mit denen wir bessere Tokenlisten generieren können:

Tokenization mit String-Methoden

In [1]:

```
my_text1 = """This is a book about Natural Language Processing. By "natural language" w
e mean a language that is used for everyday communication by humans; languages like Eng
lish, Hindi or Portuguese. In contrast to artificial languages such as programming lang
uages and mathematical notations, natural languages have evolved as they pass from gene
ration to generation, and are hard to pin down with explicit rules. We will take Natura
l Language Processing — or NLP for short — in a wide sense to cover any kind of compute
r manipulation of natural language. At one extreme, it could be as simple as counting w
ord frequencies to compare different writing styles. At the other extreme, NLP involves
"understanding" complete human utterances, at least to the extent of being able to give
useful responses to them."""

tokens_v2 = []
interpunktion = (",", ";", ":", ".", "?", "!")

for c in my_text1:
  if c in interpunktion:
    tokens_v2.append(c)

for c in interpunktion:
  my_text1 = my_text1.replace(c, "")

tokens_v2 = sorted(tokens_v2 + my_text1.lower().split())
print(tokens_v2)
print(len(tokens_v2))

print("-----")

types_v2 = set(tokens_v2)
print(types_v2)
print(len(types_v2))
```

Tokenization mit Regulären Ausdrücken

In [1]:

```
import re
token_muster = re.compile("[a-z]+|\.|;|,|[a-z]+-[a-z]+|—")

tokens_v3 = re.findall(token_muster, my_text1.lower())
print(tokens_v3)
print(len(tokens_v3))

print("-----")

types_v3 = set(tokens_v3)
print(len(types_v3))
print(types_v3)

print("-----")

missed_types = types_v2 - types_v3
print(missed_types)
```

Das NLTK beinhaltet ein eigenes Teilmodul zur Tokenization. Dieses enthält mehrere Tokenizer zum Beispiel zur Erkennung von Satzgrenzen und weitere Tokenizer zur "Worterkennung" für verschiedene Sprachen. Diese basieren auf fortgeschritteneren Analyse-Methoden.

Stoppwörter

Wenn wir uns mit den Inhalt von Texten maschinell analysieren wollen, ist es hilfreich Wörter zu ignorieren, die vorraussichtlich in jedem Text vorhanden sind (z.B. Pronomen, Hilfsverben, Präpositionen oder Interpunktion). Dieses lexikalische Material nennt man **Stoppwörter**.

Wir können für unsere Zwecke selbst Sequenzen von Stoppwörtern definieren. Das NLTK hält aber auch für verschiedene Sprachen Listen von Stoppwörtern bereit.

In [1]:

```
import nltk
from nltk.book import *
from nltk.corpus import stopwords as SW  #

print("\n-------------------------------------\n")

sw_german = SW.words("german")
sw_english = SW.words("english")
sw_italian = SW.words("italian")
print(sw_german)
print(sw_english)
print(sw_italian)
```

Beachten Sie, dass unter Umständen homomorphe lexikalische Elemente auch mit ausgefiltert werden können (z.B. Eigennamen: "Weil" oder Nomen: "Weg").

N-Gramme mit dem NLTK

Das NLTK hilft Ihnen auch N-Gramme aus einem Text zu generieren. Mit Hilfe der der Bigrams-Funktion können Sie aus einer Sequenz Bigramme generieren. Die Everygrams-Funktion ermöglicht es beliebige N-Gramme zu generieren.

In [1]:

```
import nltk
import re
from nltk.util import bigrams
from nltk.util import everygrams

my_text1 = """This is a book about Natural Language Processing. By "natural language" w
e mean a language that is used for everyday communication by humans; languages like Eng
lish, Hindi or Portuguese. In contrast to artificial languages such as programming lang
uages and mathematical notations, natural languages have evolved as they pass from gene
ration to generation, and are hard to pin down with explicit rules. We will take Natura
l Language Processing — or NLP for short — in a wide sense to cover any kind of compute
r manipulation of natural language. At one extreme, it could be as simple as counting w
ord frequencies to compare different writing styles. At the other extreme, NLP involves
"understanding" complete human utterances, at least to the extent of being able to give
useful responses to them."""

token_muster = re.compile("[a-z]+|\.|;|,|[a-z]+-[a-z]+|—")

tokens_v3 = re.findall(token_muster, my_text1.lower())

my_bigrams = bigrams(tokens_v3)
print(list(my_bigrams))

print("-----")

my_trigrams = everygrams(tokens_v3, min_len=3,max_len=3)
print(list(my_trigrams))
```

Aufgabe

Nutzen Sie Tokenlisten und Stoppwörter um die Bigramm- und Trigrammlisten sinnvoll zu reduzieren.

Sie haben heute gelernt…

- was N-Gramme sind.
- wie Sie direkt auf Texte aus dem Internet zugreifen können.
- was der Unterschied zwischen Types und Tokens ist.
- was Stoppwörter sind.
- wie man mit dem NLTK N-Gramme generieren kann.