Technische Hochschule Rosenheim

Modul - Introduction to AI - part II (AI2)

Bachelor Programme AAI

10 - Applications of NLP

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Agenda

Technische Hochschule Rosenheim

On the menu for today:

- Similarity and Similarity Search
  - Term frequency and others, e.g. TF-IDF
  - Cosine Similarity
- Language Models

Recap NLP Taxonomy

PreProcessing

Technische Hochschule Rosenheim

- The first thing you need to do in any NLP project is text preprocessing.
- Preprocessing input text simply means putting the data into a predictable and analyzable form.
- It's a crucial step for building an NLP application.
- There are different ways to preprocess text:
  - tokenization
  - stemming
  - lemmatization
  - stop word removal

Tokenization

Technische Hochschule Rosenheim

What is tokenization?

Tokenization

Technische Hochschule Rosenheim

What is tokenization?

- Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*.
- *Tokenization* can separate sentences, words, characters, or subwords.
- Forms of tokenization:
  - White Space Tokenization or by separator

```
sentence = "I was born in Germany in 2000. Thus, I grew up in Germany."
sentence.split()
sentence.split(',')
```

  - NLTK tokenization (`import nltk`)
  - word, sentence, punctuation, TreeBank, Tweet, MWET TextBlob, spaCy, Gensim, Keras

Stemmming

Technische Hochschule Rosenheim

What is stemmming?

Stemmming

Technische Hochschule Rosenheim

What is stemmming?

- For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing.
- Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization.
- The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

```
am, are, is => be
car, cars, car's, cars' => car
```

- The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is *Porter's algorithm*.

(F)

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Lemmatization

Technische Hochschule Rosenheim

What is lemmatization?

Lemmatization

Technische Hochschule Rosenheim

What is lemmatization?

- Rather than using a stemmer, you can use a *lemmatizer*!
- Lemmatization is a text normalization technique used in Natural Language Processing (NLP). It has been studied for a very long time and lemmatization algorithms have been made since the 1960s.
- Essentially, lemmatization is a technique that switches any kind of a word to its base root mode.
- A lemmatization algorithm would know that the word *better* is derived from the word *good*, and hence, the lemme is *good*.

Stop word removal

Technische Hochschule Rosenheim

What is Stop word removal?

Stop word removal

Technische Hochschule Rosenheim

What is Stop word removal?

- *Stop word removal* is removing the words that occur commonly across all the documents in the corpus.
- Typically, articles and pronouns are generally classified as *stop words*.
  - Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.
- These words have no significance in some of the NLP tasks like information retrieval and classification, which means these words are not very discriminative.

```
import nltk
from nltk.corpus import stopwords
sw_nltk = stopwords.words('english')
print(sw_nltk)
# to be continued..
```

Stopwords in practice

Technische Hochschule Rosenheim

```
text = "When I first met her she was very quiet. She remained quiet during the
new_text = " ".join(words)

print("Old length: ", len(text), " - ", text)
print("New length: ", len(new_text), " - ", new_text)
```

with spaCY:

```
import spacy
#loading the english language small model of spacy
en = spacy.load('en_core_web_sm')
sw_spacy = en.Defaults.stop_words
print(sw_spacy)
words = [word for word in text.split() if word.lower() not in sw_spacy]
new_text = " ".join(words)
print("Old length: ", len(text), " - ", text)
print("New length: ", len(new_text), " - ", new_text)
```

Application of NLP

Term-Frequency (TF)

Technische Hochschule Rosenheim

- A common approach when defining a similarity measure is to map the objects into a Euclidean vector space, because there are many ways to calculate distances or similarities (e.g. the Euclidean distance).
- Example:
  - A simple option to generate vectors from wine reviews is to arrange the terms that occur in any wine review (our vocabulary) in any order, to count the number of times each term occurs in each review, and to use this number as the value of the corresponding dimension. This is a good first step, since the count of a term (word count, or term frequency, TF) provides information on how often the term occurs in the review, i.e. presumably how important it is for the content of this review.

Example: TF

Technische Hochschule Rosenheim

Term Frequencies					
	w1	w2	w3	w4	w5
leather	1	1	0	2	3
party	1	1	1	1	1
lovely	2	0	1	2	0
wine	1	2	1	2	3
bit	1	1	0	0	0
good	1	0	1	0	0
delicate	1	1	0	1	1
drinking	0	0	0	2	2
port	0	4	0	2	3
Total	8	10	4	12	13

TF in Python

Technische Hochschule Rosenheim

```
# lets write it down as an array

w=[
    [1,1,2,1,1,1,1,0,0],
    [1,1,0,2,1,0,1,0,4],
    [0,1,1,1,0,1,0,0,0],
    [2,1,2,2,0,0,1,2,2],
    [3,1,0,3,0,0,1,2,3]
]

total = [sum(x) for x in w]

print(total)
```

Normalized TF

- Another aspect that we want to take into account is that wine reviews are of different lengths.
- The TFs (number of occurrences of a term) are therefore higher for a long review than for a short review, even if exactly the same terms occur with the same relative frequency in the two reviews.
- This is unattractive, but very easy to solve: we normalize the vector of each review (e.g. so that the TFs contained therein add up to 1.
- This results in the *normalized term frequency*.

Normalized TFs

	w1	w2	w3	w4	w5
leather	0,13	0,10	0,00	0,17	0,23
party	0,13	0,10	0,25	0,08	0,08
lovely	0,25	0,00	0,25	0,17	0,00
wine	0,13	0,20	0,25	0,17	0,23
bit	0,13	0,10	0,00	0,00	0,00
good	0,13	0,00	0,25	0,00	0,00
delicate	0,13	0,10	0,00	0,08	0,08
drinking	0,00	0,00	0,00	0,17	0,15
port	0,00	0,40	0,00	0,17	0,23
Total	1	1	1	1	1

Inverse Document Frequency (IDF)

Technische Hochschule Rosenheim

- We also observe that some terms appear in a large number of reviews (e.g. the term "wine" will appear in almost every review since it is not included in our stop word list).
- This provides very little help in distinguishing the reviews, while other terms appear in only a few reviews, so are probably a strong indication of the similarity of these reviews and the difference to all other reviews.

- We define the *inverse document frequency* (IDF) of a term as log (N / n), where N is the number of all reviews, n the number of reviews in which the term occurs. Terms that appear in almost all reviews have an IDF of almost 0. The fewer reviews the terms is contained in, the higher its IDF value.

Example: IDF

Technische Hochschule Rosenheim

Normalized TFs								
	w1	w2	w3	w4	w5		# wines	IDF
leather	0,13	0,10	0,00	0,17	0,23		4	0,22
party	0,13	0,10	0,25	0,08	0,08		5	0,00
lovely	0,25	0,00	0,25	0,17	0,00		3	0,51
wine	0,13	0,20	0,25	0,17	0,23		5	0,00
bit	0,13	0,10	0,00	0,00	0,00		2	0,92
good	0,13	0,00	0,25	0,00	0,00		2	0,92
delicate	0,13	0,10	0,00	0,08	0,08		4	0,22
drinking	0,00	0,00	0,00	0,17	0,15		2	0,92
port	0,00	0,40	0,00	0,17	0,23		3	0,51
Total	1	1	1	1	1			

nTF and IDF in Python

Technische Hochschule Rosenheim

```
import numpy as np


w_t = np.asarray(w).transpose()

wines = [ np.count_nonzero(x) for x in w_t]
IDF = (-1)*np.log(np.divide(np.asarray(wines),5))

print("wines: " + str(wines))
print("IDF  : " + str(IDF))

w_tf = np.divide(w_t, total)

total2 = [sum(x) for x in w_tf.transpose()]

print("TF    :\n "+ str(w_tf))
print("Total : " + str(total2))
```

TF-IDF

Technische Hochschule Rosenheim

We generate the final TF-IDF vector for each review by multiplying the (normalized) TF value of the review by the IDF value of the term, i.e. weighting the term with its distinctiveness.

TF-IDF					
	w1	w2	w3	w4	w5
leather	0,028	0,022	0,000	0,037	0,051
party	0,000	0,000	0,000	0,000	0,000
lovely	0,128	0,000	0,128	0,085	0,000
wine	0,000	0,000	0,000	0,000	0,000
bit	0,115	0,092	0,000	0,000	0,000
good	0,115	0,000	0,229	0,000	0,000
delicate	0,028	0,022	0,000	0,019	0,017
drinking	0,000	0,000	0,000	0,153	0,141
port	0,000	0,204	0,000	0,085	0,118

in Python:

```
tf_idf = np.multiply(w_tf.transpose(),IDF).transpose()

print("TF-IDF : \n" + str(tf_idf))
```

Similarity

Technische Hochschule Rosenheim

- We could could simply use the Euclidean distance of the TF-IDF vectors as a measure of the diversity of the reviews.
- However, this has two drawbacks:
  - on the one hand, it is a distance and not a similarity.
  - on the other hand, the TF-IDF vectors are very long, i.e. the vector space is very high dimensional: there are as many dimensions as there are different terms in all reviews together.

> The Euclidean distance suffers from the "curse of dimensionality", i.e. the higher dimensional the vector space becomes, the less meaningful it is (the distances become more and more similar).

Cosine-Similarity

Technische Hochschule Rosenheim

- We can solve both problems elegantly by using the *cosine similarity* instead of the Euclidean distance.
- If we imagine the TF-IDF vectors as "arrows" from the origin, the direction of the arrow describes the content of the review. The angle between two such "review arrows" describes how similar the two reviews are - if the angle is 0 degrees or close to 0, if the reviews go in the same direction, if they are 90 degrees or close, the reviews are as different as possible.
- If we apply the cosine function to this angle, we get a number between 1 = cos(0 degrees) for maximum similarity and 0 = cos(90 degrees) for maximum dis-similarity.
- The cosine of two vectors of an Euclidean vector space is the dot product of the two divided by the product of the norms.

Cosine-Similarity as formula:

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||}$$

Revisit the example

Technische Hochschule Rosenheim

Cosine-Similarity

	w1	w2	w3	w4	w5
w1	1	0,25	0,77	0,30	0,05
w2	0,25	1	0,00	0,41	0,59
w3	0,77	0,00	1	0,21	0,00
w4	0,30	0,41	0,21	1	0,89
w5	0,05	0,59	0,00	0,89	1

in Python:

```
from numpy.linalg import norm

cos_sim = []
tf_idf_t = tf_idf.transpose()
for x in tf_idf_t:
  for y in tf_idf_t:
    cos_sim.append( np.dot(x, y)/(norm(x)*norm(y)) )
cos_sim = np.asarray(cos_sim).reshape(5,5)
print(cos_sim)
```

Some more code

Technische Hochschule Rosenheim

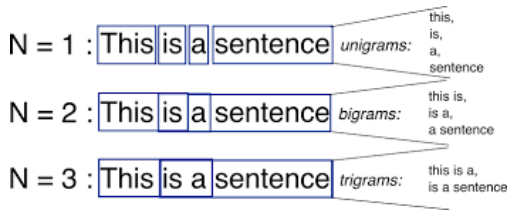
Open the following Jupyter Notebook: ./nlp_part1.ipynb

Language Model

N-Grams Revisited

Technische Hochschule Rosenheim

- Text n-grams are commonly utilized in natural language processing and text mining.
- It's essentially a string of words that appear in the same window at the same time.



- While creating language models, n-grams are utilized not only to create unigram models but also bigrams and trigrams.

```
sentence = "I really like python, it's pretty awesome.".split()
N = 3
grams = [sentence[i:i+N] for i in range(len(sentence)-N+1)]

for gram in grams:
    print(gram)
```

Some more code

Technische Hochschule Rosenheim

Open the following Jupyter Notebook: ./nlp_part2.ipynb

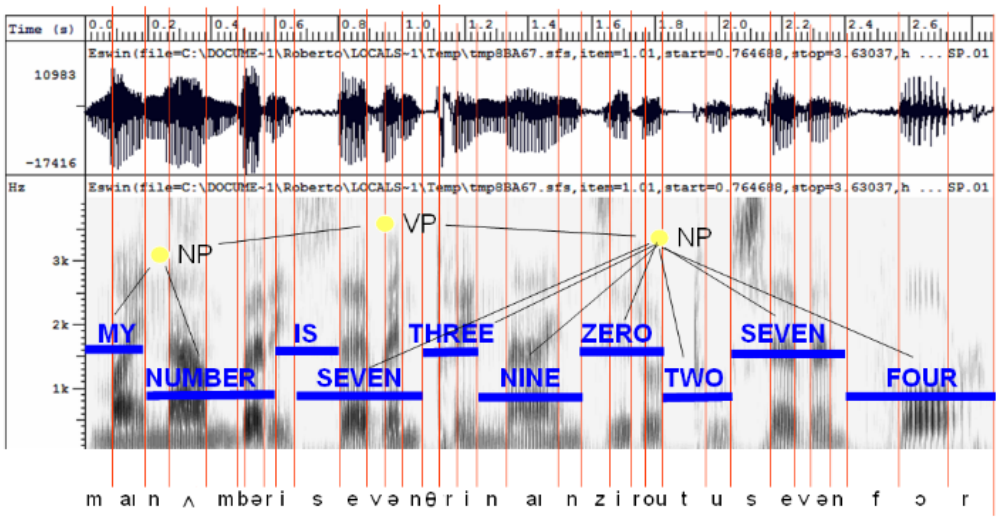
Speech Recognition in a Nutshell

Speech Recognition in short

Technische Hochschule Rosenheim

Audio -> Spectrogram -> Phonemes -> Words> Sets (fragmente)

Accustic Model



Language Model

```
The code that was used to produce this
Please visit http://www.speech.cs.cmu.

The (fixed) discount mass is 0.5. The
This model based on a corpus of 11 sen

\data\
ngram 1=69
ngram 2=92
ngram 3=89

\1-grams:
-1.3127 </s> -0.3010
-1.3127 <s> -0.2671
-1.7520 A -0.2933
-2.0531 AND -0.2972
-2.3541 ANGINA -0.2991
-2.3541 ANTIGEN -0.2952
-2.3541 ARE -0.2991
-2.3541 ARTERY -0.2794
-2.3541 ASYMPTOMATIC -0.2991
-2.3541 BARR -0.2991
-2.3541 BILATERALLY -0.2794
-2.3541 BOTH -0.2991
-2.3541 BRUIT -0.2991
```

Summary

Technische Hochschule Rosenheim

Lessons learned today:

- NLP taxonomy recap
- Similarity Matching
- Language Models
- Speech recognition