Natural Language Processing Workshop

Computer Engineering Department
Sharif University of Technology

Spring 2023

Outline

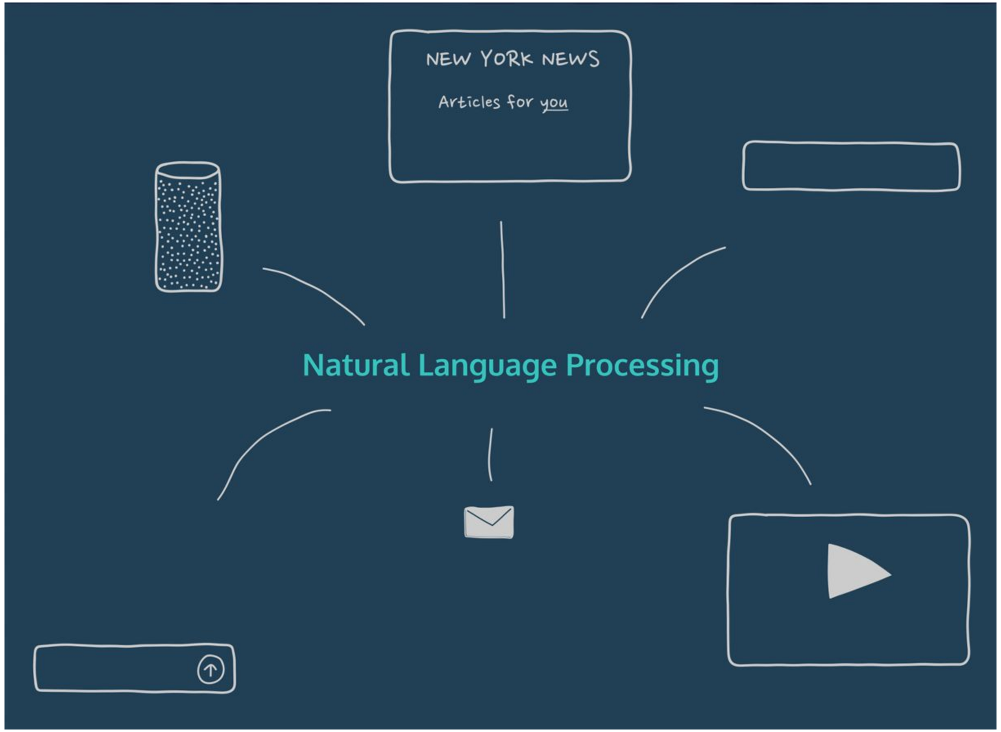
Intro to NLP

NLP in Real World

NLP
Analyzing, understanding, and generating the languages that humans use naturally

Applications of NLP

- Machine translation(Google Translate)
- Natural language generation
- Web Search
- Spam filters
- Sentiment Analysis
- Chatbots



Data Preprocessing

Data Preprocessing

What?

- Remove special characters, symbols, punctuation, HTML tags<>, etc.
- Depends on the problem statement
- Better to do **lowercase** before starting it

Why?

- Remove text that contains no information for the model to learn
- simply noise in our data

```
# Preprocessing

import re

#sample review from the IMDB dataset.
review = "<b>A touching movie!!</b> It is full of emotions and wonderful acting.<br> I could have sat
through it a second time."

cleaned_review = re.sub(re.compile('<.*?>'), '', review) #removing HTML tags
cleaned_review = re.sub('[^A-Za-z0-9]+', ' ', cleaned_review) #taking only words

print(cleaned_review)
> A touching movie It is full of emotions and wonderful acting I could have sat through it a second time
```

Stop words Removal

What?

- Stop words are common words that do not contribute much of the information in a text document.

Why?

- Words like ‘the’, ‘is’, ‘a’ have less value and add noise to the text data.

How?

- There is an in-built stopword list in NLTK which we can use to remove stop words from text documents. However this is not the standard stopwords list for every problem, we can also define our own set of stopwords based on the domain.

Tokenization

What?

- Break up text document into individual words called **tokens**

Why?

- Split paragraphs and sentences into smaller units that can be more easily assigned meaning

```
# Tokenization

# import nltk
# nltk.download('punkt')

from nltk.tokenize import word_tokenize
tokens = nltk.word_tokenize(cleaned_review)
print(tokens)
> ['a', 'touching', 'movie', 'it', 'is', 'full', 'of', 'emotions', 'and', 'wonderful', 'acting', 'i',
'could', 'have', 'sat', 'through', 'it', 'a', 'second', 'time']
```

Stemming

What?

- Reduce a word to its stem/root word

Why?

- Give the same information for words which differs in morphological affixes to the model to learn.

```
# Stemming

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
stemmed_review = [stemmer.stem(word) for word in filtered_review]

print(stemmed_review)
> ['touch', 'movi', 'full', 'emot', 'wonder', 'act', 'could', 'sat', 'second', 'time']
```

Lemmatization

What?

- same thing as stemming, converting a word to its root form but with one difference i.e., the root word, in this case, belongs to a valid word in the language.
- Example: the word caring would map to ‘care’ and not ‘car’ in case of stemming.

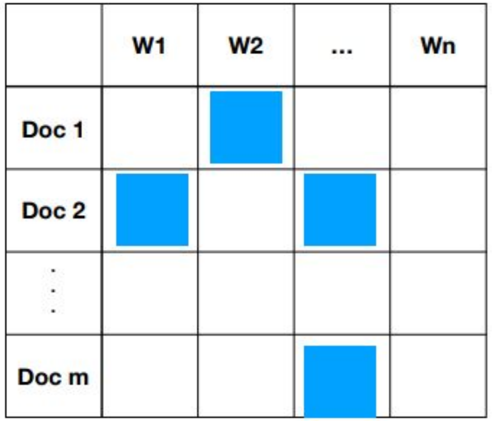
```
# Lemmatization
# import nltk
# nltk.download('wordnet')
# nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemm_review = [lemmatizer.lemmatize(word) for word in filtered_review]
print(lemm_review)
> ['touching', 'movie', 'full', 'emotion', 'wonderful', 'acting', 'could', 'sat', 'second', 'time']
```

Text Data Vectorization

Text Data Vectorization

What?

- The process of converting text into numbers.



	W1	W2	...	Wn
Doc 1		🟦		
Doc 2	🟦		🟦	
⋮				
Doc m			🟦	

Why?

- Now after text preprocessing, we need to numerically represent text data i.e., encoding the data in numbers which can be further used by algorithms.

Example

- Bag of Words (BoW)
- **CountVectorizer**
- **TF-IDF**

CountVectorizer

What?

- Convert a collection of text documents to a matrix of token counts.
- Each doc will be represented with an encoding vector with the length of the entire vocabulary.
- Each vector element is an integer count for the number of times each word occurs in the doc.
- You can use `sklearn.feature_extraction.text.CountVectorizer`

TF-IDF

What?

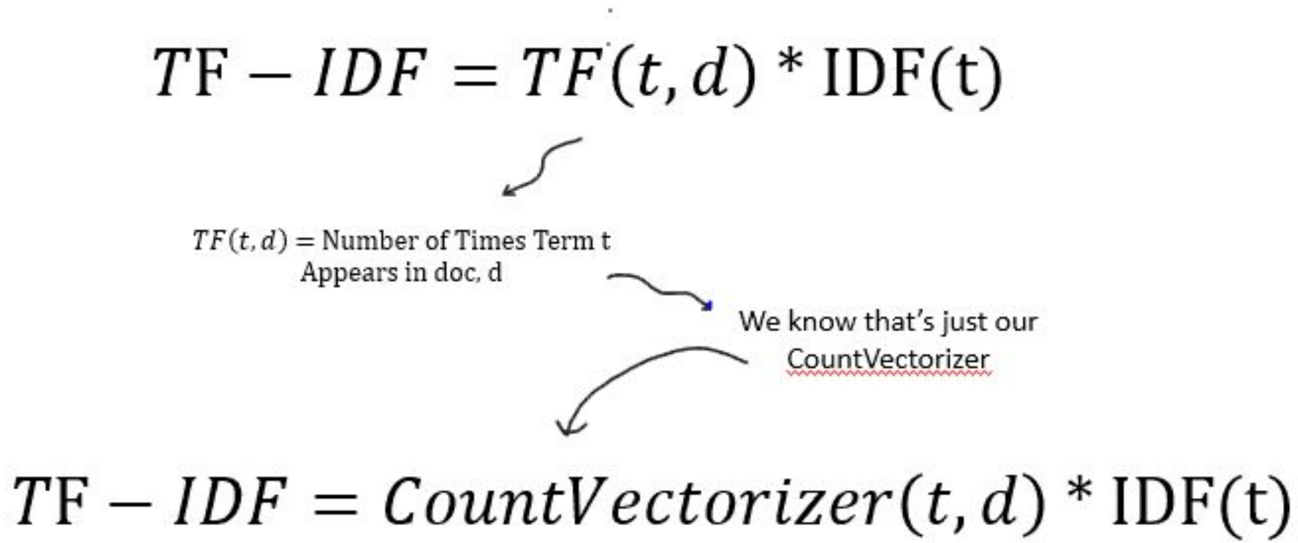
- Stands for Term Frequency(TF)-Inverse Document Frequency.
- **Term Frequency:** The probability of finding a word in the document.
- **Inverse Document Frequency:** How unique is the word in the total corpus.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

TF-IDF vs. CountVectorizer



$$TF - IDF = TF(t,d) * \mathrm{IDF}(t)$$

$TF(t,d) =$ Number of Times Term t Appears in doc, d

We know that's just our CountVectorizer

$$TF - IDF = CountVectorizer(t,d) * \mathrm{IDF}(t)$$

HandsOn Coding

https://drive.google.com/file/d/15jeXegNyRnyD7bI7MGopa0DUCyQCD8Lv/view?usp=sharing

Extra

Word Representation

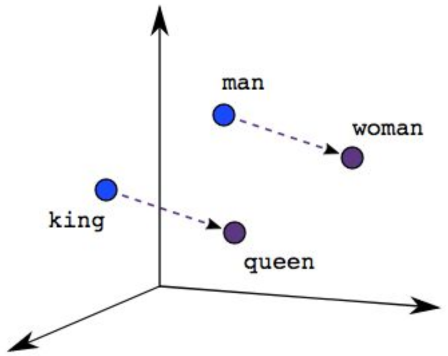
Base Idea:

- A popular idea in modern machine learning is to **represent words by vectors**
- Need a large text corpus
- Depending on the corpus,, word vectors will capture different information.

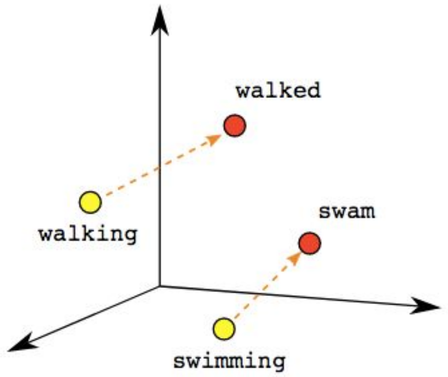
Why?

- capture hidden information about a language, like word analogies or semantic
- also used to improve performance of text classifiers

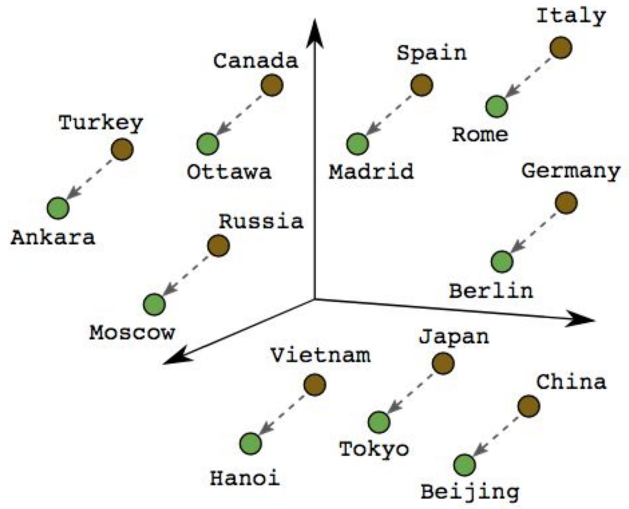
Word Representation



Male-Female

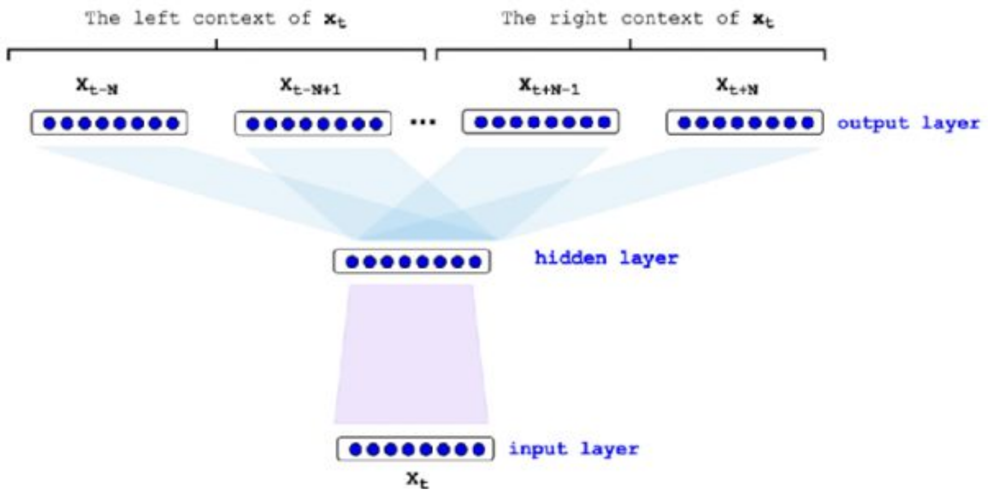


Verb Tense



Country-Capital

FastText

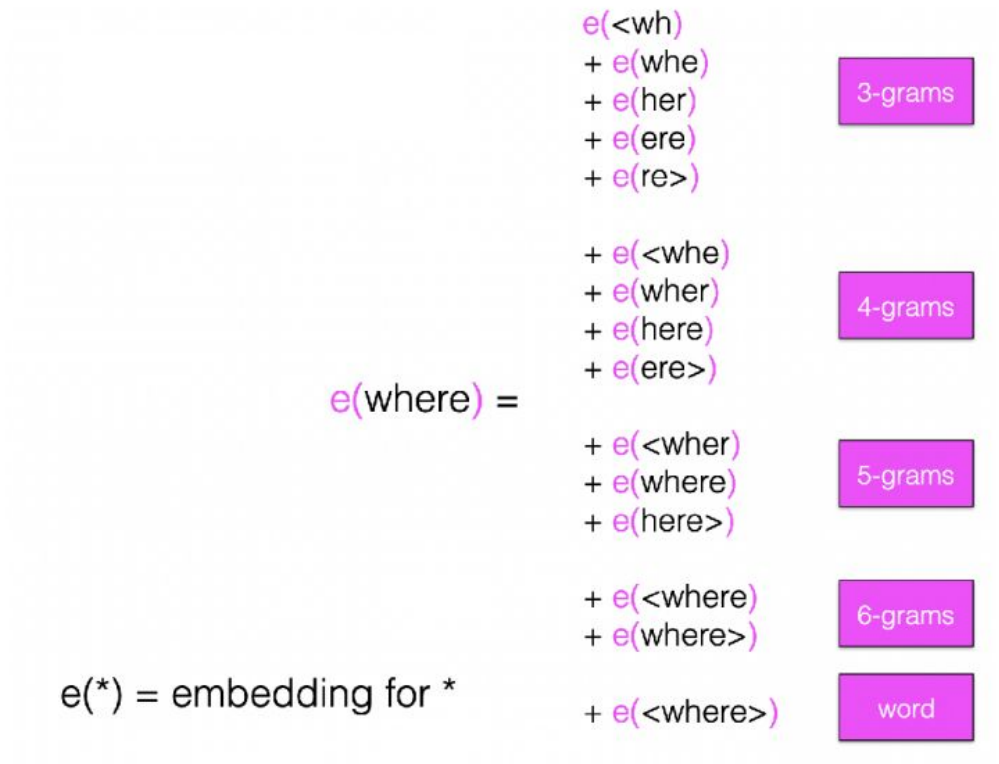


$$\sum_{t=1}^{M} \sum_{c \in [t-N, t+N]} \log p(w_c \mid w_t) \longrightarrow \sum_{t=1}^{T} \left[\sum_{c \in [t-N, t+N]} -\log\left(1 + e^{-s(w_t,\, w_c)}\right) - \sum_{w_r \in \mathcal{N}_{t,c}} \log\left(1 + e^{s(w_t,\, w_r)}\right) \right]$$

$$p(w_c \mid w_t; \theta) = \frac{e^{v_c \cdot v_t}}{\sum_{c' \in \mathcal{C}} e^{v_{c'} \cdot v_t}}$$

$$s(w_t, w_c) = \sum_{x \in \mathcal{S}_{w_t}} v_x^\top v_c$$

FastText


e(where) =
e(<wh)
+ e(whe)
+ e(her)
+ e(ere)
+ e(re>)
3-grams
+ e(<whe)
+ e(wher)
+ e(here)
+ e(ere>)
4-grams
+ e(<wher)
+ e(where)
+ e(here>)
5-grams
+ e(<where)
+ e(where>)
6-grams
+ e(<where>)
word
e(*) = embedding for *

HandsOn Coding

[https://drive.google.com/file/d/1a9cvG_cSes4YWvi8QLZqEa
hZOPDd96Q_/view?usp=sharing](https://drive.google.com/file/d/1a9cvG_cSes4YWvi8QLZqEahZOPDd96Q_/view?usp=sharing)

Open-source Libraries

- NLTK
- Hazm
- Sklearn

Questions?