National Technical
University of Ukraine
"Igor Sikorsky
Kyiv Polytechnic Institute"

Institute of
Physics and
Technology

Intellectual Data Analysis

Practice 4: Introduction to Natural Language Processing (NLP)

Dr. Nataliya K. Sakhnenko

Please review Lecture 5 before this practical

Text preprocessing

- **NLTK:** natural language toolkit

Basic Preprocessing Text
- Sentence Tokenization
- Word Tokenization
- Text Lemmatization and Stemming
- Stop Words
- Regex

Text to Vector Representations

Bag-of-words

A **bag-of-words** model describes the occurrence of each word within a document.

To use this model, we need to:

- Design a **vocabulary** of known words (also called **tokens**)

- Choose a **measure of the presence** of known words

- Any information about **the order** or **structure** of words **is discarded**.

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

fairy always love to it
it whimsical it I
and seen are
friend anyone
happy dialogue
adventure recommend
who sweet of satirical
it I but to romantic it I
several yet
the again it the humor
seen would
to scenes I the manages
the times and
fun I and about and
whenever while
conventions have
with

- unigram (1-gram):

a	swimmer	likes	swimming	thus	he	swims

- bigram (2-gram):

a swimmer	swimmer likes	likes swimming	swimming thus	...

- trigram (3-gram):

a swimmer likes	swimmer likes swimming	likes swimming thus	...
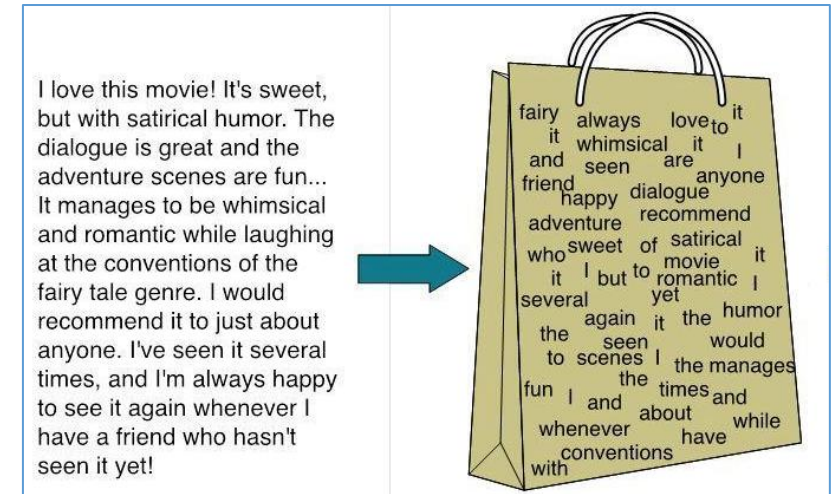
A bag-of-n-grams model represents a text document as an unordered collection of its n-grams.

Word frequency

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
corpus = [ 'This is the first document.',  'This document is the second
document.',  'And this is the third one.',  'Is this the first document?']
```

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
vectorizer.get_feature_names()
```

CountVectorizer: convert a collection of text documents to a matrix of token counts

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

```
X.toarray()
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

sklearn.feature_extraction.text.**CountVectorizer** (*lowercase=True, stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word',max_df, min_df, vocabulary=None*)

TF-IDF

One problem with **scoring word frequency** is that the most frequent words in the document start to have the highest scores. These frequent words may not contain as much "**informational gain**" to the model compared with some rarer and domain-specific words.

TF-IDF, short for **term frequency-inverse document frequency**

$$TF(\textbf{term}) = \frac{Number\ of\ times\ \textbf{term}\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$

$$IDF(\textbf{term}) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ \textbf{term}\ in\ it}\right)$$

$$TFIDF(\textbf{term}) = TF(\textbf{term}) * IDF(\textbf{term})$$

Parameters are similar to CounVectorizer

sklearn.feature_extraction.text.**TfidfVectorizer**()

✓ Topic identification (text classification): Is this news article about Politics, Sports, or Technology?

✓ Spam detection: Is this email a spam or not?

✓ Sentiment analysis: Is this movie review positive or negative?