

KPI BIGDATA CLUB  
ПРЕДСТАВЛЯЕТ

# Intro to Natural Language Processing

## ПРАКТИКУМ



15 НОЯБРЯ | 18:00  
BELKA SPACE

# Definition

- **Natural language processing** is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages.



# Common NLP Tasks



Easy



Medium



Hard

- |  |   |   |
|--|---|---|
| <ul style="list-style-type: none"><li>• Part-of-Speech Tagging</li></ul>   | <ul style="list-style-type: none"><li>• Syntactic Parsing</li></ul>         | <ul style="list-style-type: none"><li>• Machine Translation</li></ul>       |
| <ul style="list-style-type: none"><li>• Named Entity Recognition</li></ul> | <ul style="list-style-type: none"><li>• Word Sense Disambiguation</li></ul> | <ul style="list-style-type: none"><li>• Text Generation</li></ul>           |
| <ul style="list-style-type: none"><li>• Spam Detection</li></ul>           | <ul style="list-style-type: none"><li>• Sentiment Analysis</li></ul>        | <ul style="list-style-type: none"><li>• Automatic Summarization</li></ul>   |
| <ul style="list-style-type: none"><li>• Thesaurus</li></ul>                | <ul style="list-style-type: none"><li>• Topic Modeling</li></ul>            | <ul style="list-style-type: none"><li>• Question Answering</li></ul>        |
|  | <ul style="list-style-type: none"><li>• Information Retrieval</li></ul>     | <ul style="list-style-type: none"><li>• Conversational Interfaces</li></ul> |

**NLTK**

# NLTK

**Language:** Python

**Area:** Natural Language Processing

**Usage:** Symbolic and statistical natural language processing

**Advantages:**

- easy-to-use

- over 50 corpora and lexical resources such as WordNet

- a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning

# Tokenization

# Tokenization

**tokenization** is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens

Wikipedia

# Tokenization

into sentences

into words

`nltk.tokenize.sent_tokenize()`

`nltk.tokenize.word_tokenize()`

! punctuation == word



# Tokenize not-english text

There are total 17 european languages that NLTK support for sentence tokenize, and you can use them as the following steps:

Here is a spanish sentence tokenize example:

```
>>> spanish_tokenizer = nltk.data.load('tokenizers/punkt/spanish.pickle')
```

```
>>> spanish_tokenizer.tokenize('Hola amigo. Estoy bien.')
```

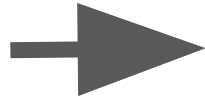
```
['Hola amigo.', 'Estoy bien.']
```

price

.

The  
U.S.  
and  
China

increased  
the  
number  
of  
supercomputers

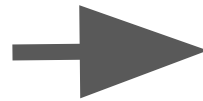


price

.

The  
U.S.  
and  
China

increased  
the  
number  
of  
supercomputers



price  
U.S.

China  
increased  
number  
supercomputers

# Stop Words

[illegible]

help away  
hour  
schit  
java  
kids  
headed  
thinking  
class  
world  
perfect  
shake  
few  
show  
fix  
lady  
shop  
left  
least  
ward

saw  
said  
stuff  
mom  
ur  
music  
≤3  
look  
extra  
found  
thought  
pretty  
love  
put  
ugh  
big  
old  
trip  
#fb  
dear

what  
white  
being  
cold  
bruh  
try  
joe  
karen  
hit  
many  
without  
wards  
since  
most  
far  
back  
fun  
news  
him  
back  
mall  
friday

some day not was work going got need know  
if be today new so life but good up  
it's or me of into at  
oh you over out w/ morning much getting that cup  
who were frap house real enjoying mean them sad  
take wif beach having any thru hell milk  
stop way social their double near yay looking minutes feeling  
order guy back sitting her as viareally more latte where would find god walk thank  
food start we went very job girl little meeting wow car cool make yum  
last our yes tell right like have ice then do  
yeah place friends donuts mmm walking money every needs sweet wanna re  
they one what here want am had love time your lol give  
next sunny wake gonna tonight another hate things closed talking watch street year keep water cute rain never  
cafe green lunch too now get this from just off bed us  
card also still free can drink when think will leticed days  
taste yummy cards seattle yet might airport chillin these seriously crazy church shit  
hey chicko mocha caramel chai live people could tried thing man again  
wait vanilla open than waiting why home maybe cant only  
also still free can drink when think will leticed days

rt im go all down see com by are nice into say about watching  
while apples bring hot well ve best bad she his venti long times ever wish even friend night read  
guys dunkin whole afternoon makes before eat bean don't quick miss great are into say about watching

# Stop Words Lists

```
from nltk.corpus import stopwords  
  
stop = set(stopwords.words('english'))
```

153

**Terrier stop word** list – this is a pretty comprehensive stop word list published with the Terrier package:

<https://bitbucket.org/kganes2/text-mining-resources/downloads>

733

**Remove Punctuation**

# Regular Expressions

a **sequence of characters** that define a **search pattern**

Wikipedia



# pythex

Your regular expression:

```
(?P<street>\w*\W+), (?P<house_number>[буд.майд№ ]*\d+[а-яА-Я\\-]*\d*[а-яА-я/]*), (?P<city>Киев)
```

IGNORECASE

MULTILINE

DOTALL

VERBOSE

Your test string:

ул. Днепро́вская наб., 14, Киев

Match result:

ул. Днепро́вская наб., 14, Киев

Match captures:

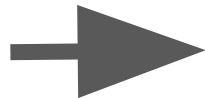
## Match 1

house_number	14
city	Киев
street	ул. Днепро́вская наб.

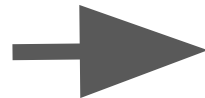
```
In [ ]: def removePunctuation(text):  
        p = re.compile('[^a-zA-Z0-9_ ]')  
        return p.sub('', text.lower()).strip()
```

- '[^a-zA-Z0-9\_ ]'  
Regex, any symbol but letters, numbers, '\_' and space
- `re.sub(pattern, repl, string, count=0, flags=0)`  
Return the string obtained by replacing the leftmost non-overlapping occurrences of *pattern* in *string* by the replacement *repl*.
- `string_name.lower()`  
Apply lowercase  
How Do You DO?      ->    how do you do?
- `string_name.strip([chars])`  
Delete spaces, '\n', '\r', '\t' in the beginning and in the end

price  
U.S.  
China  
increased  
number  
supercomputers



price  
U.S.  
China  
increased<sup>ed</sup>  
number  
supercomputers<sup>s</sup>



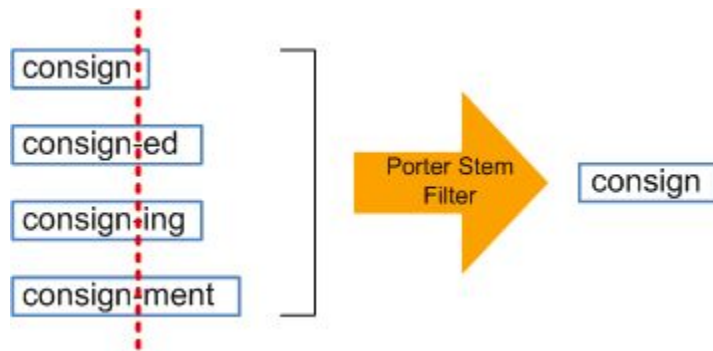
price  
U.S.  
China  
increase  
number  
supercomputer

# Stemming

# Stemming

**stemming** is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form

Wikipedia



# Lemmatization

# Lemmatization

**lemmatisation** (or **lemmatization**) is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form

Wikipedia

# Lemmatization result

cats

dishes

wolves

are

stopping

enjoyed



cat

dish

wolf

be

stop

enjoy



# Stemming vs Lemmatization

→ token normalization

*a.k.a. token "regularization"*

*(although that is technically the wrong wording)*

## • Stemming

- › produced by "**stemmers**"
- › produces a word's "stem"

- › am → am
- › the going → the go
- › having → hav

- › fast and simple (pattern-based)
- › **Snowball; Lovins; Porter**
- `nltk.stem.*`

## • Lemmatization

- › produced by "**lemmatizers**"
- › produces a word's "lemma"

- › am → be
- › the going → the going
- › having → have

- › requires: a dictionary and PoS
- › **LemmaGen; morpha**
- `nltk.stem.wordnet`

the lemmatize method default pos  
argument is “n” == noun!

# Speech Tagging

# Simplified Tagset of NLTK

Tag	Meaning	Examples
ADJ	adjective	<i>new, good, high, special, big, local</i>
ADV	adverb	<i>really, already, still, early, now</i>
CNJ	conjunction	<i>and, or, but, if, while, although</i>
DET	determiner	<i>the, a, some, most, every, no</i>
EX	existential	<i>there, there's</i>
FW	foreign word	<i>dolce, ersatz, esprit, quo, maitre</i>
MOD	modal verb	<i>will, can, would, may, must, should</i>
N	noun	<i>year, home, costs, time, education</i>
NP	proper noun	<i>Alison, Africa, April, Washington</i>
NUM	number	<i>twenty-four, fourth, 1991, 14:24</i>
PRO	pronoun	<i>he, their, her, its, my, I, us</i>
P	preposition	<i>on, of, at, with, by, into, under</i>
TO	the word <i>to</i>	<i>to</i>
UH	interjection	<i>ah, bang, ha, whee, hmpf, oops</i>
V	verb	<i>is, has, get, do, make, see, run</i>
VD	past tense	<i>said, took, told, made, asked</i>
VG	present participle	<i>making, going, playing, working</i>
VN	past participle	<i>given, taken, begun, sung</i>
WH	<i>wh</i> determiner	<i>who, which, when, what, where, how</i>

# More about tags

NLTK provides documentation for each tag, which can be queried using the tag, e.g. `nltk.help.upenn_tagset('RB')`, or a regular expression, e.g. `nltk.help.upenn_tagset('NN.*')`.

To get information about all tags just execute:

```
nltk.help.upenn_tagset()
```

# Word Count

## Word count

```
In [99]: all_words = nltk.FreqDist(lemmatized)
print(all_words.most_common(10))

[('bayes', 18), ('naive', 16), (u'classifier', 11), (u'feature', 7), (u'
model', 5), ('method', 4), ('classification', 4), ('bayesian', 4), ('pro
blem', 4), ('probability', 3)]
```

```
In [100]: print(all_words["problem"])
```

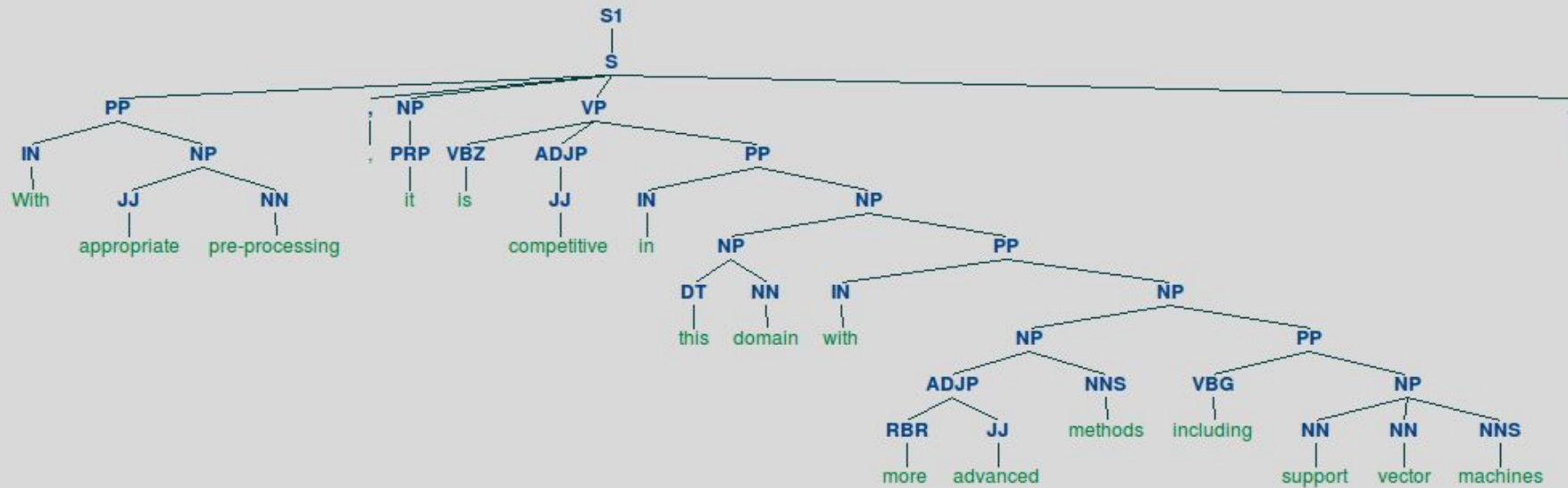
4

```
In [101]: print(len(all_words))
```

169

# Syntax Trees





With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines.

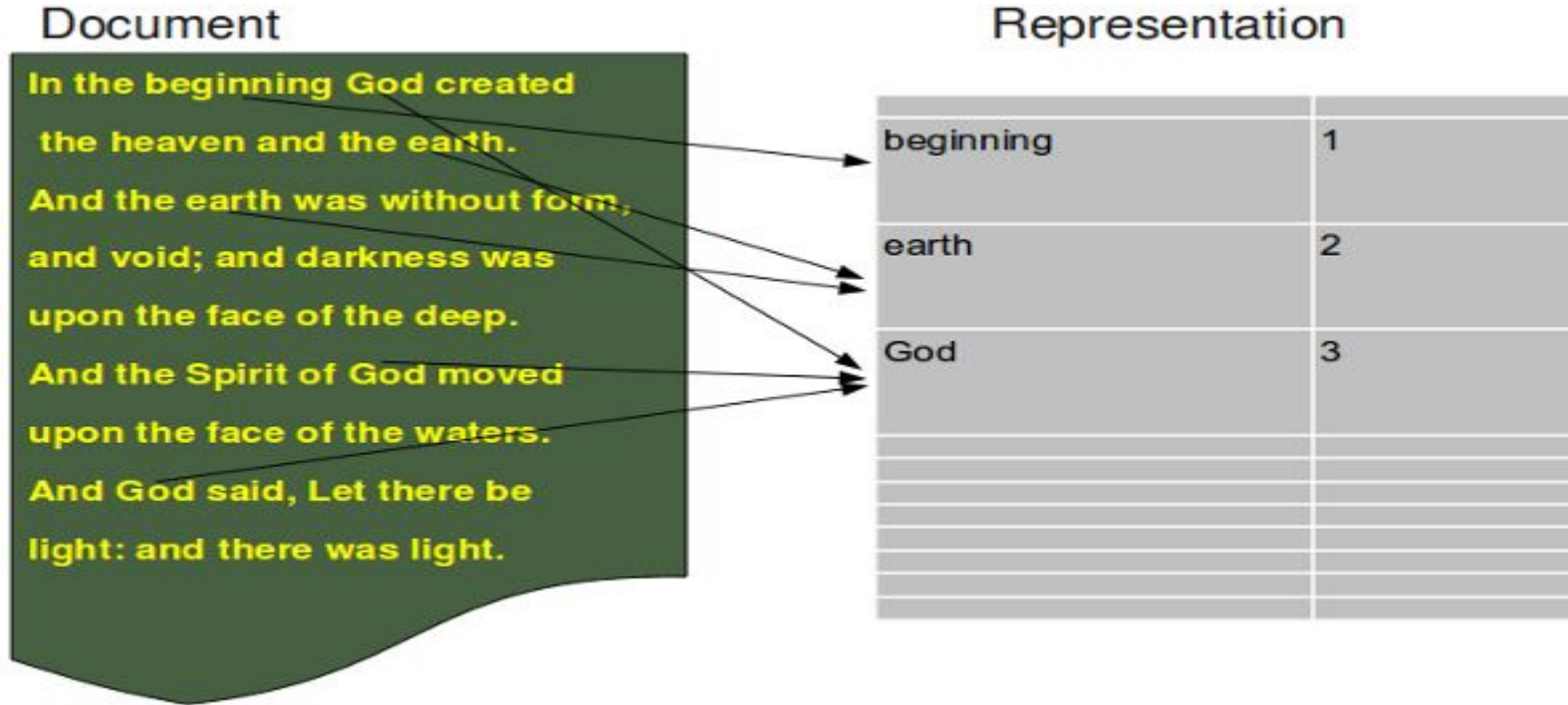
# Clustering with scikit-learn



# fetch\_20newsgroups

- **subset:** 'train' or 'test', 'all', optional :
- **categories:** None or collection of string or unicode :
- **shuffle:** bool, optional :  
Whether or not to shuffle the data: might be important for models that make the assumption that the samples are independent and identically distributed (i.i.d.), such as stochastic gradient descent.
- **random\_state:** numpy random number generator or seed integer :  
Used to shuffle the dataset.

# Clustering. Bag of words



# TF-IDF

- **term frequency-inverse document frequency** - a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

$$w_{x,y} = tf_{x,y} \times \log \left( \frac{N}{df_x} \right)$$

**TF-IDF**

Term  $x$  within document  $y$

$tf_{x,y}$  = frequency of  $x$  in  $y$

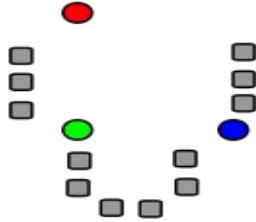
$df_x$  = number of documents containing  $x$

$N$  = total number of documents

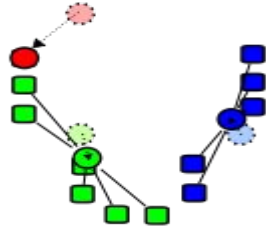
# sklearn.TfidfVectorizer

- **preprocessor** : callable or None (default)
- **tokenizer** : callable or None (default)
- **stop\_words** : string {'english'}, list, or None (default)
- **lowercase** : boolean, default True
- **max\_df** : float in range [0.0, 1.0] or int, default=1.0
- **min\_df** : float in range [0.0, 1.0] or int, default=1
- **max\_features** : int or None, default=None  
If not None, build a vocabulary that only consider the top max\_features ordered by term frequency across the corpus. This parameter is ignored if vocabulary is not None.

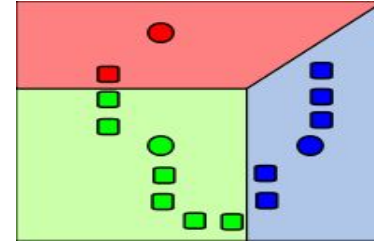
# k-means



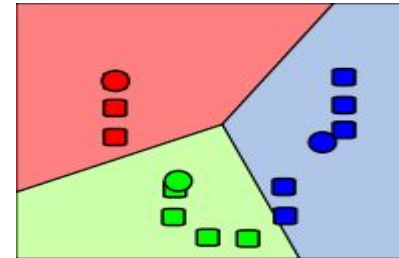
1.  $k$  initial "means" (in this case  $k=3$ ) are randomly generated within the data domain (shown in color).



3. The centroid of each of the  $k$  clusters becomes the new mean.



2.  $k$  clusters are created by associating every observation with the nearest mean.



4. Steps 2 and 3 are Repeated until convergence has been reached.

# sklearn.KMeans

- **n\_clusters** : int, optional, default: 8

- **max\_iter** : int, default: 300

- **n\_init** : int, default: 10

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.

- **init** : {'k-means++', 'random' or an ndarray}

Method for initialization, defaults to 'k-means++':

- 'k-means++' : selects initial cluster centers in a smart way to speed;
- 'random': choose k observations (rows) at random from data for the initial centroids.



# Metrics

- Homogeneity:

All of its clusters contain only data points which are members of a single class.

- Completeness

All the data points that are members of a given class are elements of the same cluster.

- V-measure:

$$v = 2 * (\text{homogeneity} * \text{completeness}) / (\text{homogeneity} + \text{completeness})$$

# Results

```
vectorizer = TfidfVectorizer(max_df=0.5, max_features=n_features,  
                             min_df=2, stop_words='english',  
                             use_idf=True)
```

```
km = KMeans(n_clusters=true_k, init='k-means++', max_iter=200, n_init=1)
```

Top terms per cluster:

Cluster 0: space nasa henry access digex gov toronto pat alaska writes shuttle article moon com just  
Cluster 1: window com mit server motif windows xterm application uk use problem widget file using display  
Cluster 2: god sandvik jesus com christian people kent brian writes koresh apple bible article say newton  
Cluster 3: sale com university 00 posting host nntp new distribution mail offer state drive 10 usa

**Thank you for your attention!**