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

- Part-of-Speech Tagging
- Named Entity Recognition
- Spam Detection
- Thesaurus
- Syntactic Parsing
- Word Sense Disambiguation
- Sentiment Analysis
- Topic Modeling
- Information Retrieval
- Machine Translation
- Text Generation
- Automatic Summarization
- Question Answering
- Conversational Interfaces

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

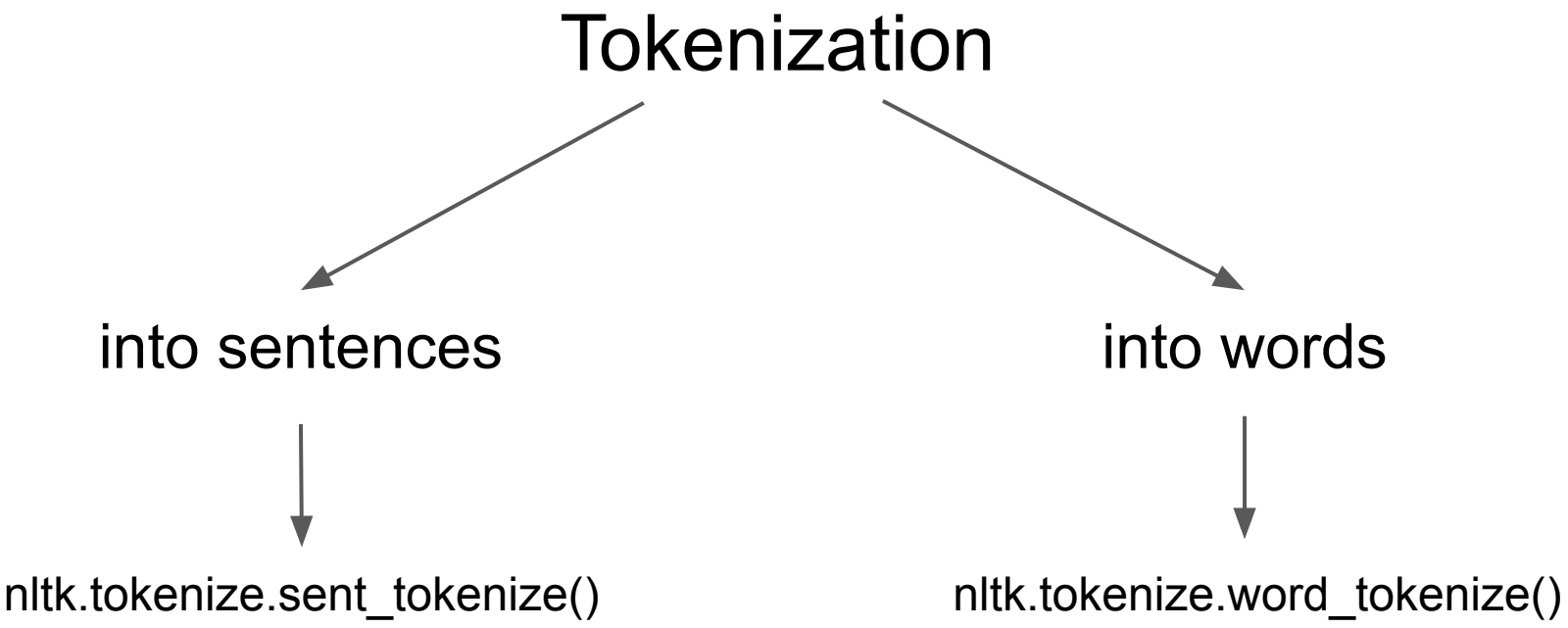
a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning

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

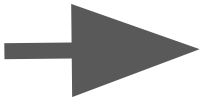
Tokenize not-english text

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

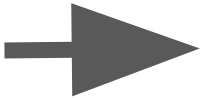
Here is a spanish sentence tokenize example:

```
>>> 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



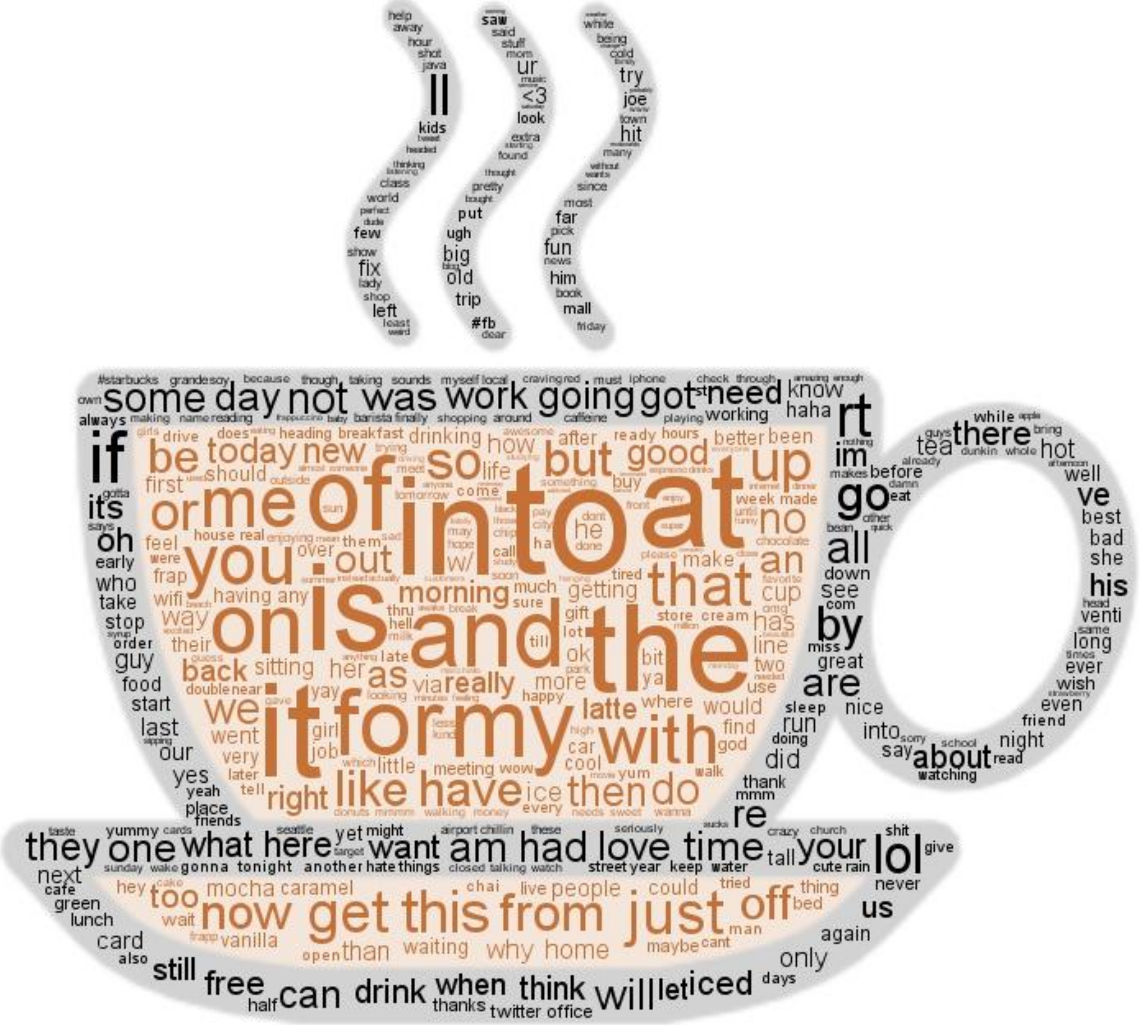
price
.
The
U.S.
and
China
increased
the
number
of
supercomputers



price
U.S.
China
increased
number
supercomputers

Stop Words


the
and
to
of
was
that
not
her
his
had
in
he
a
she
it
for
i
you
at
but
as
said
with
henchard
have
by
him
on
be
this
which
from
there
is
my
me
they
now
one
an
or
out
so
when
what
would
their
no
all
been
were
farfrae
elizabeth-jane
lucetta
man
who
your
will
such
then
like
could
upon
more
did
up
do
if
into
about
some
went
well
than
here
elizabeth
henchard's
casterbridge
down
before
back
time
door
little
should
two
very
house
other
first
come
ye
way



in to at of the is and on it for my you with that but so up me or be today new good going got need know work was not day some if rt go all by are there tea hot its oh like have right then do we as back really latte morning out no an cup they one what here want am had love time your lol now get this from just off too mocha caramel people could still free can drink when think will let iced only us again about into

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. Днепровская наб., 14, Киев

Match result:

ул. Днепровская наб., 14, Киев

Match captures:

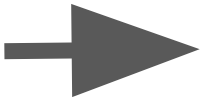
Match 1

house_number	14
city	Киев
street	ул. Днепровская наб.

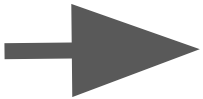
```
In [ ]: def removePunctuation(text):
            p = re.compile('[^a-zA-Z0-9_ ]')
            return p.sub('', text.lower()).strip()
```

- '[^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 lowcase
  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
number
supercomputers



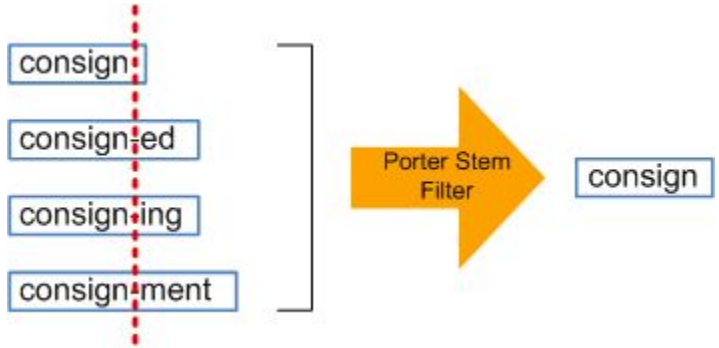
price
U.S.
China
increase
number
supercomputer

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	cat
dishes	dish
wolves	wolf
are	be
stopping	stop
enjoyed	enjoy

Stemming vs Lemmatization

➡ token normalization

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

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(lemmatizered)
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)]
```

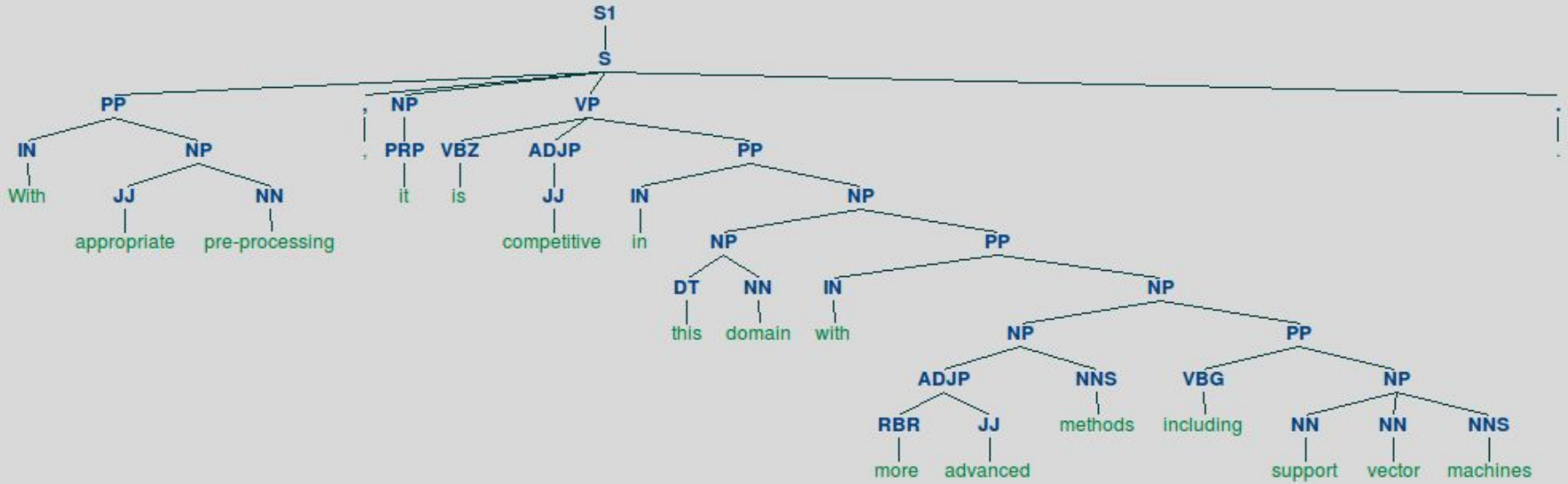
In [100]:
```
print(all_words["problem"])
```

```
4
```

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

```
169
```

Syntax Trees



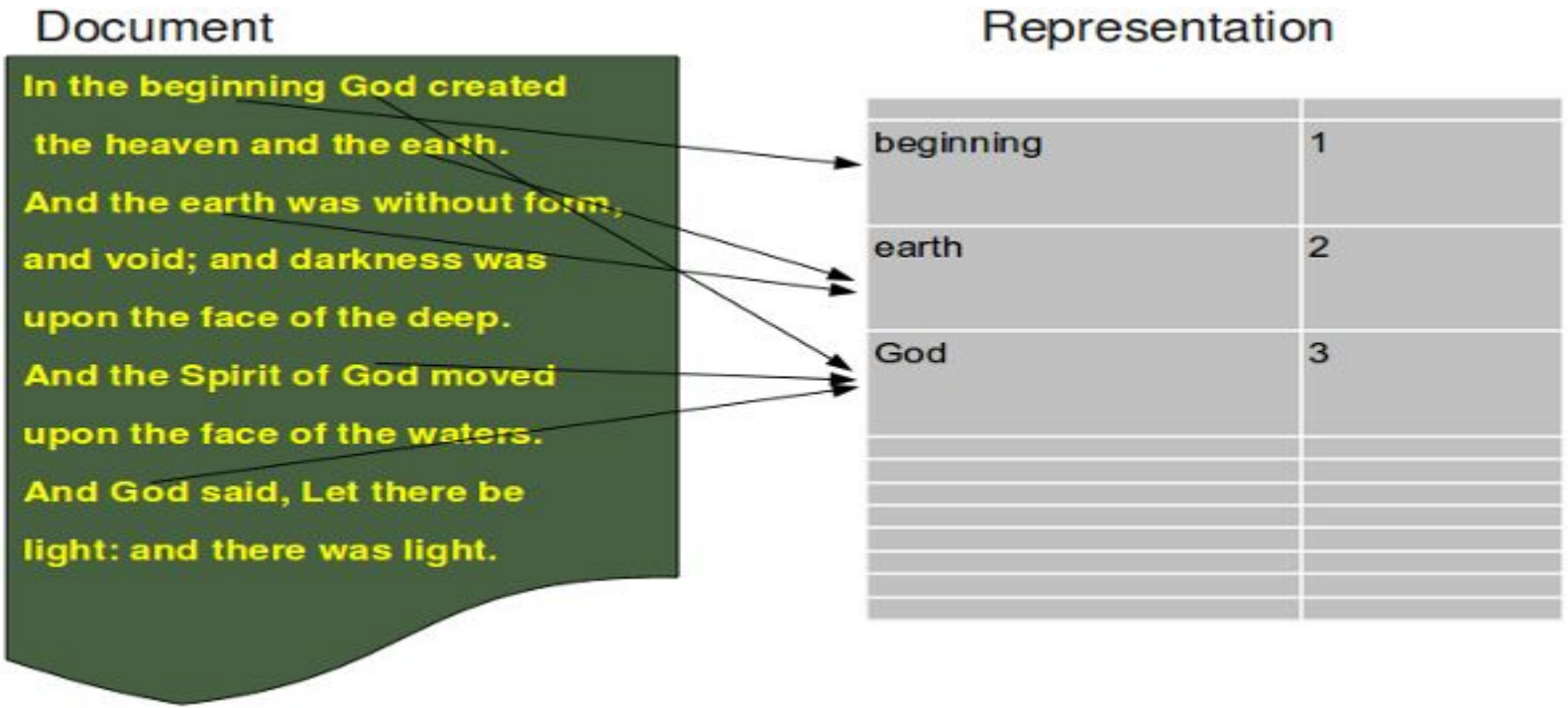
With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines.

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

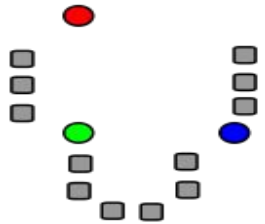
df_x = number of documents containing x

N = total number of documents

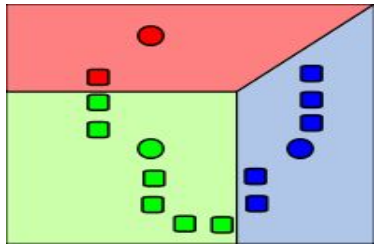
sklearn.TfidfVectorizer

- **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.

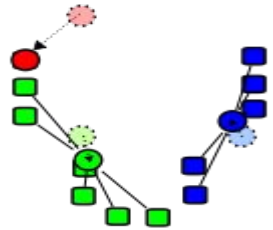
k-means



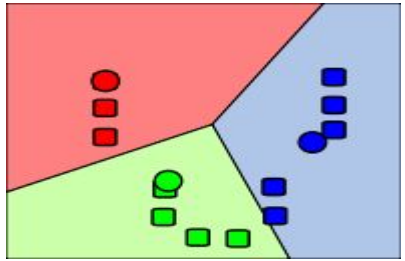
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



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



3. The centroid of each of the *k* clusters becomes the new 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 * (homogeneity * completeness) / (homogeneity + 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!