



Text Processing and Machine Learning

With Python



Fatih Erikli

Senior Software Developer

fatih_erikli@epam.com
<http://fatiherikli.github.io>



Text Processing

- Refers to the discipline of mechanising the creation, manipulation, or interpretation of text data.

CREATION

- Generating natural language from structured data.
- Auto-generated emails.

MANIPULATION

- Spell checkers
- Text formatters

INTERPRETATION

- Natural-language understanding.
- Sentiment analysis.
- Document classification.

Machine Learning

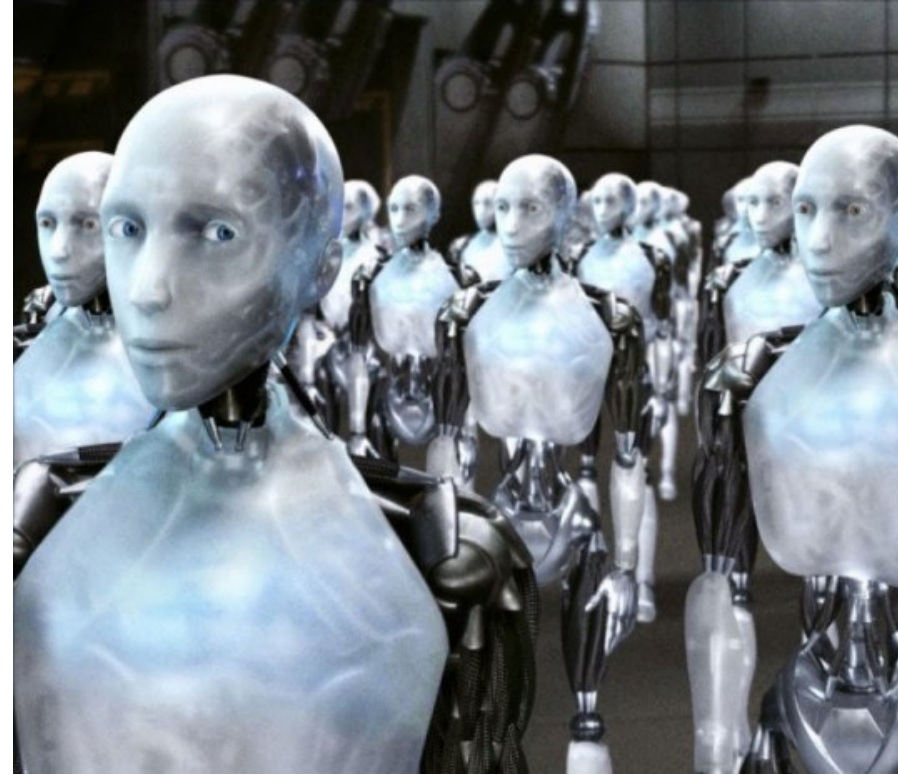
Starting more than half a century ago, scientists became very serious about addressing the question: **“Can we build a model that learns from available data and automatically makes the right decisions and predictions?”**

Machine learning is the science of getting computers to learn and perform. They learn how to perform by given training data, instead of explicitly given set of rules. They learn how to perform and act as same as humans do.

https://sebastianraschka.com/Articles/2014_naive_bayes_1.html

Will A.I. take over the world and kill us?

- A.I. is the intelligence demonstrated by machines.
- Machine learning is an application of artificial intelligence.
- A.I. won't kill us, unless we want them to do.



The difference

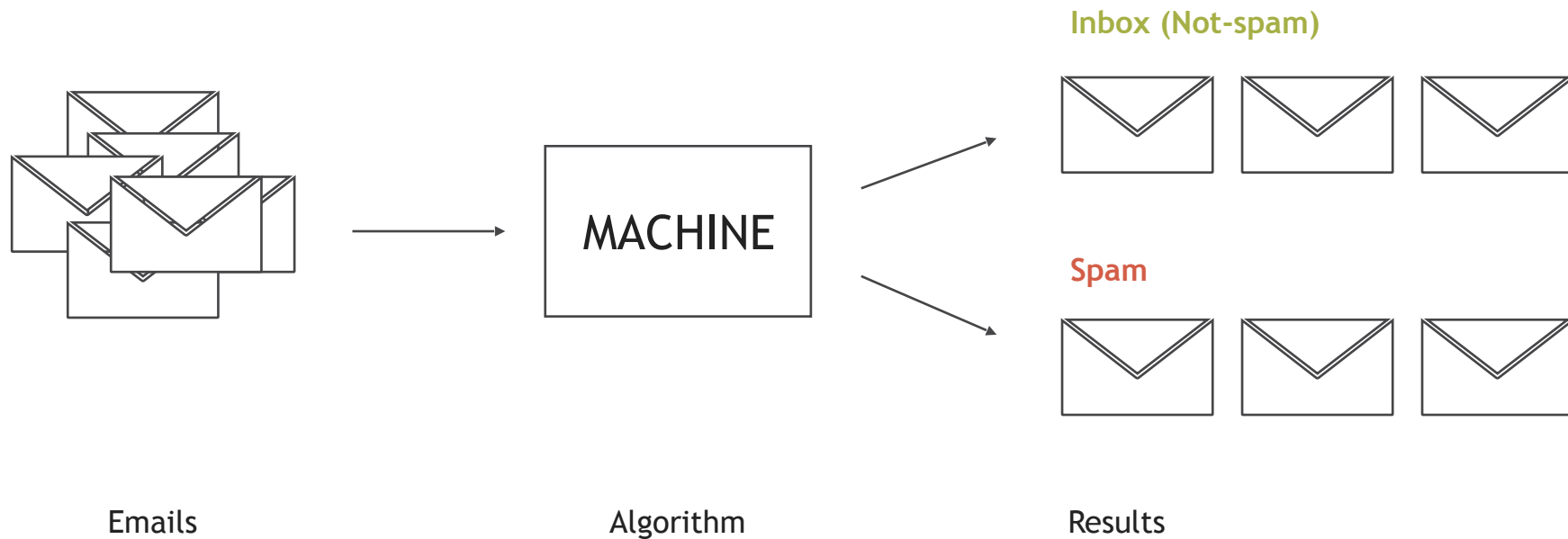
Traditional Approach



Machine Learning



An example application

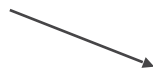


How they learn

Inbox (Not-spam)



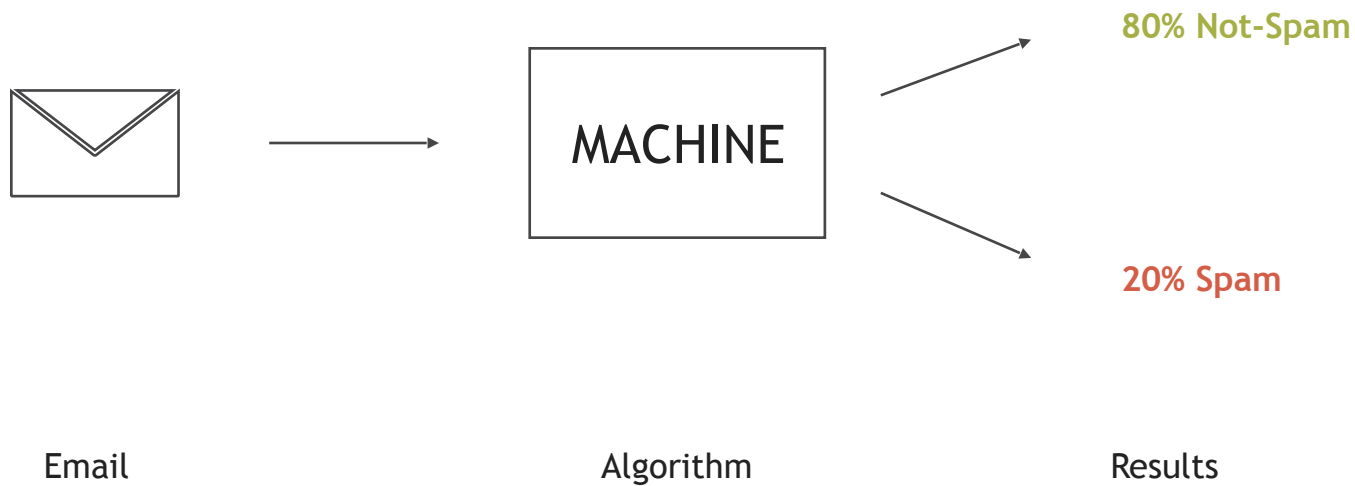
Spam



Training Data

Algorithm

How they perform

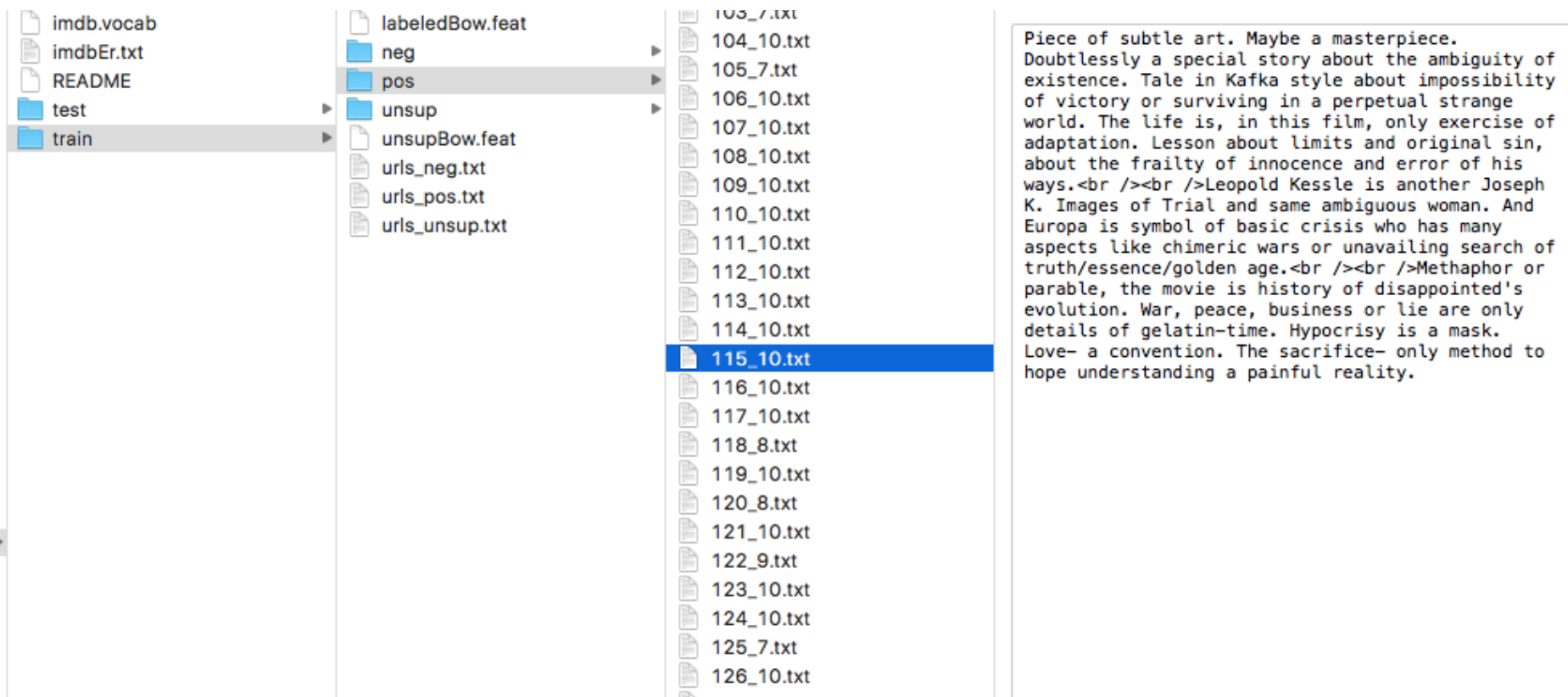




RESULTS CANNOT BE 100% ACCURATE.



Example training set: IMDB reviews



imdb.vocab
imdbEr.txt
README
test
train

labeledBow.feab
neg
pos
unsup
unsupBow.feab
urls_neg.txt
urls_pos.txt
urls_unsup.txt

103_7.txt
104_10.txt
105_7.txt
106_10.txt
107_10.txt
108_10.txt
109_10.txt
110_10.txt
111_10.txt
112_10.txt
113_10.txt
114_10.txt
115_10.txt
116_10.txt
117_10.txt
118_8.txt
119_10.txt
120_8.txt
121_10.txt
122_9.txt
123_10.txt
124_10.txt
125_7.txt
126_10.txt

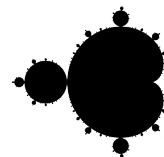
Piece of subtle art. Maybe a masterpiece.
Doubtlessly a special story about the ambiguity of
existence. Tale in Kafka style about impossibility
of victory or surviving in a perpetual strange
world. The life is, in this film, only exercise of
adaptation. Lesson about limits and original sin,
about the frailty of innocence and error of his
ways.

Leopold Kessle is another Joseph
K. Images of Trial and same ambiguous woman. And
Europa is symbol of basic crisis who has many
aspects like chimeric wars or unavailing search of
truth/essence/golden age.

Methaphor or
parable, the movie is history of disappointed's
evolution. War, peace, business or lie are only
details of gelatin-time. Hypocrisy is a mask.
Love- a convention. The sacrifice- only method to
hope understanding a painful reality.

TOOLS

- NLTK
Natural Language Toolkit
- TextBlob
TextBlob: Simplified Text Processing
- Jupyter Notebook
A web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.



TextBlob



jupyter

python -m nltk.downloader

NLTK Downloader

Collections Corpora Models All Packages

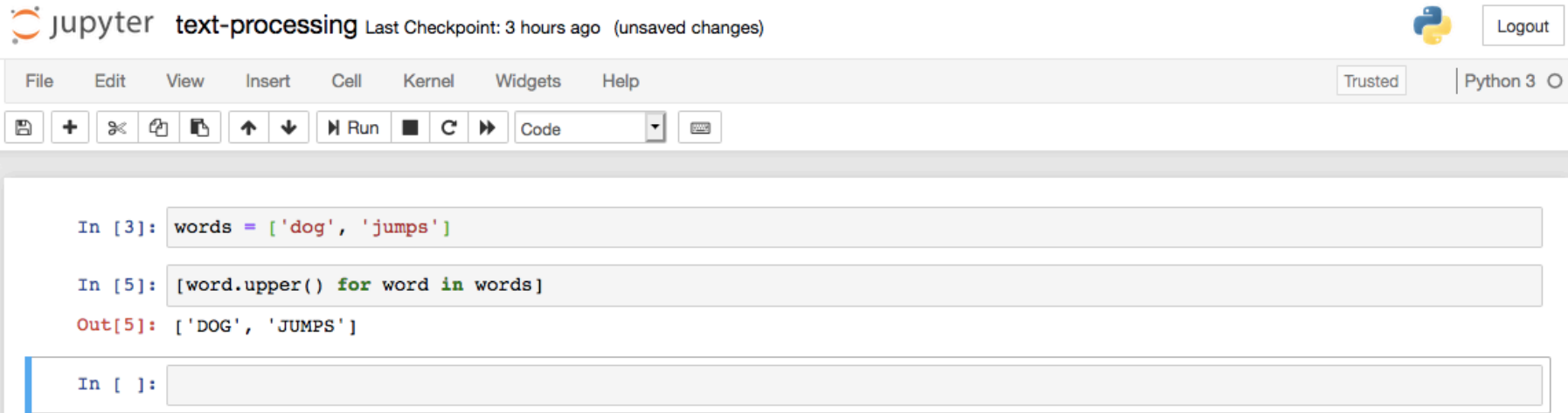
Identifier	Name	Size	Status
abc	Australian Broadcasting Commission 2006	1.4 MB	installed
alpino	Alpino Dutch Treebank	2.7 MB	installed
averaged_perceptron_tagg	Averaged Perceptron Tagger	2.4 MB	not installed
averaged_perceptron_tagg	Averaged Perceptron Tagger (Russian)	8.2 MB	not installed
basque_grammars	Grammars for Basque	4.6 KB	not installed
biocreative_ppi	BioCreAtivE (Critical Assessment of Information Extraction Sys	218.3 KB	installed
blip_wsj_no_aux	BLLIP Parser: WSJ Model	23.4 MB	not installed
book_grammars	Grammars from NLTK Book	8.9 KB	not installed
brown	Brown Corpus	3.2 MB	installed
brown_tei	Brown Corpus (TEI XML Version)	8.3 MB	installed
cess_cat	CESS-CAT Treebank	5.1 MB	installed
cess_esp	CESS-ESP Treebank	2.1 MB	installed
chat80	Chat-80 Data Files	18.8 KB	installed
city_database	City Database	1.7 KB	installed
cmudict	The Carnegie Mellon Pronouncing Dictionary (0.6)	875.1 KB	installed
comparative_sentences	Comparative Sentence Dataset	272.6 KB	installed

Download Refresh

Server Index:

Download Directory:

An example notebook



The screenshot displays a Jupyter Notebook interface. At the top, the header shows the Jupyter logo, the notebook name "text-processing", and a status message "Last Checkpoint: 3 hours ago (unsaved changes)". On the right, there is a "Logout" button and a Python 3 logo. Below the header is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for "Trusted" and "Python 3". Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, and running code. The main area of the notebook contains three input cells. The first cell contains the code `words = ['dog', 'jumps']`. The second cell contains the code `[word.upper() for word in words]`. The third cell shows the output `Out[5]: ['DOG', 'JUMPS']`. Below the output, there is an empty input cell labeled `In []:`.

jupyter text-processing Last Checkpoint: 3 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run

```
In [3]: words = ['dog', 'jumps']
```

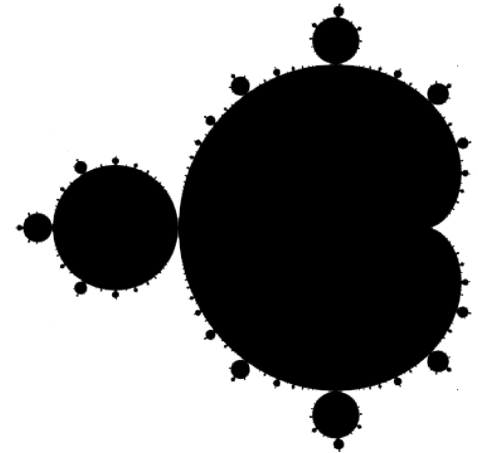
```
In [5]: [word.upper() for word in words]
```

```
Out[5]: ['DOG', 'JUMPS']
```

```
In [ ]:
```

TEXTBLOB: SIMPLIFIED TEXT PROCESSING

- Tokenization (Converting texts or words.)
- Builtin classification algorithms
- Sentiment Analysis
- Integration with GoogleTranslate
- Access to NLTK dictionaries and algorithms
- Integration with WordNet database



TextBlob


```
In [4]: from textblob import TextBlob
```

```
blob = TextBlob('''
The titular threat of The Blob has always struck me as the ultimate movie
monster: an insatiably hungry, amoeba-like mass able to penetrate
virtually any safeguard, capable of--as a doomed doctor chillingly
describes it--"assimilating flesh on contact.
Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario
proposed by technological theorists fearful of
artificial intelligence run rampant.
''')
```

```
In [6]: blob.words
```

```
Out[6]: WordList(['The', 'titular', 'threat', 'of', 'The', 'Blob', 'has', 'always', 'struck', 'me', 'as', 'the', 'ultimate',
'movie', 'monster', 'an', 'insatiably', 'hungry', 'amoeba-like', 'mass', 'able', 'to', 'penetrate', 'virtually', 'any',
', 'safeguard', 'capable', 'of', 'as', 'a', 'doomed', 'doctor', 'chillingly', 'describes', 'it', 'assimilating', 'fle
sh', 'on', 'contact', 'Snide', 'comparisons', 'to', 'gelatin', 'be', 'damned', 'it', 's', 'a', 'concept', 'with', 't
he', 'most', 'devastating', 'of', 'potential', 'consequences', 'not', 'unlike', 'the', 'grey', 'goo', 'scenario', 'pr
oposed', 'by', 'technological', 'theorists', 'fearful', 'of', 'artificial', 'intelligence', 'run', 'rampant'])
```

```
In [7]: blob.sentences
```

```
Out[7]: [Sentence("
The titular threat of The Blob has always struck me as the ultimate movie
monster: an insatiably hungry, amoeba-like mass able to penetrate
virtually any safeguard, capable of--as a doomed doctor chillingly
describes it--"assimilating flesh on contact."),
Sentence("Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario
proposed by technological theorists fearful of
artificial intelligence run rampant.")]
```

Tokenization

Clean API to work with the structure of text.

```
In [16]: blob.sentences
```

```
Out[16]: [Sentence("
The titular threat of The Blob has always struck me as the ultimate movie
monster: an insatiably hungry, amoeba-like mass able to penetrate
virtually any safeguard, capable of--as a doomed doctor chillingly
describes it--"assimilating flesh on contact."),
Sentence("Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario
proposed by technological theorists fearful of
artificial intelligence run rampant.")]
```

```
In [20]: blob.words[33]
```

```
Out[20]: 'describes'
```

```
In [21]: blob.words[33].lemmatize("v")
```

```
Out[21]: 'describe'
```

```
In [ ]:
```

Spelling Correction

Usually about 70% accurate, the implementation based on Peter Norvig's "How to Write a Spelling Corrector"

```
In [155]: blob = TextBlob("I havv goood spelling!")
```

```
In [156]: blob.correct()
```

```
Out[156]: TextBlob("I have good spelling!")
```

```
In [157]: Word('goood').spellcheck()
```

```
Out[157]: [('good', 1.0)]
```

```
In [ ]:
```

Sentiment Analysis

Polarity is a value between -1.0 and 1.0,
Subjectivity is between 0.0 and 1.0

```
In [22]: comment = TextBlob('Poland is a beautiful country')
```

```
In [23]: comment.sentiment
```

```
Out[23]: Sentiment(polarity=0.85, subjectivity=1.0)
```

```
In [34]: comment = TextBlob('Poland is a country located in Europe')
```

```
In [33]: comment.sentiment
```

```
Out[33]: Sentiment(polarity=0.0, subjectivity=0.0)
```

```
In [42]: comment = TextBlob("The worst restaurant I have ever seen")
```

```
In [43]: comment.sentiment
```

```
Out[43]: Sentiment(polarity=-1.0, subjectivity=1.0)
```

```
In [ ]:
```

Translation

Integration with GoogleTranslate API.

```
In [44]: comment = TextBlob('Poland is a country located in Europe')
```

```
In [47]: translated = comment.translate(to='pl')
```

```
In [48]: translated
```

```
Out[48]: TextBlob("Polska to kraj położony w Europie")
```

```
In [49]: translated.detect_language()
```

```
Out[49]: 'pl'
```

```
In [51]: translated.translate(to='tr')
```

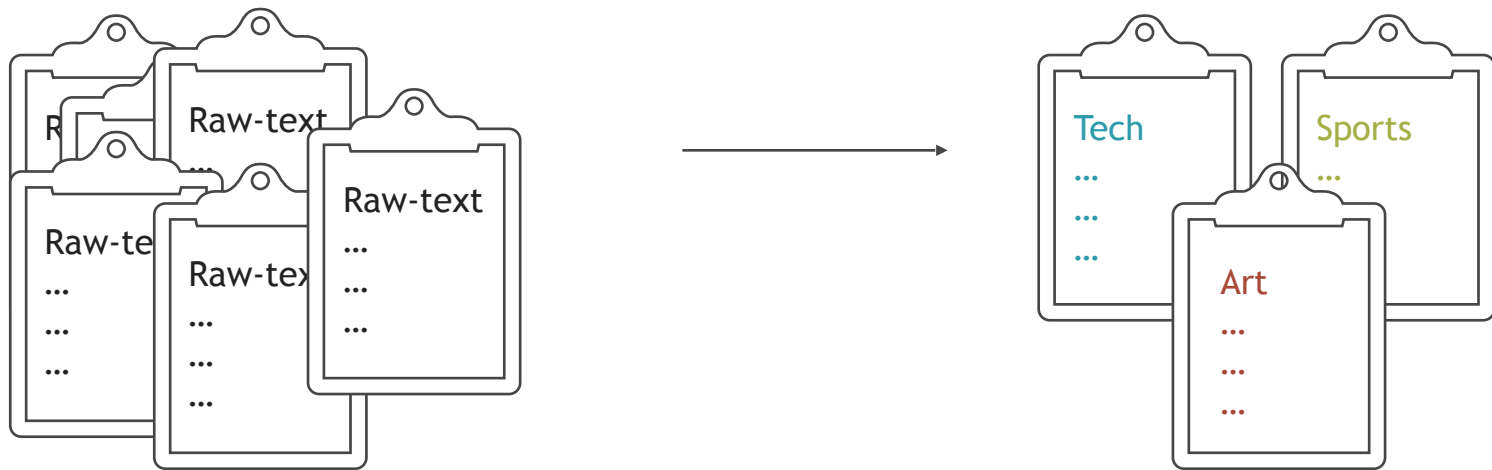
```
Out[51]: TextBlob("Polonya Avrupa'da bulunan bir ülkedir")
```

```
In [ ]:
```

BUILDING A CLASSIFIER

Text classification

Text classification is the process of assigning tags or categories to text according to its content.



Training Data

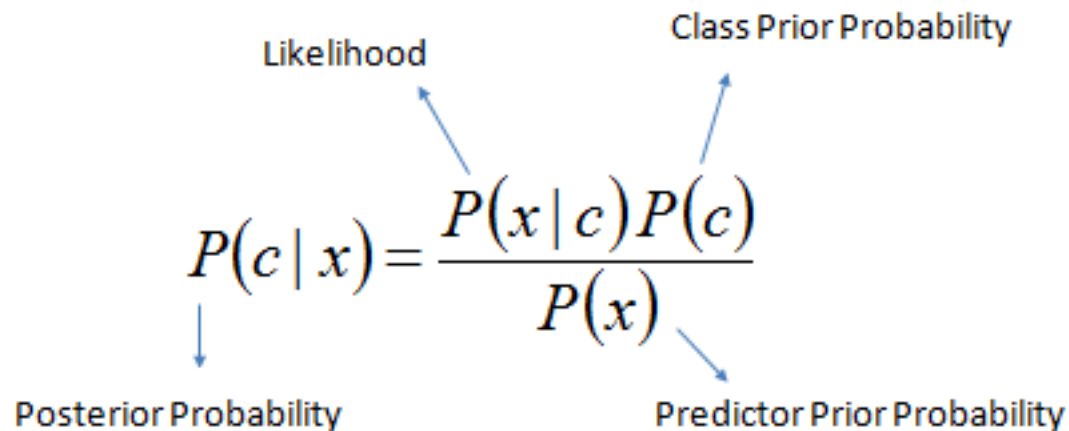
Training data should contain a raw text and pre-assigned label.
Also having a test data set is important to check classifiers health.

```
In [56]: train = [  
    —>('Time is not linear.', 'science'),  
    —('There is no such thing as A.I', 'science'),  
    —('Science is evidence based process.', 'science'),  
    —('The Universe does not "expand"', 'science'),  
    —('Python is better than Ruby', 'programming'),  
    —('You can\'t measure how good a programming language is.', 'programming'),  
    —('C++ is better for programming PC apps than Java', 'programming'),  
    —('Semi-colons are pointless in Javascript', 'programming'),  
    —('Emacs is better than vim', 'programming'),  
    —('Art is for soceity, not for profit.', 'art'),  
    —('Picasso was a brilliant artist.', 'art'),  
    —('Picasso pioneered cubism.', 'art'),  
    ]
```

```
In [57]: test = [  
    —('Time is relative', 'science'),  
    —('Cubism is an avant-garde movement', 'art'),  
    —('Python is created by Guido Van Rossum', 'programming')  
    ]
```


Naive Bayes Classification

Naive Bayes is a probabilistic classifier inspired by the Bayes theorem.



The diagram shows the Naive Bayes formula with arrows pointing from descriptive labels to the corresponding parts of the equation:

$$P(c | x) = \frac{P(x | c) P(c)}{P(x)}$$

- Likelihood** points to $P(x | c)$
- Class Prior Probability** points to $P(c)$
- Posterior Probability** points to $P(c | x)$
- Predictor Prior Probability** points to $P(x)$

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

<https://medium.com/datadriveninvestor/classification-algorithms-in-machine-learning-85c0ab65ff4>

Naive Bayes Classification

Type	Long	Not Long	Sweet	Not Sweet	Yellow	Not Yellow	Total
Banana	400	100	350	150	450	50	500
Orange	0	300	150	150	300	0	300
Other	100	100	150	50	50	150	200
Total	500	500	650	350	800	200	1000

<https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/>

TextBlob implementation

```
In [58]: from textblob.classifiers import NaiveBayesClassifier
```

```
In [59]: classifier = NaiveBayesClassifier(train)
```

```
In [60]: classifier.classify('Time is relative')
```

```
Out[60]: 'science'
```

```
In [61]: classifier.classify('Picasso is a painter')
```

```
Out[61]: 'art'
```

```
In [64]: probs = classifier.prob_classify('Picasso is a painter')
```

```
In [65]: probs.prob('art')
```

```
Out[65]: 0.7868710930095634
```

```
In [66]: probs.prob('science')
```

```
Out[66]: 0.09233965615566112
```

```
In [ ]:
```

Feature Engineering

Features are the basic primitives of classification process. Most of the classifiers use simple feature extractor that indicates which words in the training set are contained in a document.

We can customise that feature extraction behaviour. The process of selecting what features should be taken into account in the classification process is called **feature engineering**.

Feature Engineering

```
In [168]: classifier.show_informative_features()
```

Most Informative Features

contains(is) = False	art : progra = 2.5 : 1.0
contains(Picasso) = False	progra : art = 2.4 : 1.0
contains(than) = False	scienc : progra = 2.2 : 1.0
contains(better) = False	scienc : progra = 2.2 : 1.0
contains(is) = True	progra : art = 2.0 : 1.0
contains(not) = False	progra : scienc = 1.8 : 1.0
contains(programming) = False	scienc : progra = 1.5 : 1.0
contains(a) = True	art : progra = 1.5 : 1.0
contains(for) = True	art : progra = 1.5 : 1.0
contains(cubism) = False	progra : art = 1.5 : 1.0

To get more accurate result, we might want to take only first and last words of the document. To achieve this, we need to create a feature extractor function.

```
def end_word_extractor(document):  
    tokens = document.split()  
    first_word, last_word = tokens[0], tokens[-1]  
  
    return {  
        'first({%s})' % first_word: True,  
        'last({%s})' % last_word: False,  
    }
```

Feature Engineering

```
In [68]: def end_word_extractor(document):  
    tokens = document.split()  
    first_word, last_word = tokens[0], tokens[-1]  
    return {  
        'first({s})' % first_word: True,  
        'last({s})' % last_word: False,  
    }
```

```
In [69]: classifier = NaiveBayesClassifier(train, feature_extractor=end_word_extractor)
```

```
In [76]: probs = classifier.prob_classify('Picasso is an artist.')
```

```
In [77]: probs.prob('art')
```

```
Out[77]: 0.9079205257292188
```

```
In [79]: 1 probs = classifier.prob_classify('Picasso is an painter.')
```

```
In [80]: probs.prob('art')
```

```
Out[80]: 0.7065948855989233
```

WORDNET

LEXICAL DATABASE FOR THE ENGLISH LANGUAGE

Wordnet

Wordnet is a graph dataset which each node is connected to each other. The top node is “Entity” and all of the words branch off from that node.

Lexical nodes can be useful for feature engineering.



Each entity is somehow connected to each other since everything is branched off from “Entity”. Thanks to that feature, we’re able to measure similarity between nodes.

```
In [140]: from textblob import Word
```

```
In [144]: poland = Word('poland').synsets[0]
```

```
In [150]: hungary = Word('hungary').synsets[0]
```

```
In [151]: poland.path_similarity(hungary)
```

```
Out[151]: 0.3333333333333333
```

```
In [152]: hungary
```

```
Out[152]: Synset('hungary.n.01')
```

```
In [153]: japan = Word('japan').synsets[0]
```

```
In [154]: poland.path_similarity(japan)
```

```
Out[154]: 0.09090909090909091
```

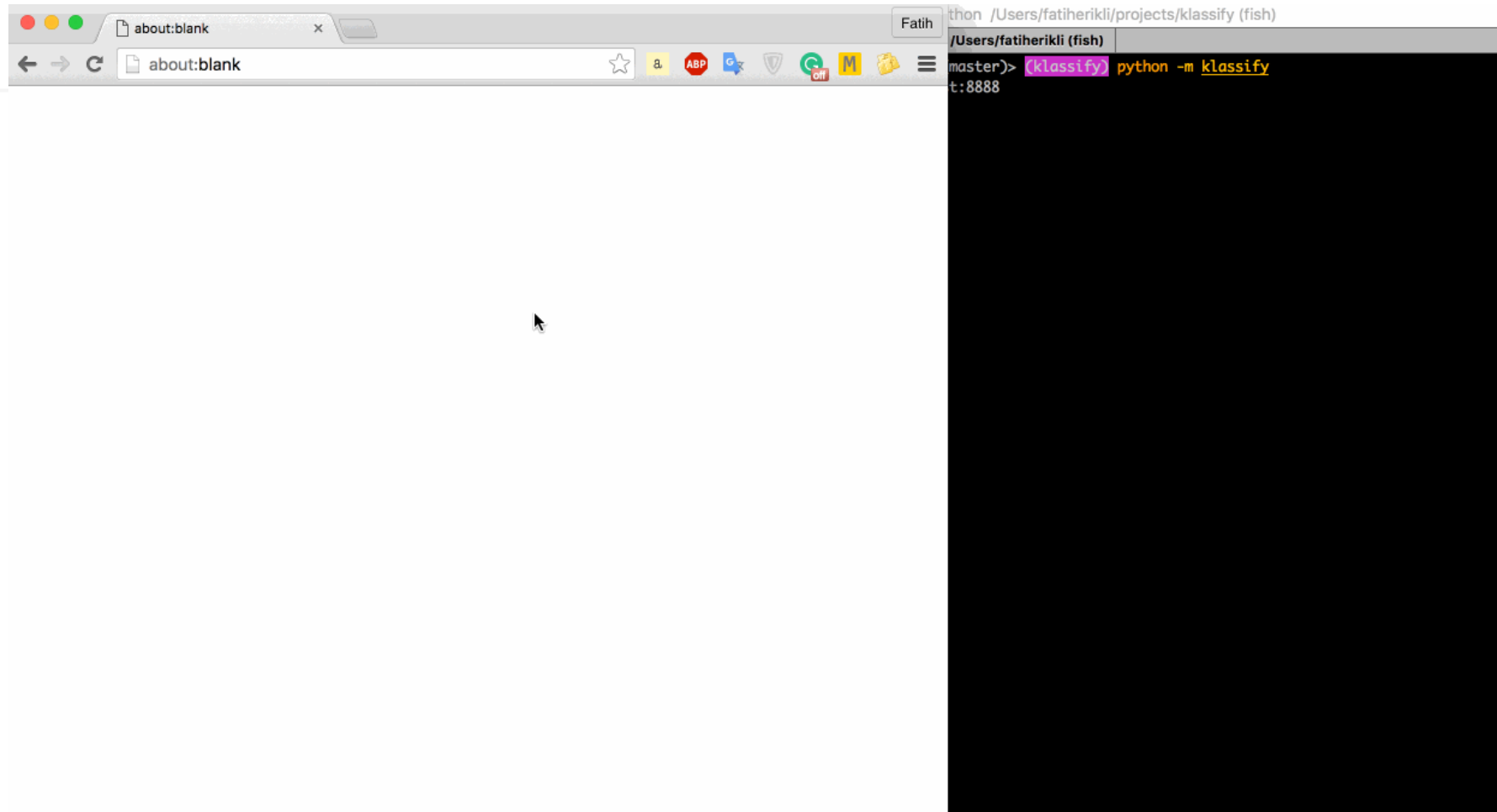
KLASSIFY
TEXT CLASSIFICATION SERVICE - POC

KLASSIFY

Bayesian Text classification service based on Redis. Built on top of Tornado and React.js
<https://github.com/fatihelikli/klassify>

It can be used for:

- Spam filters
- Web page classification
- News and and topic categorization
- Sentiment Analysis



SOURCES

- https://sebastianraschka.com/Articles/2014_naive_bayes_1.html
- <https://natureofcode.com/book/chapter-10-neural-networks/>
- <http://ml-playground.com/>
- <https://playground.tensorflow.org>
- <https://keras.io/>

QUESTIONS?