Concepts

P. S. Langeslag

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

String

A raw, unprocessed sequence of text as typically assigned to a (constant or) variable:

```
>>> phrase = 'the tortoise and the hare'
>>> print(phrase)
'the tortoise and the hare'
```

List / Array

A container or object containing a sequence of items (hence known as a “sequence type”):

```
>>> phraseList = ['the', 'tortoise', 'and', 'the', 'hare']
>>> phraseList
['the', 'tortoise', 'and', 'the', 'hare']
```

List / Array

A container or object containing a sequence of items (hence known as a “sequence type”):

```
>>> phraseList = ['the', 'tortoise', 'and', 'the', 'hare']
>>> phraseList
['the', 'tortoise', 'and', 'the', 'hare']
```

Python distinguishes between `list` and `array`. Both are **ordered**; their items are **mutable**; and they may be **non-unique**. But:

List / Array

A container or object containing a sequence of items (hence known as a “sequence type”):

```
>>> phraseList = ['the', 'tortoise', 'and', 'the', 'hare']
>>> phraseList
['the', 'tortoise', 'and', 'the', 'hare']
```

Python distinguishes between `list` and `array`. Both are **ordered**; their items are **mutable**; and they may be **non-unique**. But:

List

- Accessed through Python Standard Library
- May contain items of different data types

Array

- Accessed through `numPy` package or `array` module
- Only `numPy` arrays permit different data types

Tuple

An immutable (but reassignable) list of fixed length:

```
>>> certainties = ('death', 'taxes')
>>> certainties
('death', 'taxes')
>>> certainties.append('fulfilment')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  AttributeError: 'tuple' object has no attribute 'append'
```

Tuple

An immutable (but reassignable) list of fixed length:

```
>>> certainties = ('death', 'taxes')
>>> certainties
('death', 'taxes')
>>> certainties.append('fulfilment')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  AttributeError: 'tuple' object has no attribute 'append'
```

NB defining a single-member tuple involves a trailing comma:

```
>>> void = ('empty')
>>> void
'empty'
>>> void = ('empty',)
>>> void
('empty',)
```

List vs Tuple

"A list is typically a sequence of objects all having the *same type*, of *arbitrary length*. We often use lists to hold sequences of words. In contrast, a tuple is typically a collection of objects of *different types*, of *fixed length*. We often use a tuple to hold a **record**, a collection of different **fields** relating to some entity. This distinction between the use of lists and tuples takes some getting used to, so here is another example:

```
>>> lexicon = [
...     ('the', 'det', ['Di:', 'D@']),
...     ('off', 'prep', ['Qf', 'O:f'])
... ]
```

Here, a lexicon is represented as a list because it is a collection of objects of a single type — lexical entries — of no predetermined length. An individual entry is represented as a tuple because it is a collection of objects with different interpretations, such as the orthographic form, the part of speech, and the pronunciations [...]. Note that these pronunciations are stored using a list." (Bird et al. ch. 4)

Dictionary

An extensible list of mutable key-value pairs.

```
>>> settings = {'iso': 6400, 'aperture': 1.7, 'shutter': '1/200'}
>>> settings['aperture']
1.7
>>> settings['aperture'] = 5.6
>>> settings
{'iso': 6400, 'aperture': 5.6, 'shutter': '1/200'}
```

Dictionary

An extensible list of mutable key-value pairs.

```
>>> settings = {'iso': 6400, 'aperture': 1.7, 'shutter': '1/200'}
>>> settings['aperture']
1.7
>>> settings['aperture'] = 5.6
>>> settings
{'iso': 6400, 'aperture': 5.6, 'shutter': '1/200'}
```

In practice, it is often best to define a dictionary before populating it:

```
>>> settings = {}
>>> settings['aperture'] = 1.7
```

This avoids having to check whether the dictionary already exists before defining one of its keys.

The Syntax of Lists, Tuples, and Dictionaries in Python

```
>>> myList = ['value', 'value', 20]
>>> myTuple = ('value', 'value', 20)
>>> myDict = {'key1': 'value', 'key2': 'value', 'key3': 20}
```

The Syntax of Lists, Tuples, and Dictionaries in Python

```
>>> myList = ['value', 'value', 20]
>>> myTuple = ('value', 'value', 20)
>>> myDict = {'key1': 'value', 'key2': 'value', 'key3': 20}
>>> myList.append('newvalue') # takes one argument only
>>> myList
['value', 'value', 20, 'newvalue']
>>> myDict['key4'] = 'newvalue'
>>> myDict
{'key1': 'value', 'key2': 'value', 'key3': 20, 'key4': 'newvalue'}
```

(Word) Token

When reading a text (corpus) sequentially, each instance of a word form you encounter is its own token. We may write or employ functions to **tokenize** a text:

```
>>> phrase = 'the tortoise and the hare'
>>> phraseTokens = str.split(phrase)
>>> phraseTokens
['the', 'tortoise', 'and', 'the', 'hare']
```

(Word) Token

When reading a text (corpus) sequentially, each instance of a word form you encounter is its own token. We may write or employ functions to **tokenize** a text:

```
>>> phrase = 'the tortoise and the hare'
>>> phraseTokens = str.split(phrase)
>>> phraseTokens
['the', 'tortoise', 'and', 'the', 'hare']
>>> len(phraseTokens)
5
>>> phraseTokens.sort()
>>> phraseTokens
['and', 'hare', 'the', 'the', 'tortoise']
```

(Word) Token

When reading a text (corpus) sequentially, each instance of a word form you encounter is its own token. We may write or employ functions to **tokenize** a text:

```
>>> phrase = 'the tortoise and the hare'
>>> phraseTokens = str.split(phrase)
>>> phraseTokens
['the', 'tortoise', 'and', 'the', 'hare']
>>> len(phraseTokens)
5
>>> phraseTokens.sort()
>>> phraseTokens
['and', 'hare', 'the', 'the', 'tortoise']
```

(Word) Type

Each distinct orthographical form (i.e. spelling) in the corpus.

```
>>> phraseTypes = list(dict.fromkeys(phraseTokens))
>>> len(phraseTypes)
4
>>> phraseTypes
['and', 'hare', 'the', 'tortoise']
```

n-Gram

A sequence consisting of n words as they occur in a string of text.

n-Gram

A sequence consisting of n words as they occur in a string of text.

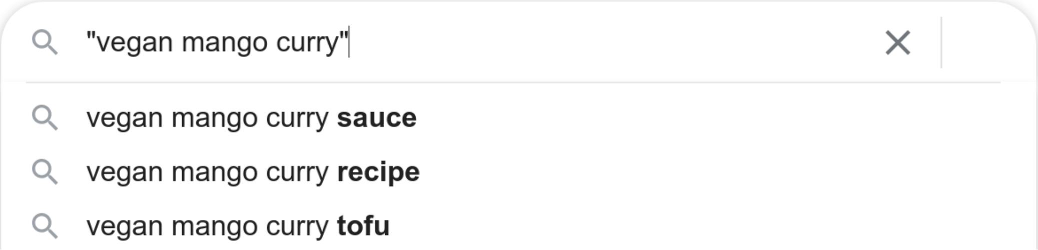


Figure 1: Double quotes yield n-grams on most search engines

n-Gram

A sequence consisting of n words as they occur in a string of text.

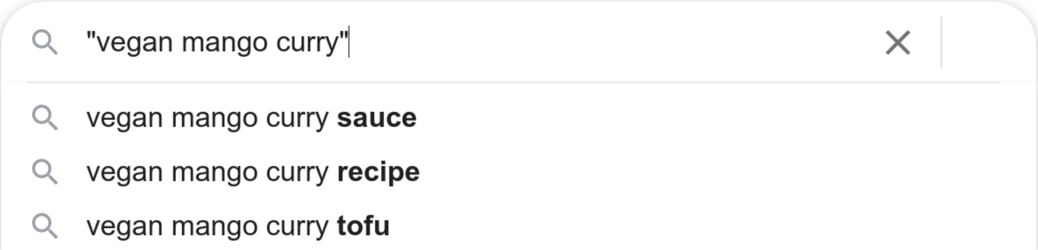
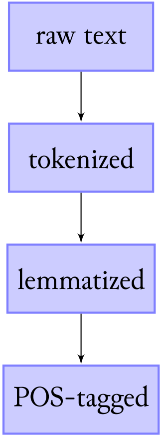


Figure 1: Double quotes yield n-grams on most search engines

- we speak of bigrams and trigrams but commonly write 2-gram, 3-gram
- n-grams offer the benefit of **dimension reduction**.

Pipeline



The processing sequence from input to the desired structured data.

Mathematical Operators

symbol	meaning
x	input variable
y	output variable
$\sum$ (sigma)	sum
$\prod$ (pi)	product
ĉ	predicted class
$\in$	is a member of the following set
$argmax$	the point at which the function values are maximized
$P(d\|c)$	the probability of d (document) given c (class)

Regular Expression

Search string relying on an extensive, conventional pattern-matching grammar

```
>>> import re
>>> haystack = "thesis2022-04-19q_formatted.md"
>>> needle = "^thesis[0-9]{4}-[0-9]{2}-[0-9]{2}[a-z]_formatted\..*$"
>>> re.search(needle, haystack)
<re.Match object; span=(0, 30), match='thesis2022-04-19q_formatted.md'>
```

Bag of Words

A model storing information on each word type (i.e. form) and its frequency in a text (corpus), but discarding syntax and word order.

```
>>> from collections import Counter
>>> Counter(phraseTokens)
Counter({'the': 2, 'tortoise': 1, 'and': 1, 'hare': 1})
```

Levenshtein Distance

A count of the character edits (addition, deletion, or substitution) required to turn one string into another.

```
>>> import Levenshtein
>>> string1 = "fisherman"
>>> string2 = "fisherwomen"
>>> Levenshtein.distance(string1, string2)
3
```

Stem

Linguistic Definition

The base of a given word form, to which inflectional information is added.

Stem

Linguistic Definition

The base of a given word form, to which inflectional information is added.

NLP Definition

The base to which a given type may be reduced by stripping away (known) inflectional (and sometimes derivational) information, whether or not the resulting form is linguistically recognized.

```
>>> import re
>>> sentence = 'Jael rushed hurtling down the stairs'
>>> tokens = sentence.split()
>>> pattern = '(s|ing|ed)$'
>>> stems = [re.sub(pattern,'',token) for token in tokens]
>>> stems
['Jael', 'rush', 'hurtl', 'down', 'the', 'stair']
```

Lemma

Linguistic Definition

Dictionary headword

Lemma

Linguistic Definition

Dictionary headword

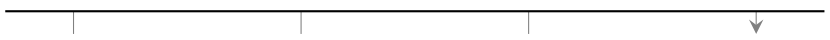
NLP Definition

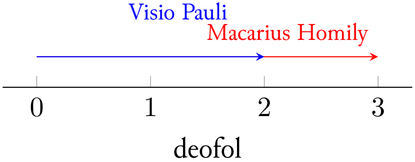
Unique identifier to which inflected forms of the same word may be assigned

Vector Space Model

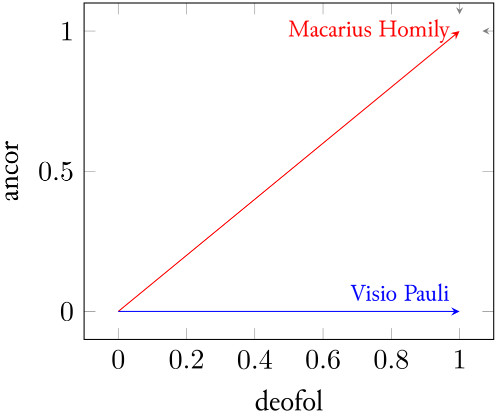
- A vector is "an ordered list of numbers, or coordinates, in a vector space" (Lane 79)
- VSM is a similarity model for comparing (e.g.) queries and documents
- If a query has three keywords, each is represented as a vector starting from (0,0,0) in a three-dimensional graph; the number of dimensions grows with each additional keyword
- Documents are matched to these keywords in the corresponding graph
- But the graph is only a visualization; we are in fact dealing with (float) numbers
- The advantage of the model is that documents (i.e. vector angles) may be compared and their similarity calculated, and document relevance is assumed to be proportional to document-query similarity
- Each **term** (here word type) is represented by its own **dimension**
- A **vector** is an index of dimensions

Example: A One-Dimensional Vector Space

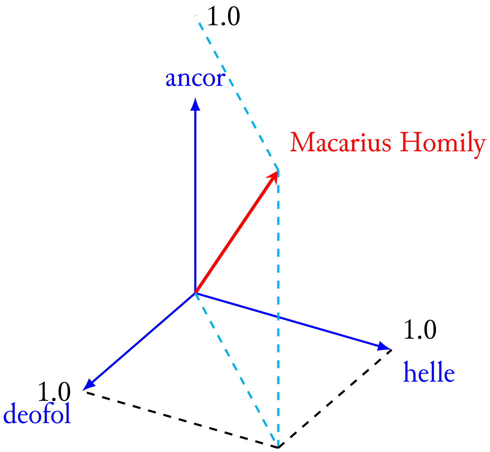




Example: A Two-Dimensional Vector Space



Example: A Three-Dimensional Vector Space



Example: A Four-Dimensional Vector Space

Given

- four dimensions `"deofol"`, `"helle"`, `"ancor"`, `"þunorrad"`
- two documents `m` = Macarius Homily, `p` = Visio Pauli

```
mHits = [1, 1, 1, 1]
pHits = [1, 1, 0, 0]
```

Dot Product

The sum of products between corresponding entries in two sequences of numbers: $\sum_{i=1}^{n} a_i b_i$

Dot Product

The sum of products between corresponding entries in two sequences of numbers: $\sum_{i=1}^{n} a_i b_i$

Thus given a query containing four terms `"deofol"`, `"helle"`, `"ancor"`, `"þunorrad"`

```
>>> import numpy
>>> query = numpy.array([1, 1, 1, 1])
>>> m = numpy.array([1, 1, 1, 1])
>>> p = numpy.array([1, 1, 0, 0])
```

the dot product between `query` and `m` would be $1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 4$, whereas `p` would come to only `2`:

```
>>> query.dot(p)
2
```

Thus the dot product allows us to calculate the overlap between bags of words.

Euclidian Space

The space model of classical geometry, irrespective of the number of dimensions used.

Zipf's Law

A word's frequency in a natural corpus $f(r)$ is inversely proportional to its rank (r) in the word frequency table.

Zipf’s Law

A word’s frequency in a natural corpus *f(r)* is inversely proportional to its rank (*r*) in the word frequency table.

$$f(r) \propto \frac{1}{(r+\beta)^{\alpha}}$$

where $\alpha \approx 1$ and $\beta \approx 2.7$

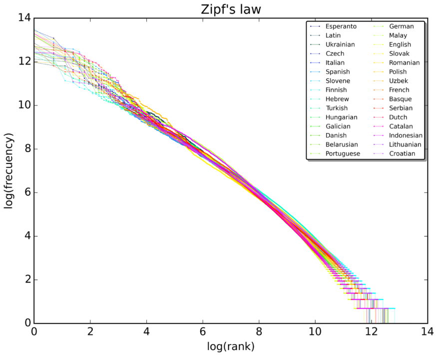


Figure 2: Frequency/rank log plot for the first 10 mln words in 30 Wikipedias (CC-BY-SA Sergio Jimenez)

Logarithm

- The logarithm of a number x is the inverse to its exponent.
- It equals the exponent by which the base must be raised to yield x, i.e. $b^y = x$
- Adding up the logarithms of two or more numbers yields the logarithm of their product. *This makes logarithmic arithmetic computationally frugal.*
- Logarithmic functions are used in NLP to represent word frequencies on a linear scale (see previous slide).
- Like exponentiation, the logarithm requires that a base b be specified, but for our purposes the chosen base doesn't matter as long as it's constant.
- The notation of the logarithm is $log_b(x)$ for antilogarithm x to base b. (Base 10 is assumed if you leave out $_b$.)

TF-IDF

- **Term frequency** is a term's (i.e. type's) frequency in a text, whether in absolute numbers or divided by its total type count ("normalized term frequency").
- **Inverse document frequency** is the ratio of total documents to the number of documents containing the term (i.e. type).
- **TF-IDF** is the product of these two numbers (but usually of their logarithms), indicating a term's statistical importance in a single document as judged by its prevalence in the corpus as a whole.
- We can speak of *statistical importance*, not just *frequency*, thanks to TF-IDF's reliance on Zipf's Law.

Cosine Similarity

The similarity between the cosines of the angles of two vectors.

- A higher similarity score indicates a closer match

Additive Smoothing

Adding a *bias* of $+1$ or some other uniform value to every input value to avoid dividing by zero and generally improve results.

Naive Bayes Classifier

A probabilistic model assigning scores for individual features without considering probabilities following from their correlation.

The model works better than one might expect, and it allows for the training of a classifier on limited training data.

Rule-Based System

A cluster of hand-written `if ... then` rules.

The default method for many NLP tasks, such as

- Lemmatization
- Named-entity recognition (NER)

Supervised Machine Learning

Training an algorithm on hand-labelled data until it correctly handles new data.

Typical applications include classification (“select all squares with traffic lights”)

Unsupervised Machine Learning

Training a classifier on unlabelled data. Common NLP applications:

- topic modelling
- word vectorizing

Perceptron

Algorithm consisting of a **single node** processing input into output and autonomously adjusting the weighting of each input after each training cycle until the actual output *just* matches the expected output.

- Supervised learning
- Binary classification (outputs `0` or `1`)
- Linearly separable data

Linearly vs Nonlinearly Separable Data

See Lane et al. pp. 164–165.

Artificial Neural Network (ANN)

Algorithm relying on **multiple nodes** processing input and autonomously adjusting the weighting of each input after each training cycle until the actual output *just* matches the expected output.

The distinction with a perceptron is that backpropagation with multiple nodes requires a nonlinear activation function (to model nonlinear relationships between input and output; see Lane et al. ch. 5).

Artificial Neural Network (ANN)

Algorithm relying on **multiple nodes** processing input and autonomously adjusting the weighting of each input after each training cycle until the actual output *just* matches the expected output.

The distinction with a perceptron is that backpropagation with multiple nodes requires a nonlinear activation function (to model nonlinear relationships between input and output; see Lane et al. ch. 5).

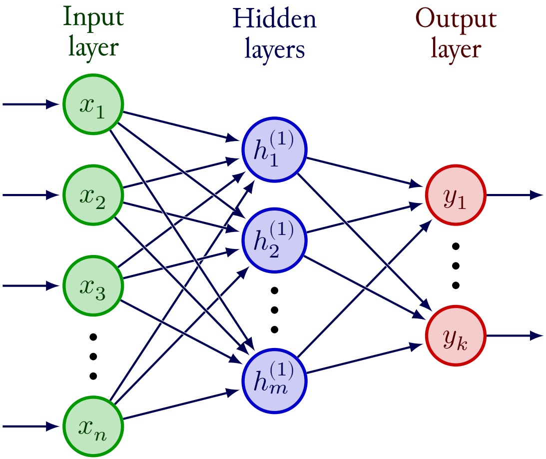
- Supervised (and unsupervised) learning
- Linearly and nonlinearly separable data
- Classification, regression analysis, clustering, filtering, etc.

Backpropagation

Backpropagation of the error calculates how much a particular weight contributed to the overall error in a training cycle (**epoch**) and adjusts the weights accordingly before running the next epoch.

In a multilayer neural network, this requires nonlinear math, because we can’t see the outputs of any but the final layer of neurons.

Deep Learning



A neural network relying on multiple (layers of) neurons, allowing nonlinear classification.

Overfitting

Training a supervised neural network so precisely on its training data that its ability to predict new data is adversely affected.

Data Separation

training data 60%	validation data 20%	test data 20%

- fitting
- selecting optimal hyperparameters
- demonstrating accuracy with new data

Shuffle your data!

Word Embedding / Word Vectorizing

Encoding words as vectors, with words with similar meaning ending up adjacent in vector space.

Arrived at by unsupervised learning, comparing the contexts of words.

Named Entity Recognition (NER)

The mechanized identification of proper names as well as quantities, time codes, etc.

Usually arrived at through rule-based algorithms, though supervised ML is possible.

Precision and Recall

Recall

How well a classifier does at assigning the accurate label: $\dfrac{TP}{TP + FN}$

Precision and Recall

Recall

How well a classifier does at assigning the accurate label: $\dfrac{TP}{TP + FN}$

Precision

How well a classifier does at foregoing assignment of an inaccurate label: $\dfrac{TP}{TP + FP}$

Precision and Recall

Recall

How well a classifier does at assigning the accurate label: $\dfrac{TP}{TP + FN}$

Precision

How well a classifier does at foregoing assignment of an inaccurate label: $\dfrac{TP}{TP + FP}$

F-Measure

The harmonic mean between the two: $\dfrac{2rp}{r + p}$

By default, all three are used of a specific label, but they can be generalized.

Bibliography

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. Sebastopol, CA: O'Reilly, 2009. https://www.nltk.org/book/.

Eisenstein, Jacob. *Introduction to Natural Language Processing*. Cambridge, MA: MIT Press, 2019.

Jurafsky, Dan, and James H. Martin. *Speech and Language Processing*. 3rd ed. draft., 2021. http://web.stanford.edu/~jurafsky/slp3/.

Lane, Hobson, Cole Howard, and Hannes Hapke. *Natural Language Processing in Action: Understanding, Analyzing, and Generating Text with Python*. Shelter Island, NY: Manning, 2019.

Matthes, Eric. *Python Crash Course*. 2nd ed. San Francisco, CA: No Starch, 2019.

Python Software Foundation. "Python," October 4, 2020. https://www.python.org/.

Vasiliev, Yuli. *Natural Language Processing Using Python and spaCy: A Practical Introduction*. San Francisco: No Starch Press, 2020.