# pattern

http://www.clips.ua.ac.be/pages/pattern

# pattern

http://www.clips.ua.ac.be/pages/pattern

# pattern

## Python programming language

http://www.python.org

http://learnpythonthehardway.org/book/

## getting started

- download
- put the pattern folder inside the zip in:

  `c:\python25\Lib\site-packages\`

  `/Library/Python/2.5/site-packages/`
- editor:

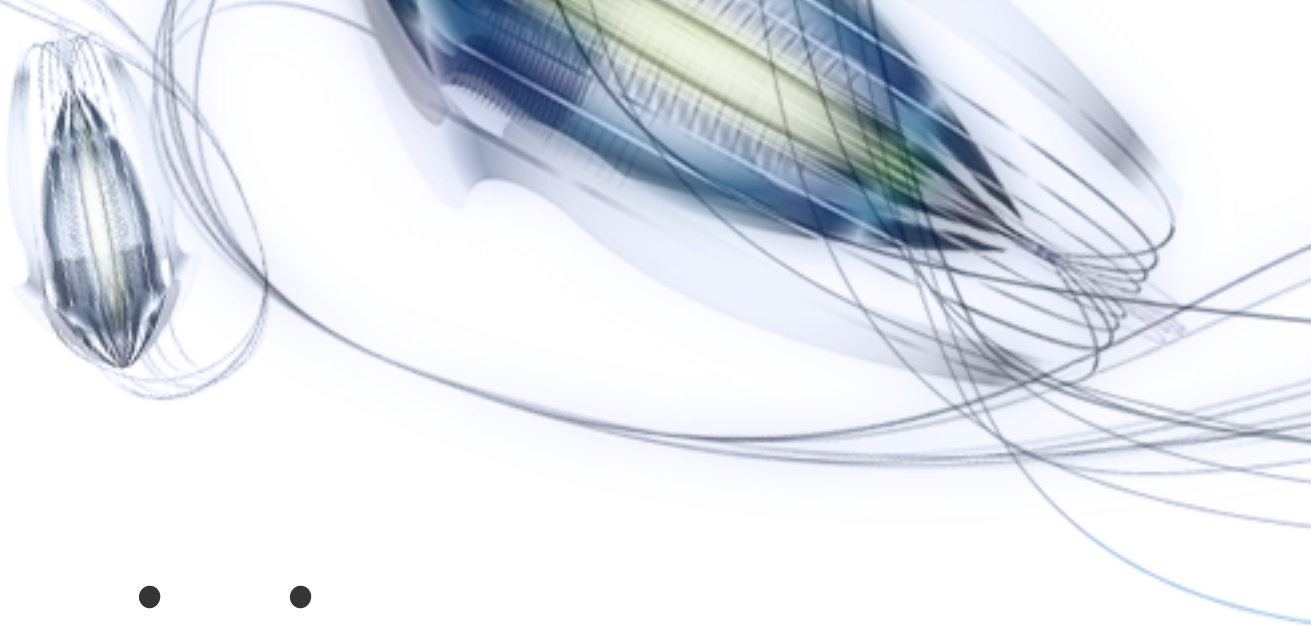
  TextMate

  TextWrangler

  Programmer's Notepad

  IDLE
- create file test.py

```
from pattern.web import URL
print URL("http://google.com").download()
```



# mining

# search engine

= officially allowed to hammer a web service


(paid)


(paid)





# search engine

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-web

```
from pattern.web import Google, Yahoo, Bing, Twitter, Wikipedia, Flickr, Newsfeed

engine = Bing(language="en")

for result in engine.search("pizza"):
    print result.title
    print result.description
    print result.url
    print result.date
    print result.author
```

# search engine

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-web

```
from pattern.web import Google, Yahoo, Bing, Twitter, Wikipedia, Flickr, Newsfeed

engine = Twitter(language="it")

for result in engine.search("pizza"):
    print result.title
    print result.description
    print result.url
    print result.date
    print result.author
```

# search engine

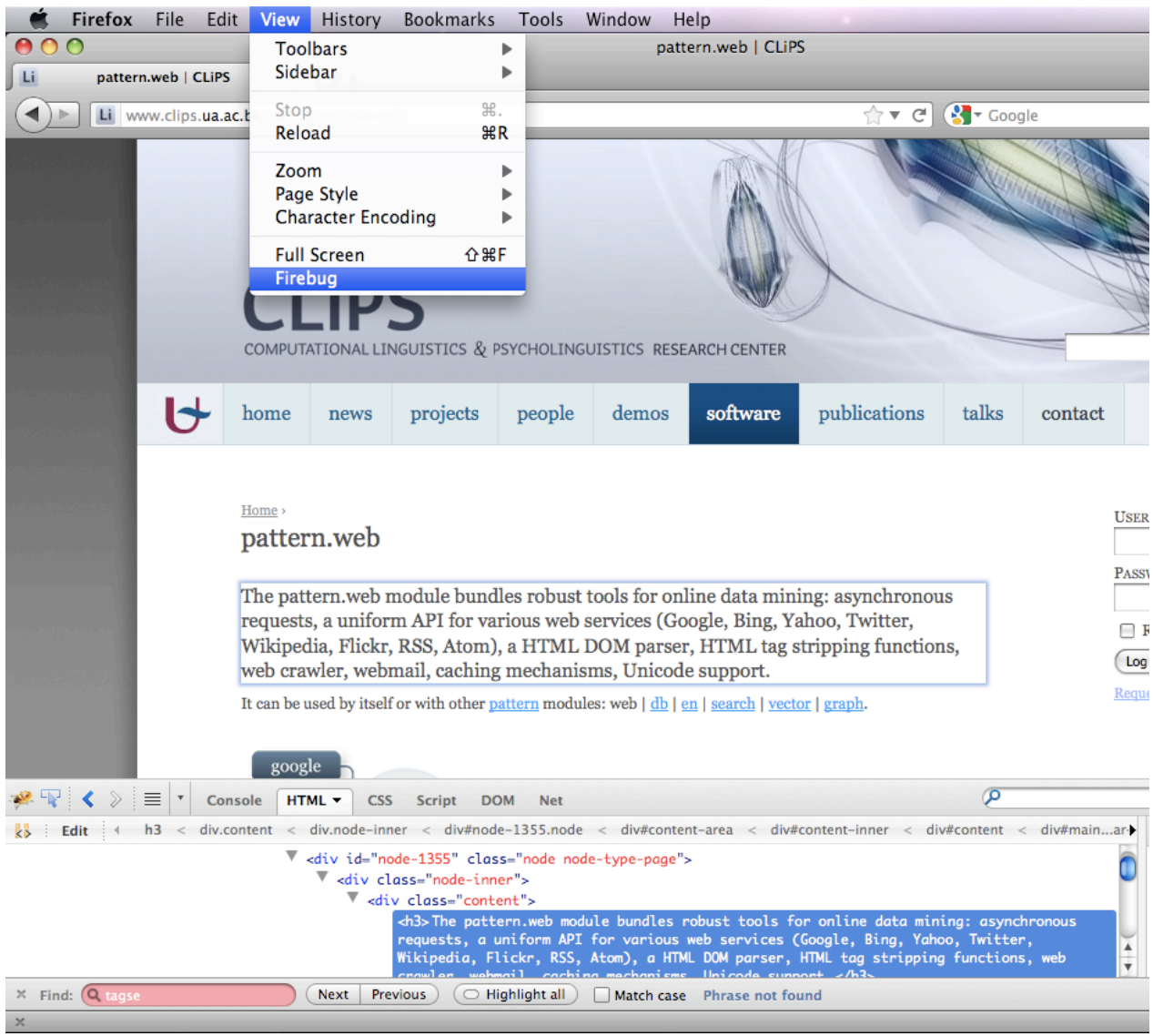
## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-web

```
from pattern.web import Google, Yahoo, Bing, Twitter, Wikipedia, Flickr, Newsfeed

engine = Wikipedia(language="en")

article = engine.search("pizza")

print article.title
print

for section in article.sections:
    print section.title.upper()
    print section.content
    print
```

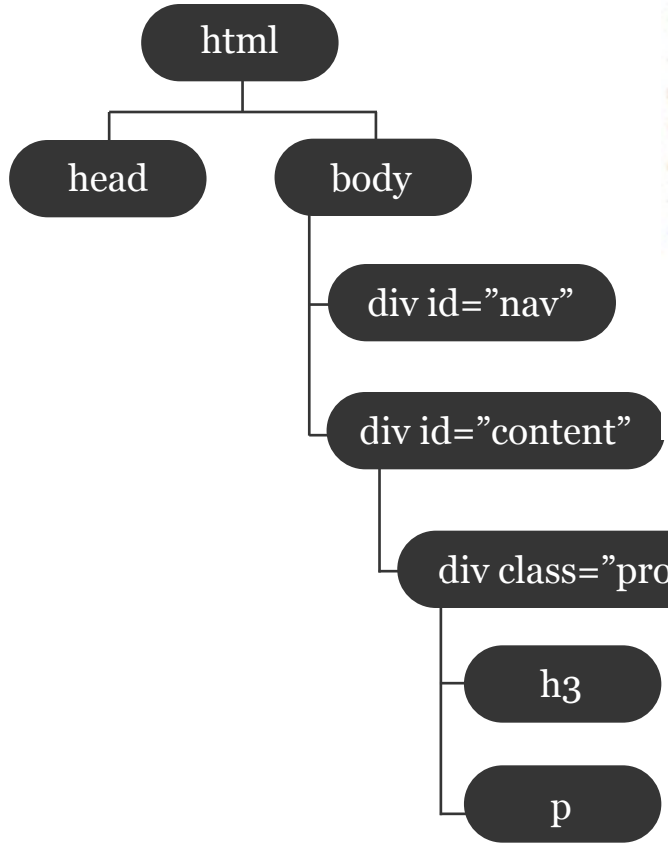
# document object model

## HTML as a tree of nested elements



# document object model

## HTML as a tree of nested elements



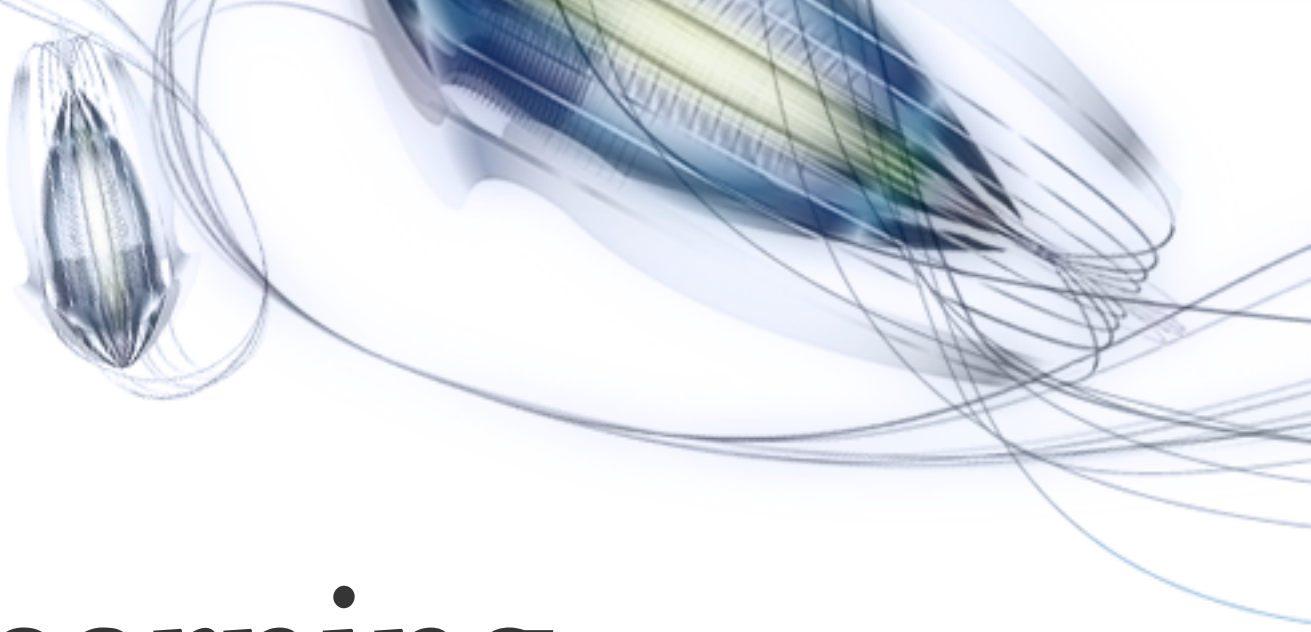

```
from pattern.web import URL, Document

url  = URL("http://somecellphonewebshop.com/products/")
html = url.download()
dom  = Document(html)

for product in dom.by_class("product"):
    title = product.by_tag("h3")[0]
    review = product.by_tag("p")[0]
    print title.content
    print review.content
    print
```

# document object model

## HTML as a tree of nested elements

`Document` is a special `Element`
`Element` can contain other `Element` objects

| | |
|---|---|
| `Element.by_id(str)` | first nested element with given id |
| `Element.by_tag(str)` | list of nested elements with given tag name |
| `Element.by_class(str)` | list of nested elements with given CSS class |
| `Element.by_attribute(attribute=value)` | list of nested elements with given attribute & value |



# learning

# corpus (one *corpus*, many *corpora*)

a collection of documents

# corpus

## a collection of documents

format

*a corpus of texts*
*a corpus of images*

# corpus

## a collection of documents

format

*a corpus of texts*
*a corpus of images*

content

*a corpus of Twitter messages*
*a corpus of images of cats*

# corpus

a collection of documents

format

*a corpus of texts*
*a corpus of images*

content

*a corpus of Twitter messages*
*a corpus of images of cats*

annotation

more interesting if it is **annotated** (“tagged”)

*a corpus of Twitter messages tagged by language*
*a corpus of images of cats tagged by color*

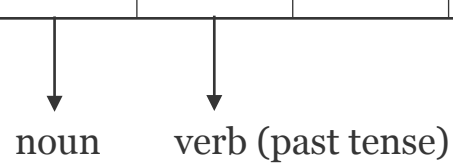
# corpus

FOR EXAMPLE

Brown corpus (1969)

- English texts annotated with a **class** SPORT MUSIC ...
- one million words
- each word is annotated with a part-of-speech tag

| The | cat | sat | on | the | mat | . |
|---|---|---|---|---|---|---|
| **DT** | **NN** | **VBD** | **IN** | **DT** | **NN** | **.** |



# corpus

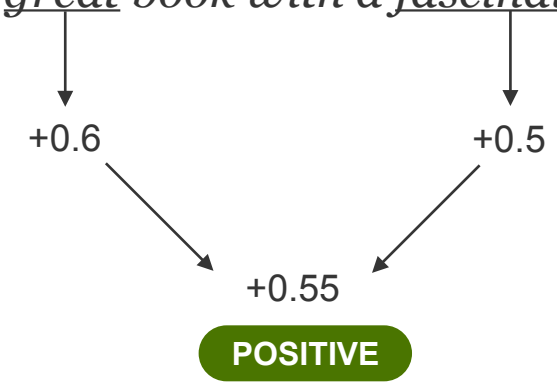
FOR EXAMPLE

PATTERN sentiment lexicon

- 5,000 adjectives mined from online book reviews
- each adjective is tagged with polarity: -1.0 → +1.0

POSITIVE NEGATIVE

*"This was a great book with a fascinating plot twist!"*

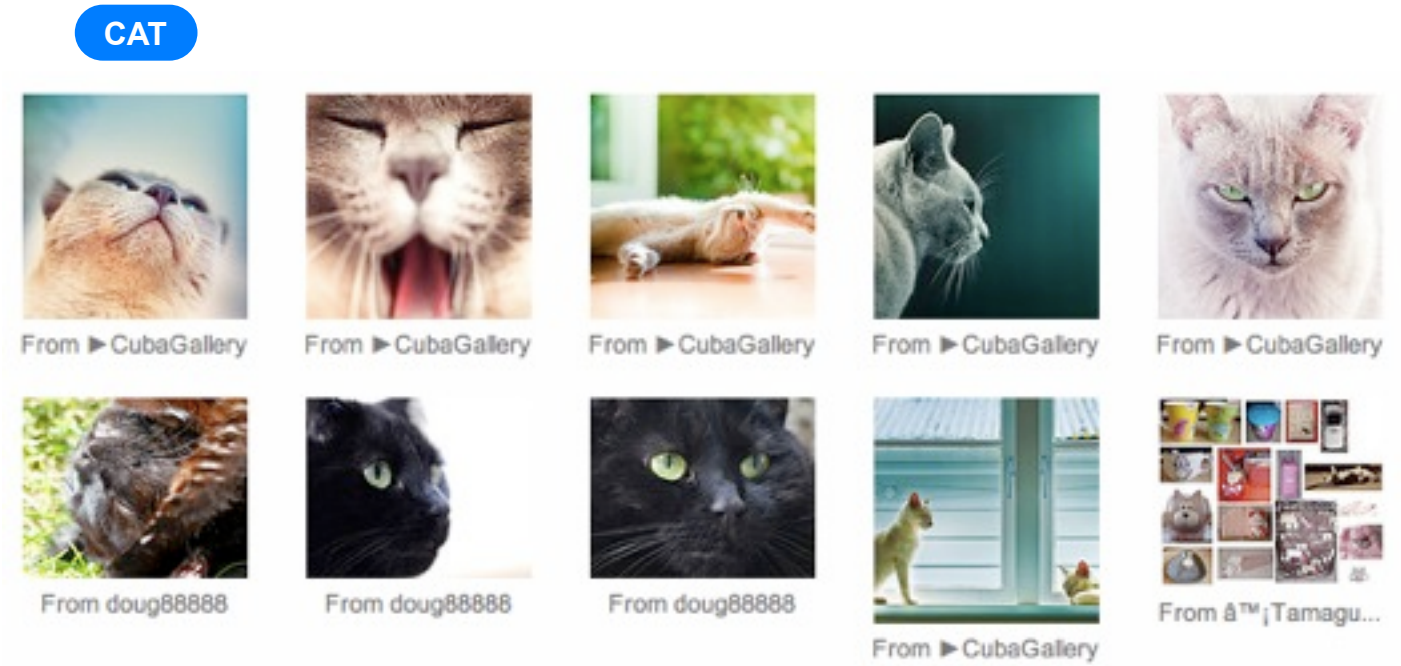


# corpus

FOR EXAMPLE

Flickr (flickr.com)

- images annotated with free tags
- images organized into categories
- it can be mined with (for example) PATTERN



# corpus

the web is a corpus

# corpus

## the web is a corpus

unfortunately, it is badly formatted


unfortunately, it is badly annotated

# corpus

## the web is a corpus

PATTERN was made to help you
mine the WWW corpus

# corpus

annotated corpora are useful for statistical prediction

# corpus

## annotated corpora are useful for statistical prediction

CAN YOU ANSWER THESE QUESTIONS?

- *Does English have more verbs for describing sports or for music?*
- *What are the best Flickr images to show when a user searches for "cat"?*
- *Are tweets about Berlusconi more positive or more negative?*
- *Did James Joyce suffer from Alzheimer's disease?*
- *Are we heading to a revolution, judging by current newspaper articles?*

# corpus

## annotated corpora are useful for statistical prediction

DO YOU KNOW THE STRANGE FIGURE IN THE MIDDLE?

# corpus

a collection of documents

# document

## a unit of data in a corpus

- word: adjective, named entity, ...
- sentence: tweet, IM, SMS, ...
- paragraph: product description, customer review, ...
- text: e-mail, webpage, book, ...
- image
- sound

TAGS

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

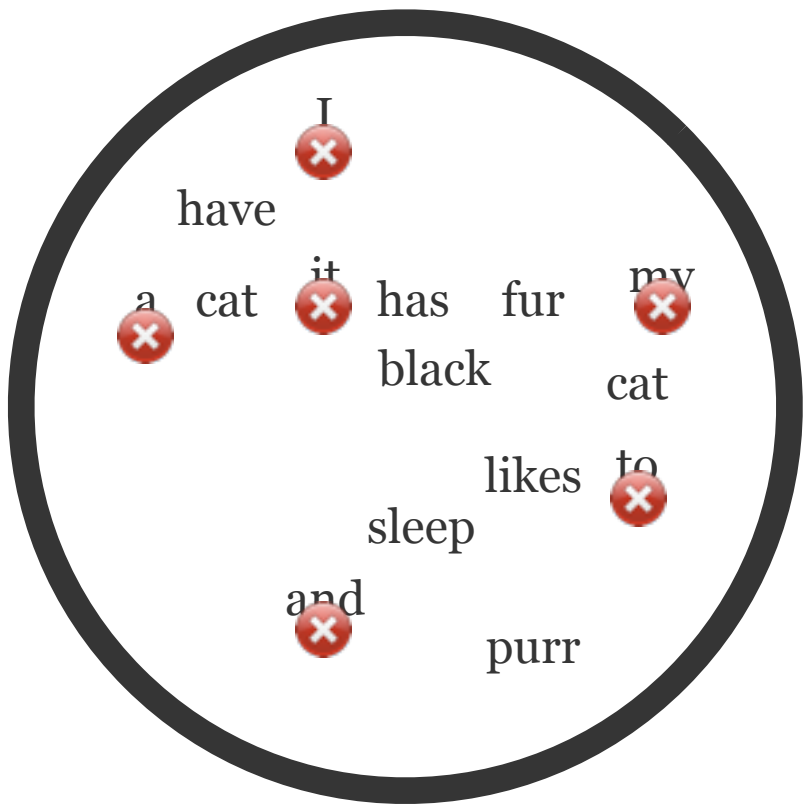
=

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=




filter

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=


filter

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=


filter

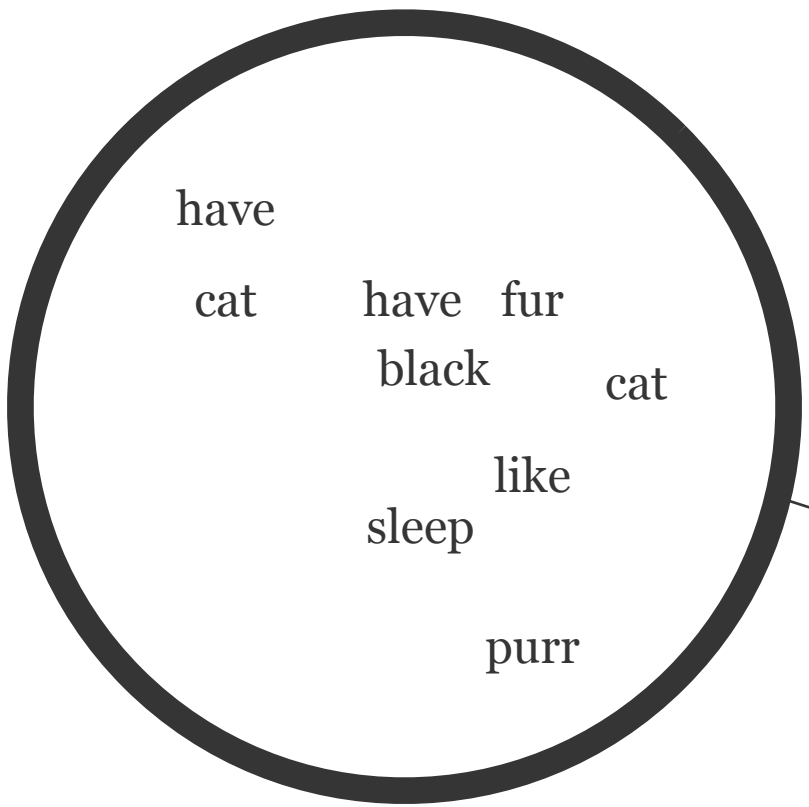
normalize

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=




filter

normalize

count

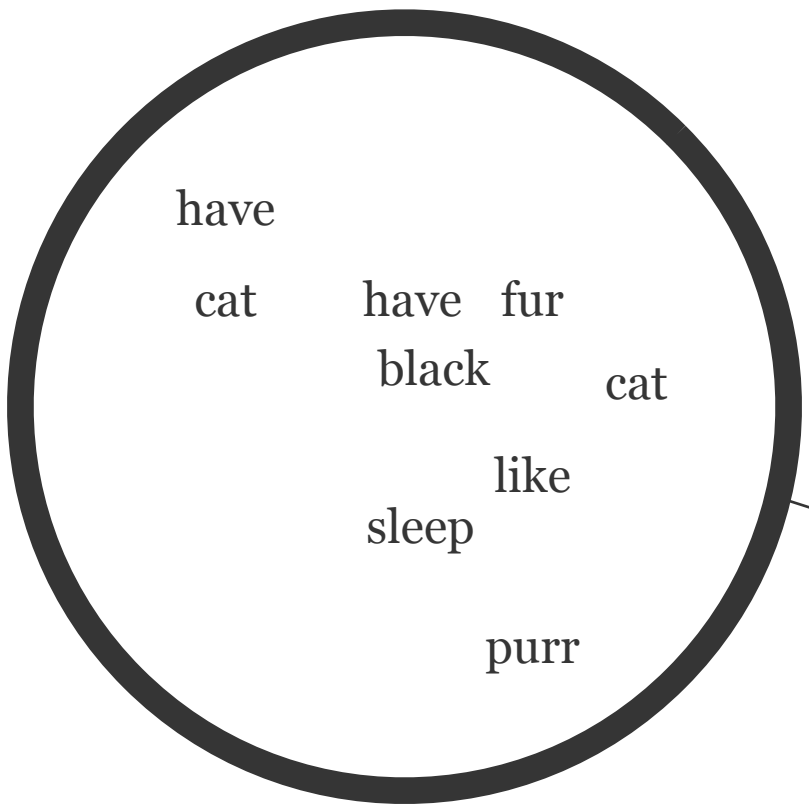
| black | cat | fur | have | like | purr | sleep |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 1 | 1 |

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=




filter

normalize

count

| black | cat | fur | have | like | purr | sleep |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 1 | 1 |

annotate

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=


filter

normalize

count

| black | cat | fur | have | like | purr | sleep |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 1 | 1 |

annotate

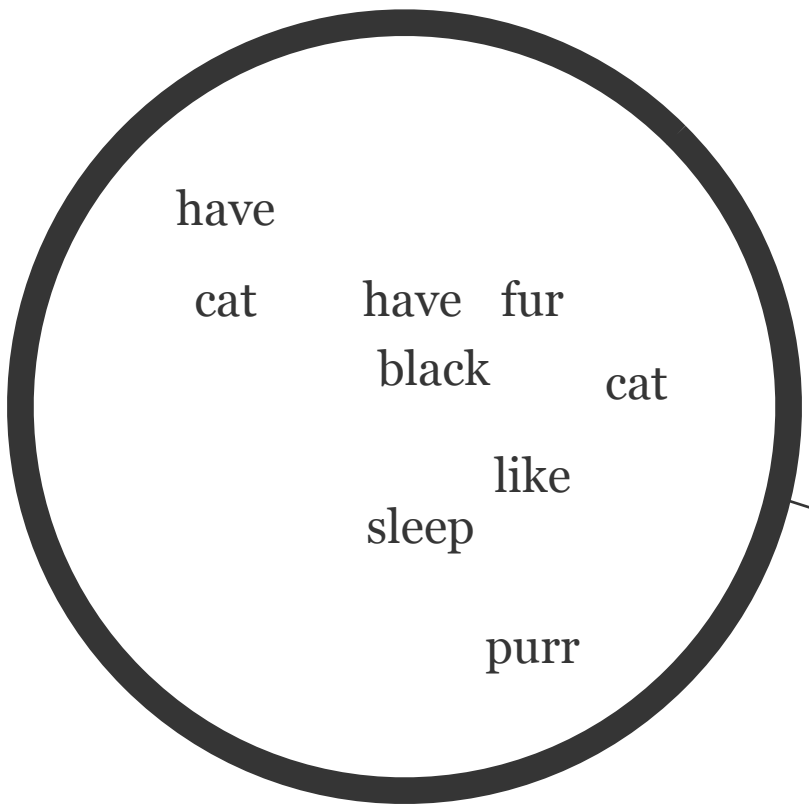
TOM’S CAT

# document

## a bag of words

“I have a cat. It has black fur. My cat likes to sleep and purr.”

=




filter

normalize

count

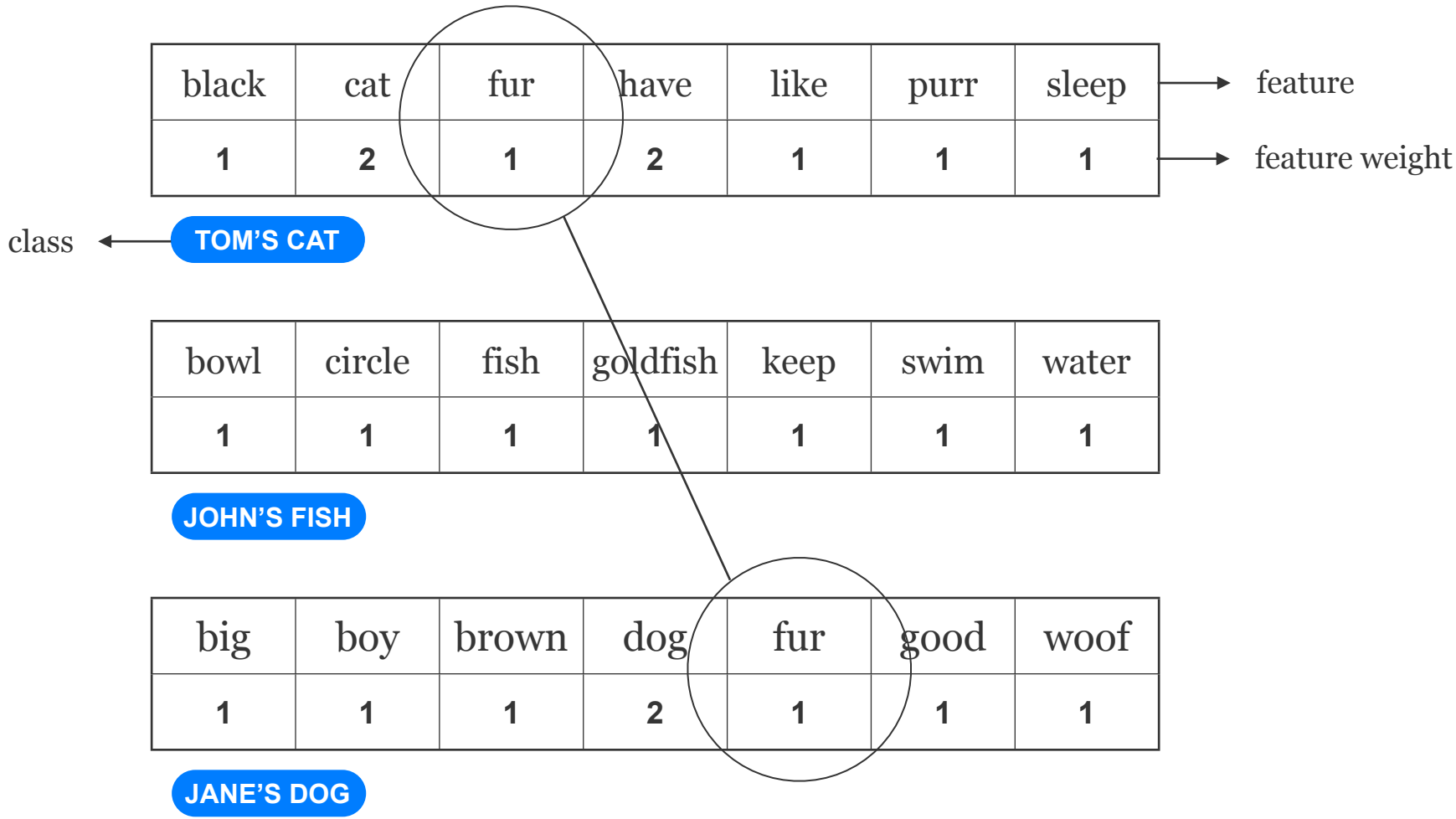
this is called the **document vector**

| black | cat | fur | have | like | purr | sleep |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 1 | 1 |

annotate

TOM’S CAT

# document vector




CAN YOU SEE WHICH PETS ARE MORE SIMILAR?

# vector space

corpus

**TOM'S CAT**

| black | cat | fur | have | like | purr | sleep |
|---|---|---|---|---|---|---|
| **1** | **2** | **1** | **2** | **1** | **1** | **1** |

**JOHN'S FISH**

| bowl | circle | fish | goldfish | keep | swim | water |
|---|---|---|---|---|---|---|
| **1** | **1** | **1** | **1** | **1** | **1** | **1** |

**JANE'S DOG**

| big | boy | brown | dog | fur | good | woof |
|---|---|---|---|---|---|---|
| **1** | **1** | **1** | **2** | **1** | **1** | **1** |

| VECTOR SPACE | TOM'S CAT | JOHN'S FISH | JANE'S DOG |
|---|---|---|---|
| BIG | 0 | 0 | 1 |
| BLACK | 1 | 0 | 0 |
| BOWL | 0 | 1 | 0 |
| BOY | 0 | 0 | 1 |
| BROWN | 0 | 0 | 1 |
| CAT | 2 | 0 | 0 |
| CIRCLE | 0 | 1 | 0 |
| DOG | 0 | 0 | 2 |
| FISH | 0 | 1 | 0 |
| FUR | 1 | 0 | 1 |
| GOLDFISH | 0 | 1 | 0 |
| GOOD | 0 | 0 | 1 |
| HAVE | 2 | 0 | 0 |
| KEEP | 0 | 1 | 0 |
| LIKE | 1 | 0 | 0 |
| PURR | 1 | 0 | 0 |
| SLEEP | 1 | 0 | 0 |
| SWIM | 0 | 1 | 0 |
| WOOF | 0 | 0 | 1 |

# vector space

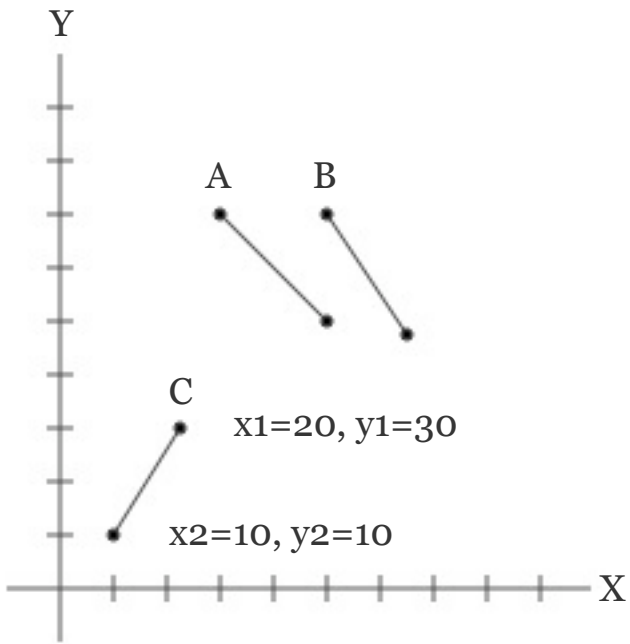
## similarity

by doing calculations on the matrix
we can compute which classes of documents
are more similar to each other

| VECTOR SPACE | TOM'S CAT | JOHN'S FISH | JANE'S DOG |
|---|---|---|---|
| BIG | 0 | 0 | 1 |
| BLACK | 1 | 0 | 0 |
| BOWL | 0 | 1 | 0 |
| BOY | 0 | 0 | 1 |
| BROWN | 0 | 0 | 1 |
| CAT | 2 | 0 | 0 |
| CIRCLE | 0 | 1 | 0 |
| DOG | 0 | 0 | 2 |
| FISH | 0 | 1 | 0 |
| FUR | 1 | 0 | 1 |
| GOLDFISH | 0 | 1 | 0 |
| GOOD | 0 | 0 | 1 |
| HAVE | 2 | 0 | 0 |
| KEEP | 0 | 1 | 0 |
| LIKE | 1 | 0 | 0 |
| PURR | 1 | 0 | 0 |
| SLEEP | 1 | 0 | 0 |
| SWIM | 0 | 1 | 0 |
| WOOF | 0 | 0 | 1 |

# vector space

## what “calculations”?



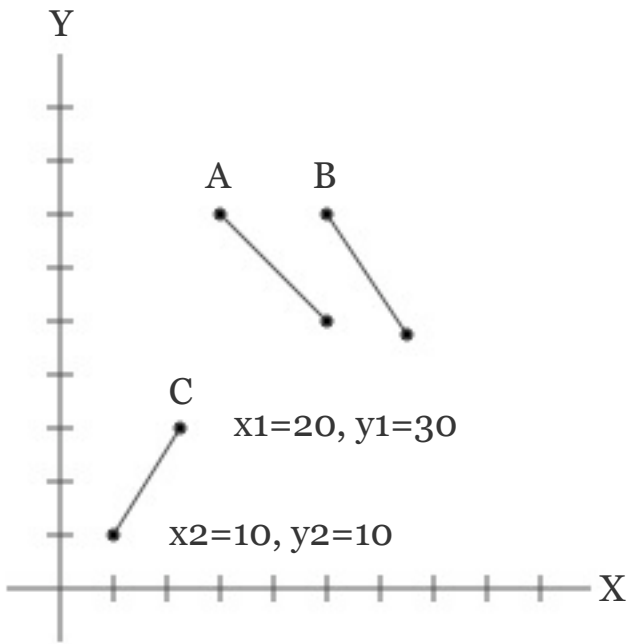

- A and B are more similar
- they point in the same direction
- they have a similar **angle**

- A, B and C are vectors!
- in 2D space

CAN YOU SEE WHICH LINES ARE MORE SIMILAR?

# vector space

## what “calculations”?




CAN YOU SEE WHICH LINES ARE MORE SIMILAR?

- A and B are more similar
- they point in the same direction
- they have a similar **angle**

- A, B and C are vectors!
- in 2D space

| VECTOR SPACE | A | B | C |
|---|---|---|---|
| X1 | 30 | 50 | 20 |
| Y1 | 70 | 70 | 30 |
| X2 | 50 | 65 | 10 |
| Y2 | 50 | 45 | 10 |

# vector space

## similarity

we can calculate angles for vectors
in 2 dimensions ... in 3 dimensions ...
and in *n* dimensions
= **cosine similarity** (remember *cos* and *sin*?)

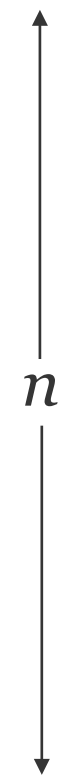
| VECTOR SPACE | TOM'S CAT | JOHN'S FISH | JANE'S DOG |
|---|---|---|---|
| BIG | 0 | 0 | 1 |
| BLACK | 1 | 0 | 0 |
| BOWL | 0 | 1 | 0 |
| BOY | 0 | 0 | 1 |
| BROWN | 0 | 0 | 1 |
| CAT | 2 | 0 | 0 |
| CIRCLE | 0 | 1 | 0 |
| DOG | 0 | 0 | 2 |
| FISH | 0 | 1 | 0 |
| FUR | 1 | 0 | 1 |
| GOLDFISH | 0 | 1 | 0 |
| GOOD | 0 | 0 | 1 |
| HAVE | 2 | 0 | 0 |
| KEEP | 0 | 1 | 0 |
| LIKE | 1 | 0 | 0 |
| PURR | 1 | 0 | 0 |
| SLEEP | 1 | 0 | 0 |
| SWIM | 0 | 1 | 0 |
| WOOF | 0 | 0 | 1 |

# vector space

## similarity

- corpus = collection of documents
- document = class tag + vector of wordcount features
- cosine similarity = angle between two n-dimensional vectors (0.0 → 1.0)
- highest similarity: **nearest neighbors**

| VECTOR SPACE | TOM'S CAT | JOHN'S FISH | JANE'S DOG |
|---|---|---|---|
| BIG | 0 | 0 | 1 |
| BLACK | 1 | 0 | 0 |
| BOWL | 0 | 1 | 0 |
| BOY | 0 | 0 | 1 |
| BROWN | 0 | 0 | 1 |
| CAT | 2 | 0 | 0 |
| CIRCLE | 0 | 1 | 0 |
| DOG | 0 | 0 | 2 |
| FISH | 0 | 1 | 0 |
| FUR | 1 | 0 | 1 |
| GOLDFISH | 0 | 1 | 0 |
| GOOD | 0 | 0 | 1 |
| HAVE | 2 | 0 | 0 |
| KEEP | 0 | 1 | 0 |
| LIKE | 1 | 0 | 0 |
| PURR | 1 | 0 | 0 |
| SLEEP | 1 | 0 | 0 |
| SWIM | 0 | 1 | 0 |
| WOOF | 0 | 0 | 1 |



# vector space search

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-vector

```
from pattern.vector import Document, Corpus, LEMMA

d1 = Document("I have a cat. It has black fur. My cat likes to sleep and purr.", type="Tom's cat", stemmer=LEMMA)
d2 = Document("I keep a goldfish in a bowl of water. It swims in circles.", type="John's fish", stemmer=LEMMA)
d3 = Document("My big dog Woof has a brown fur. He's a good boy.", type="Jane's dog", stemmer=LEMMA)

print d1.type
print d1.vector
print

corpus = Corpus(documents=[d1, d2, d3])

print corpus.features
print corpus.similarity(d1, d2)
print corpus.similarity(d1, d3)
print

print corpus.neighbors(d1, top=10)
print

print corpus.search("water")
```

# classifier

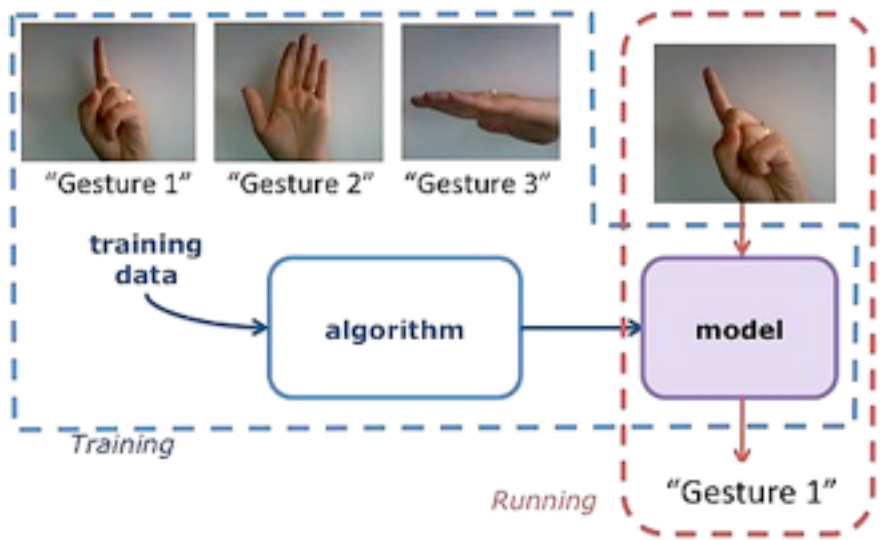
many documents can have the same class (cat, fish, dog, ...)

useful to predict the class of an unknown document:
inherit the best class of its nearest neighbors

```
from pattern.vector import Document, Corpus, kNN

d1 = Document("The cat purrs happily and wags its tail.", type="cat")
d2 = Document("The cat sinks its claws into the sofa.", type="cat")
d3 = Document("Cats have whiskers, claws, a tail and pointy ears.", type="cat")

d4 = Document("The dog barks at the burglar", type="dog")
d5 = Document("The dog pants and wags its tail at his master.", type="dog")
d6 = Document("Dogs like to go out for a walk.", type="dog")

corpus = Corpus(documents=[d1, d2, d3, d4, d5, d6])

classifier = kNN()
for document in corpus:
    classifier.train(document)

print classifier.classify("The tiger sinks its claws into his prey")

#print corpus.neighbors(Document("The tiger sinks its claws into his prey"))
```

# classifier

learning from a training corpus
= **supervised machine learning**




Rock Scissors Paper
http://www.nytimes.com/interactive/science/rock-paper-scissors.html

# clustering

## “my documents are unlabeled!” (no class)

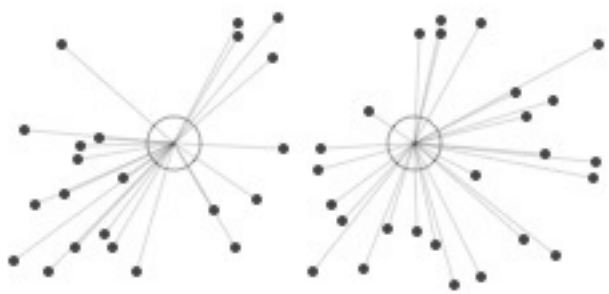
we can compute classes automatically
by creating groups of similar documents

RANDOM POINTS IN 2D

# clustering

## “my documents are unlabeled!” (no class)

we can compute classes automatically
by creating groups of similar documents



POINTS BY DISTANCE TO CENTROID

# clustering

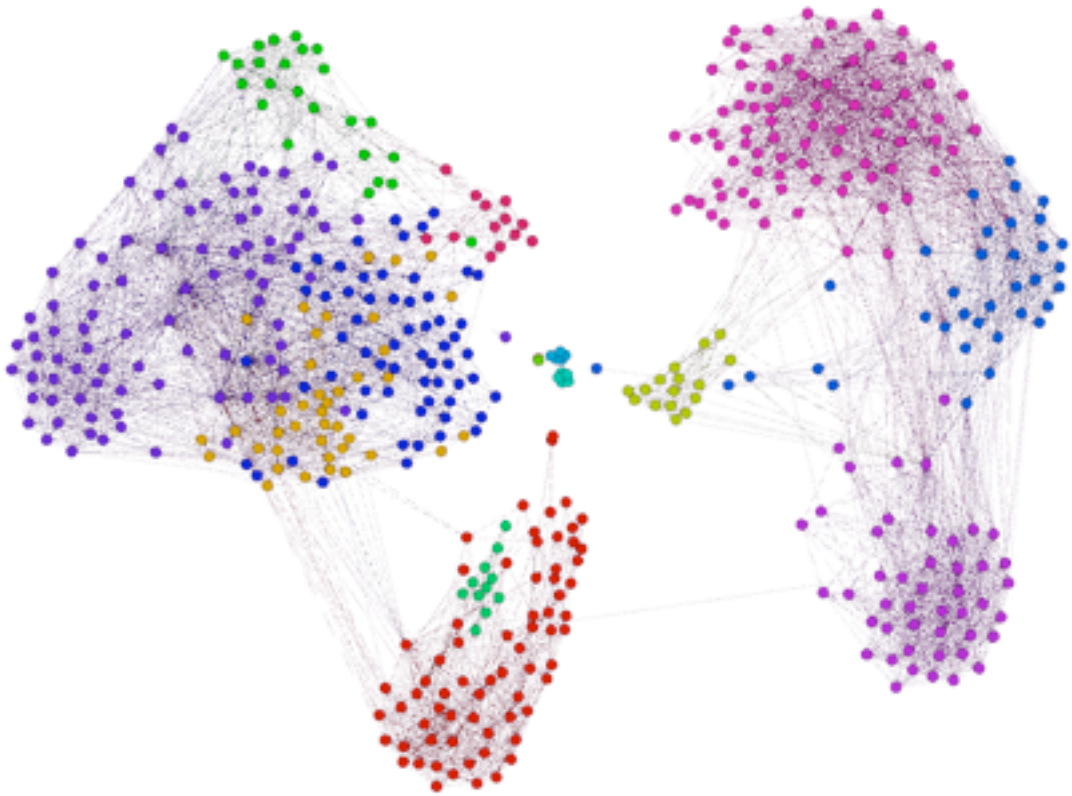
“my documents are unlabeled!” (no class)

we can compute classes automatically
by creating groups of similar documents

```
from pattern.vector import Document, Corpus, HIERARCHICAL

d1 = Document("Cats are independent pets.")
d2 = Document("Dogs are trustworthy pets.")
d3 = Document("Boxes are made of cardboard.")

corpus = Corpus(documents=[d1, d2, d3])

tree = corpus.cluster(method=HIERARCHICAL)

print tree[0]
print tree[1]
```

# clustering

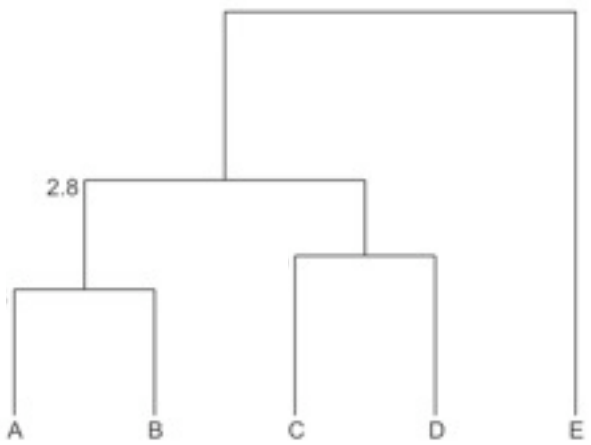
difficult + slow



# clustering

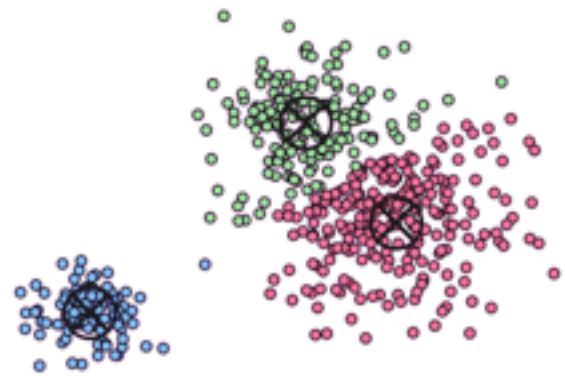
## hierarchical

find the two nearest vectors = cluster1
calculate new mean vector for cluster1
…




## *k*-means

random centers
attach each vector to nearest center
swap vectors to “tighten” the centers



# clustering

learning by statistical estimation
= **unsupervised machine learning**

# feature selection

document = class tag + vector of wordcount features

better features = better similarity = better learning
things to try:

- lemmatize words (wasn't = be)
- use part-of-speech tagging to filter specific types of words (nouns, verbs)
- use domain-specific whitelist / blacklist (research, engineer, journal, ...)
- ?
- *tf-idf*

# feature selection

## term frequency - inverse document frequency

“today was an important day”
“it is important to pay attention”
“I have something important to announce”
…

“important” ≠ important

# feature selection

## term frequency - inverse document frequency

**tf** = relative wordcount
every word is equally important

| today | was | a | very | important | day |
|---|---|---|---|---|---|
| 0.167 | 0.167 | 0.167 | 0.167 | 0.167 | 0.167 |

| it | is | important | to | pay | attention |
|---|---|---|---|---|---|
| 0.167 | 0.167 | 0.167 | 0.167 | 0.167 | 0.167 |

| I | have | something | important | to | announce |
|---|---|---|---|---|---|
| 0.167 | 0.167 | 0.167 | 0.167 | 0.167 | 0.167 |

# feature selection

## term frequency - inverse document frequency

**tf / idf** = word *relevancy*
words that occur in many documents are less important

| today | was | a | very | important | day |
|---|---|---|---|---|---|
| 0.183 | 0.183 | 0.183 | 0.183 | **0.167** | 0.183 |

| it | is | important | to | pay | attention |
|---|---|---|---|---|---|
| 0.183 | 0.183 | **0.167** | 0.183 | 0.183 | 0.183 |

| I | have | something | important | to | announce |
|---|---|---|---|---|---|
| 0.183 | 0.183 | 0.183 | **0.167** | 0.183 | 0.183 |

words that occur only in one document gain relevancy
= document keywords

# feature selection

## for classifiers & clustering: test it

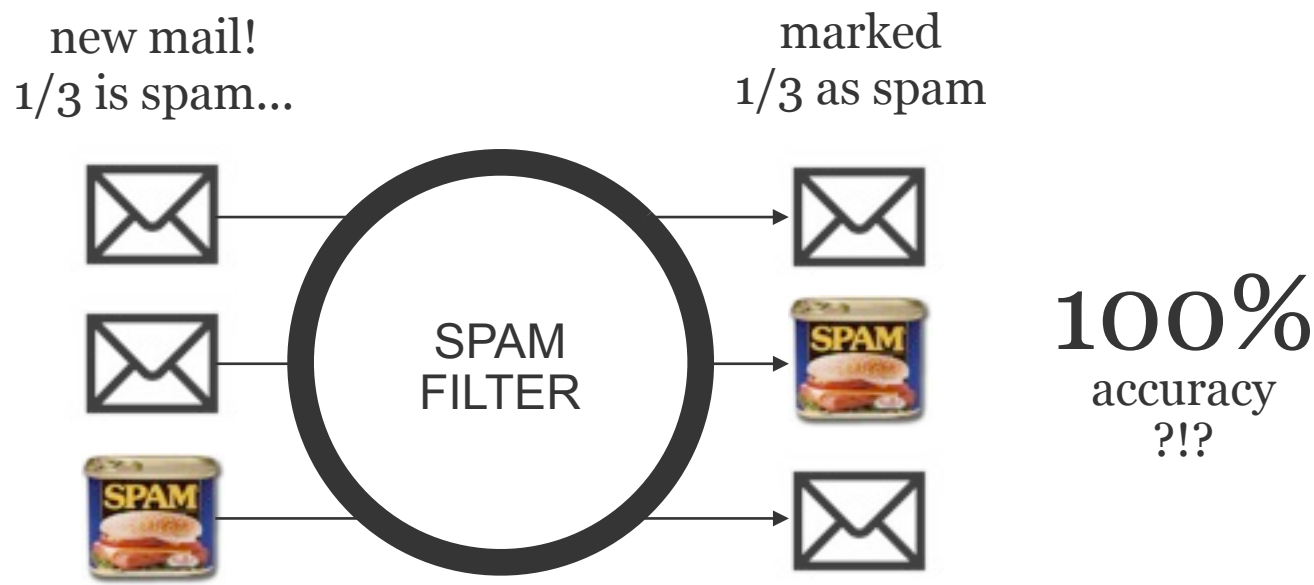
do not rely on intuition, rely on the statistical results

# testing

## accuracy, precision and recall

http://www.clips.ua.ac.be/pages/pattern-metrics

- accuracy: % correct predictions
- precision: % correct *positive* predictions
- recall: % of positive cases correctly predicted as positive

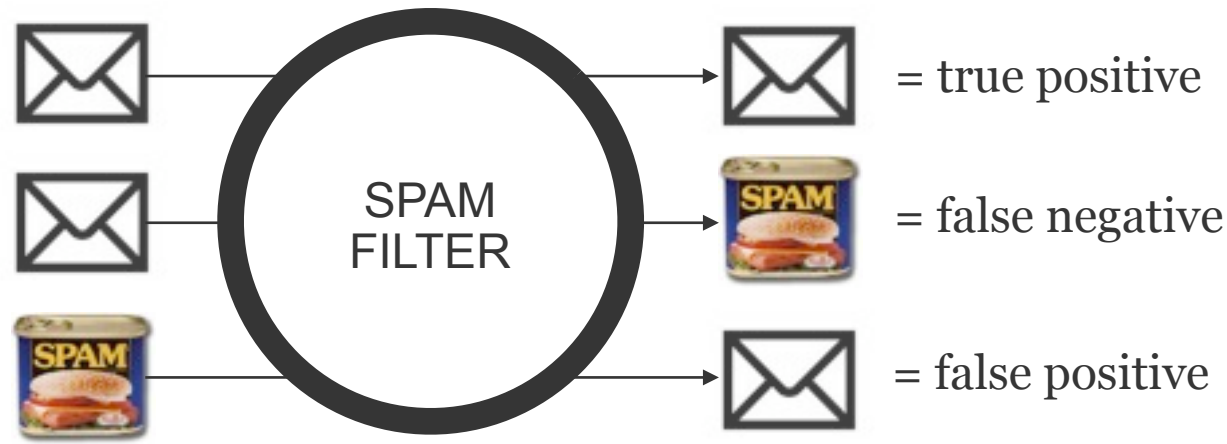


# testing

## accuracy, precision and recall

http://www.clips.ua.ac.be/pages/pattern-metrics

- accuracy: % correct predictions
- precision: % correct *positive* predictions
- recall: % of positive cases correctly predicted as positive



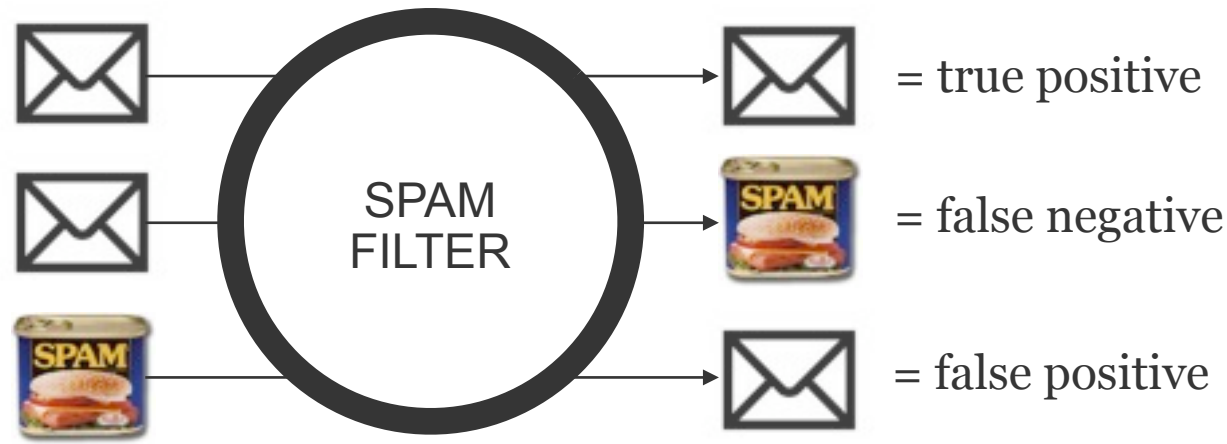
# testing

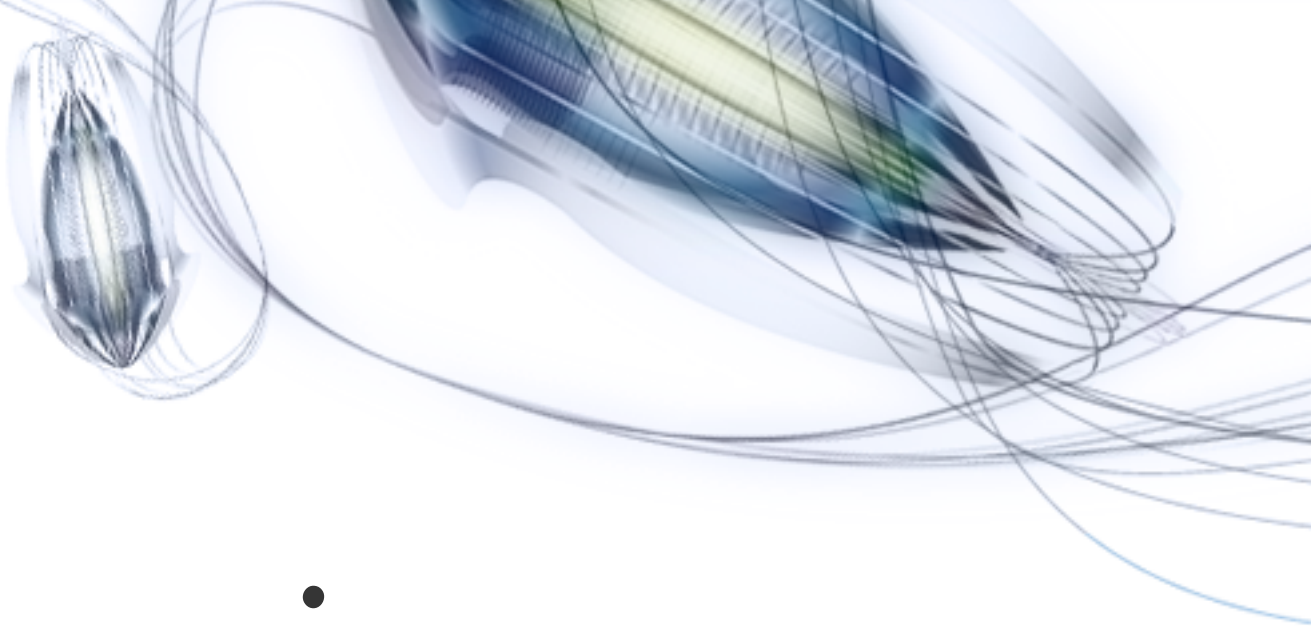
## accuracy, precision and recall

http://www.clips.ua.ac.be/pages/pattern-metrics

- accuracy: % correct predictions
- precision: % correct *positive* predictions
- recall: % of positive cases correctly predicted as positive

precision **50%**
recall **50%** ... as useful as throwing dice!





# parsing

# parser

annotates a sentence with grammatical structure

# parser

annotates a sentence with grammatical structure

tokenizer

identify words & sentence breaks (abbreviations? citations?)

# parser

annotates a sentence with grammatical structure

tokenizer

identify words & sentence breaks (abbreviations? citations?)

part-of-speech tagger

identify nouns, verbs, adjectives, ...

# parser

annotates a sentence with grammatical structure

**tokenizer**

identify words & sentence breaks (abbreviations? citations?)

**part-of-speech tagger**

identify nouns, verbs, adjectives, ...

**chunker**

identify “phrases” (“the black cat” = 1 chunk)

# parser

annotates a sentence with grammatical structure

**tokenizer**

identify words & sentence breaks (abbreviations? citations?)

**part-of-speech tagger**

identify nouns, verbs, adjectives, ...

**chunker**

identify “phrases” (“the black cat” = 1 chunk)

**lemmatizer**

normalize words (was = be, cats = cat)

# parser

FOR EXAMPLE

"The black cat sat on the mat."

**TOKENIZER** The black cat sat on the mat . **<break>**

**TAGGER** The black cat sat on the mat . **<break>**

DT JJ NN VBD IN DT NN

**CHUNKER**

NP VP PP NP

PNP

**LEMMATIZER** sit

The tags are from the Penn Treebank tagset (learn them):
http://www.clips.ua.ac.be/pages/MBSP-tags

# parser

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-en

```
from pattern.en import parse, Sentence

s = "The black cat sat on the mat."

print parse(s)
print

s = Sentence(parse(s, lemmata=True))

print s.words
print [word.lemma for word in s.words]
print

print s.chunks
print s.pnp
print

for word, tag in s.tagged:
    print word, tag
```

# parser

## “your parser makes mistakes!”

- building a parser is a supervised machine learning task
- the tagger is trained on Brown corpus
- the tagger predicts a statistically probable word tag
- ambiguity:
  *“Can I drink a can of soft drink?”*
- 95% accurate = 5/100 words are wrong

# parser

## “why should I use it?”

CAN YOU ANSWER THESE QUESTIONS?

- *What are the most frequent adjectives used in online movie reviews?*
- *How can I mine “X is cooler/better/nicer than Y” phrases from Twitter?*
- *How can my robot find my phone when I say: “my phone is on the table”?*

# parser

## “why should I use it?”

NOW THAT YOU KNOW ABOUT PARSERS AND CLASSIFIERS,
CAN YOU GUESS HOW SIRI WORKS?


Matthew is my brother

family → reinforcement learning: *“do you want me to…”*

possessive pronoun

unknown noun = person?

# search patterns

# search pattern

## a mix of words and tags

**NP** be cooler than **NP**

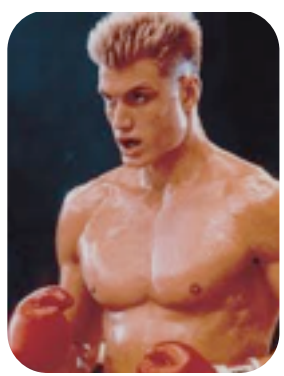
=

“Arnold Schwarzenegger is cooler than Dolph Lundgren”

“the Romans were cooler than the Celts”

“cats are cooler than dogs”

...



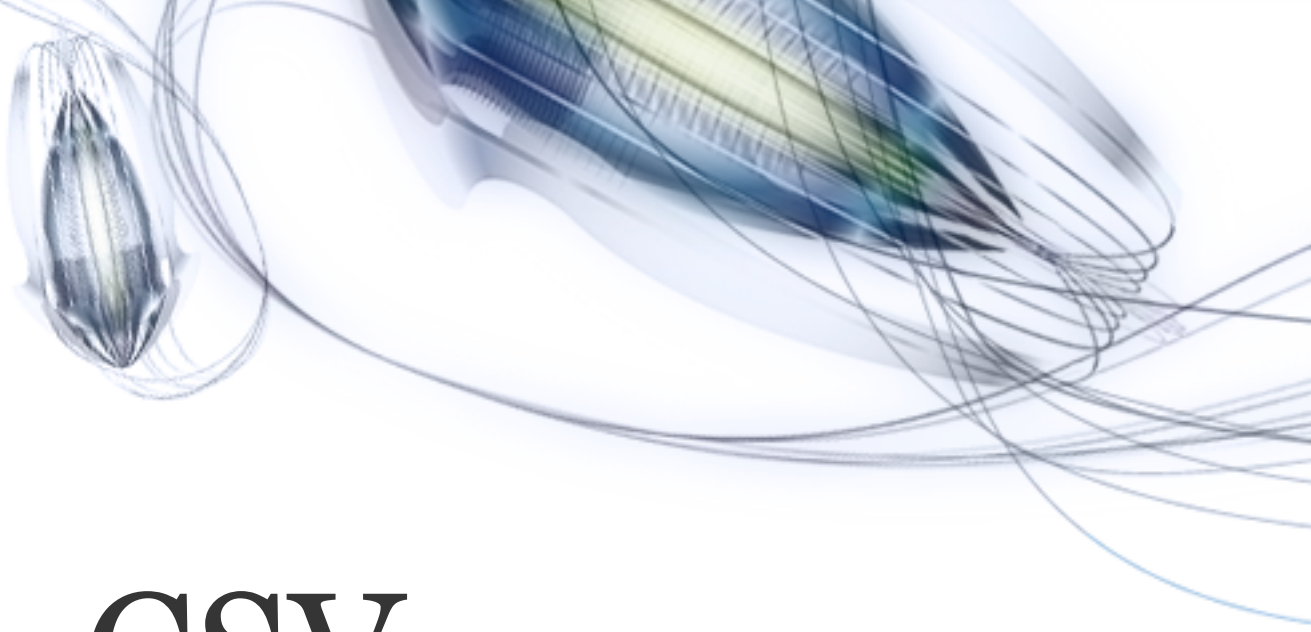
only as good as the parser can correctly tag **NP**

# search pattern

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-search

```
from pattern.web import Twitter
from pattern.en import Sentence, parse
from pattern.search import match

engine = Twitter()
for tweet in engine.search("is cooler than"):
    s = tweet.description
    s = Sentence(parse(s, lemmata=True))
    m = match("NP be cooler than NP", s)

    if m is not None:
        print m.constituents()
        print " win:", m.constituents()[0]
        print "fail:", m.constituents()[-1]
        print
```



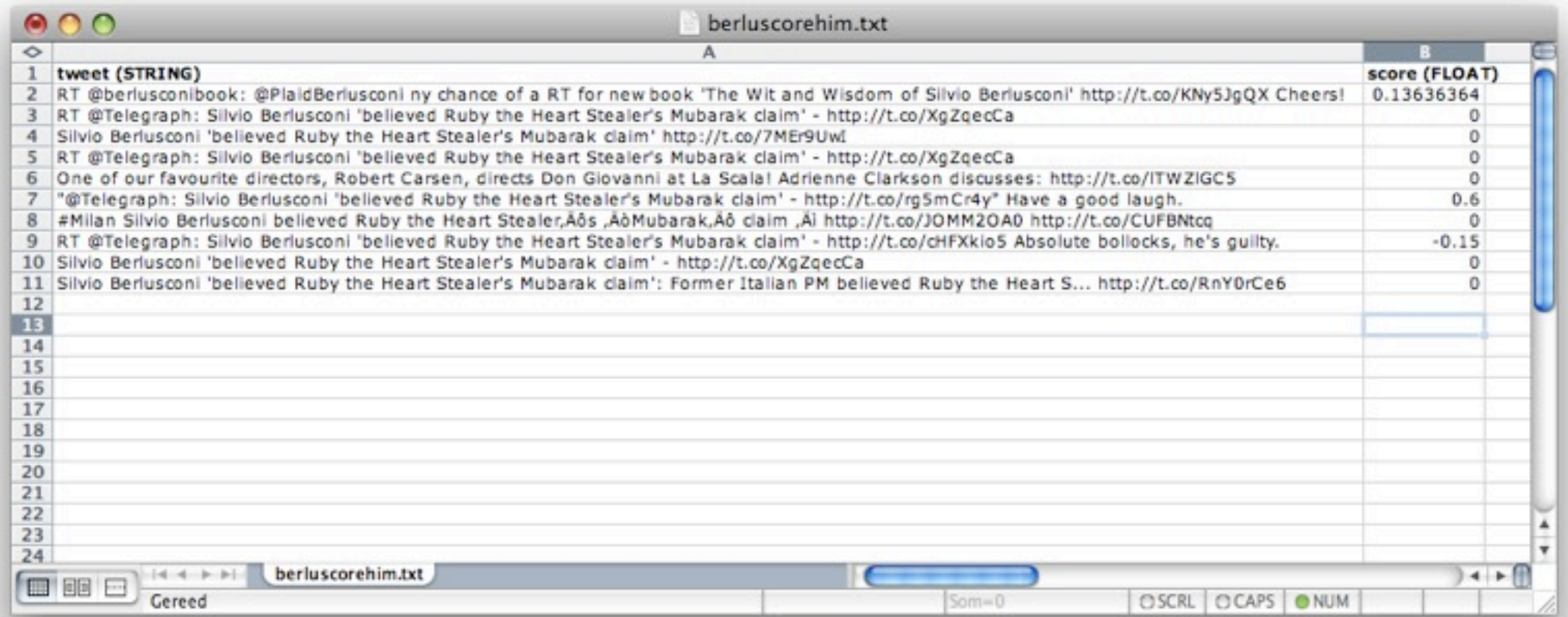
# CSV

# CSV

text file, comma-separated lines of data

# CSV

text file, comma-separated lines of data



berluscorehim.txt

| | A | B |
|---|---|---|
| 1 | tweet (STRING) | score (FLOAT) |
| 2 | RT @berlusconibook: @PlaidBerlusconi ny chance of a RT for new book 'The Wit and Wisdom of Silvio Berlusconi' http://t.co/KNy5JgQX Cheers! | 0.13636364 |
| 3 | RT @Telegraph: Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim' - http://t.co/XgZqecCa | 0 |
| 4 | Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim' http://t.co/7MEr9Uwl | 0 |
| 5 | RT @Telegraph: Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim' - http://t.co/XgZqecCa | 0 |
| 6 | One of our favourite directors, Robert Carsen, directs Don Giovanni at La Scala! Adrienne Clarkson discusses: http://t.co/ITWZIGC5 | 0 |
| 7 | "@Telegraph: Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim' - http://t.co/rg5mCr4y" Have a good laugh. | 0.6 |
| 8 | #Milan Silvio Berlusconi believed Ruby the Heart Stealer,Âôs ‚ÀòMubarak‚Äô claim ‚Äì http://t.co/JOMM2OA0 http://t.co/CUFBNtcq | 0 |
| 9 | RT @Telegraph: Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim' - http://t.co/cHFXkio5 Absolute bollocks, he's guilty. | -0.15 |
| 10 | Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim' - http://t.co/XgZqecCa | 0 |
| 11 | Silvio Berlusconi 'believed Ruby the Heart Stealer's Mubarak claim': Former Italian PM believed Ruby the Heart S... http://t.co/RnY0rCe6 | 0 |
| 12 | | |
| 13 | | |
| 14 | | |
| 15 | | |
| 16 | | |
| 17 | | |
| 18 | | |
| 19 | | |
| 20 | | |
| 21 | | |
| 22 | | |
| 23 | | |
| 24 | | |

berluscorehim.txt

Gereed | Som=0 | SCRL | CAPS | NUM

# CSV

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-db

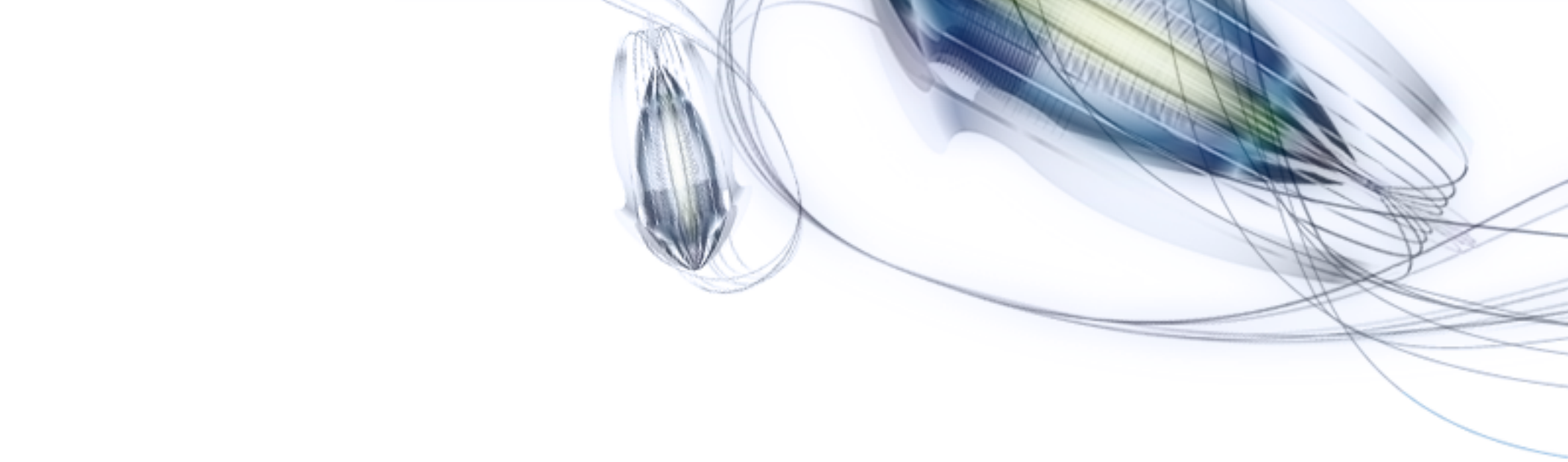
```
from pattern.web import Twitter
from pattern.en import sentiment
from pattern.db import Datasheet, FLOAT, STRING

table = Datasheet(headers=[
    ("tweet", STRING),
    ("score", FLOAT)
])

engine = Twitter(language="en")
for tweet in engine.search("Berlusconi"):
    s = tweet.description
    polarity, subjectivity = sentiment(s)
    table.append((s, polarity))

table.save("berluscorehim.txt", headers=True)
```

# CSV

## PATTERN code

http://www.clips.ua.ac.be/pages/pattern-db

```
from pattern.db import Datasheet, FLOAT, STRING

table = Datasheet.load("berluscorehim.txt", headers=True)

for i, row in enumerate(table):
    for j, cell in enumerate(row):
        print "row " + str(i+1)
        print table.headers[j][0] + ":", cell
        #if table.headers[j][0] == "score":
        #    table[i][j] = 0.0
    print

#table.save("berluscorehim2.txt", headers=True)
```



join the Google group