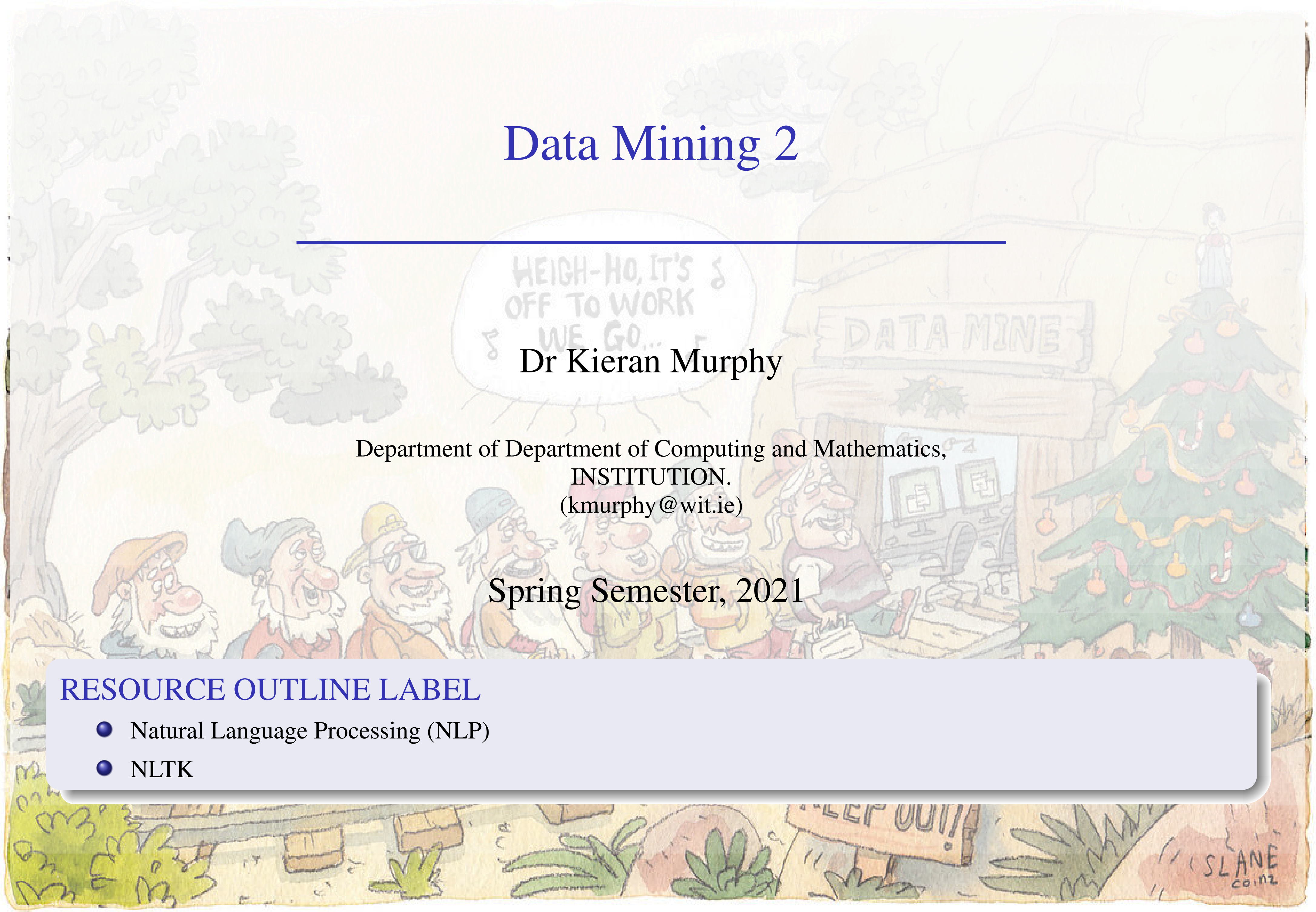



# Data Mining 2

Dr Kieran Murphy

Department of Department of Computing and Mathematics,
INSTITUTION.
(kmurphy@wit.ie)

Spring Semester, 2021

## RESOURCE OUTLINE LABEL

- Natural Language Processing (NLP)
- NLTK

# What is Natural Language Processing?

Any computation, manipulation of natural language

## Tasks

- Classify text documents
- Search for relevant text documents
- Sentiment analysis
- Topic modeling
- Counting words, counting frequency of words
- Finding sentence boundaries
- Part of speech tagging
- Parsing the sentence structure
- Identifying semantic roles
- Identifying entities in a sentence
- Finding which pronoun refers to which entity

## Natural languages evolve

- New words get added ........ selfe
- Old words lose popularity ........ thou
- Meanings of words change ........ netflix and chil
- Language rules themselves may change ........ position of verbs in sentences

# Text Mining Dimensions

- Estimated to be 2.5 Exabytes (2.5 million TB) a day
  - Grow to 40 Zettabytes (40 billion TB) by 2020 (50-times that of 2010)
- Approximately 80% of all data is estimated to be unstructured, text-rich data
  - >40 million articles (5 million in English) in Wikipedia
  - >4.5 billion Web pages
  - >500 million tweets a day, 200 billion a year
  - >1.5 trillion queries on Google a year


- Non-binary
  - Weakly Structured

    few structural cues to text based on layout or markups — research papers, . . .
  - Semi-structured

    extensive format elements, metadata, field labels research papers, . . .

# Why is Text Mining Hard?

- Language is ambiguous
- Context is needed to clarify
- The same words can mean different things (**homographs**)
  - Bear (verb) — to support or carry
  - Bear (noun) — a large animal
- Different words can mean the same thing (**synonyms**) — but synonyms can have differing connotations …
  - Mary became a kind of big sister to Ben.
  - Mary became a kind of large sister to Ben.
- Language is subtle
- Concept / Word extraction usually results in huge number of "dimensions"
  - Thousands of new attributes/features
  - Each features typically has low information content (sparse)
- Mispellings, abbreviations, spelling variants
  - Renders search engines, SQL queries, Regex, … ineffective

- For example, see order of adjectives in English

  Quantity, Quality, Size, Age, Shape, Colour, Proper adjective, Purpose

# Python NLP Libraries

## Core

- Natural Language Tool Kit (NLTK) www.nltk.org
  - De facto standard NLP library in Python.
  - Large collection of examples (data and models)
- Fuzzywuzzy github.com/seatgeek/fuzzywuzzy
  - Minimal, easy to use, module for fuzzy string matching, using the Levenshtein Distance.
  - Ideal for pre-cleaning incorrect text — e.g., miss-spelling of months.

## Of Interest

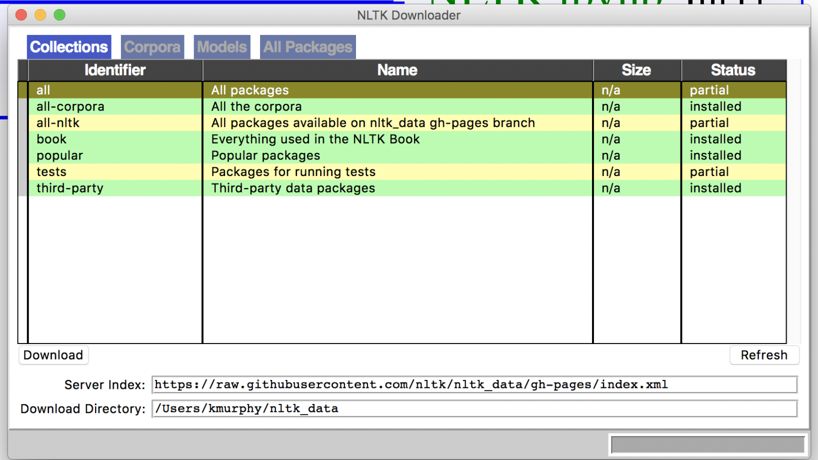
- spaCy spacy.io
  - A relatively new project, currently very popular on github.
  - The library provides most of the standard functionality (tokenisation, PoS tagging, parsing, named entity recognition, . . . ) and is built to be lightning fast.
- TextBlob textblob.readthedocs.io/en/dev/
  - Based on NLTK and Pattern and provides an more uniform/consistent interface to the NLP algorithms.

# NTK — Import and Download of Datasets

The NLTK library is very mature (started 2001) with a rich API that is well documented.*

NLTK.ipynb In[1]:

```
import nltk
```




The NLTK contains a huge amount of (example) data, corpora and pre-trained models, which are not all installed by default.
To download (uncomment) and run

NLTK.ipynb In[2]:

```
# nltk.download()
```

Access the NLTK book examples . . .

```
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

*Natural Language Processing with Python* by Bird, Klein, and Loper, www.nltk.org/book.

# Accessing the Examples

## Text Level

NLTK.ipynb In[4]:

```
texts()
```

```
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

NLTK.ipynb In[5]:

```
text1
```

```
<Text: Moby Dick by Herman Melville 1851>
```

## Sentence Level

NLTK.ipynb In[6]:

```
sents()
```

```
sent1: Call me Ishmael .
sent2: The family of Dashwood had long been settled in Sussex .
sent3: In the beginning God created the heaven and the earth .
sent4: Fellow - Citizens of the Senate and of the House of Represen
sent5: I have a problem with people PMing me to lol JOIN
sent6: SCENE 1 : [ wind ] [ clop clop clop ] KING ARTHUR : Whoa ther
sent7: Pierre Vinken , 61 years old , will join the board as a none
sent8: 25 SEXY MALE , seeks attrac older single lady , for discreet
sent9: THE suburb of Saffron Park lay on the sunset side of London
```

NLTK.ipynb In[7]:

```
sent1
```

```
['Call', 'me', 'Ishmael', '.']
```

# Simple NLP Tasks — Counting Vocabulary of Words

We can access/manipulate NLTK texts (type nltk . text . Text) and sentences (type list ) using our usual python constructs . . .

NLTK.ipynb In[8]:

```
print(text7)
print("Number␣of␣words␣%s" % len(text7))
print("Number␣of␣distinct␣words␣%s" % len(set(text7)))
```

```
<Text: Wall Street Journal>
Number of words 100676
Number of distinct words 12408
```

NLTK.ipynb In[10]:

```
print(sent1)
print("Number␣of␣words␣%s" % len(sent1))
print("Number␣of␣distinct␣words␣%s" % len(set(sent1)))
```

Note that punctation is counted!

```
['Call', 'me', 'Ishmael', '.']
Number of words 4
Number of distinct words 4
```

# Simple NLP Tasks — Frequency of Words

Given a collection, use NLTK's FreqDist function to construct a dictionary of the frequency of each word . . .

```
dist = FreqDist(text7)
print(type(dist))
print("Number of distinct words %s" % len(dist))
```

```
<class 'nltk.probability.FreqDist'>
Number of distinct words 12408
```

Comparable to standard dictionary . . . usual methods apply . . .

NLTK.ipynb In[12]:

```
vocab1 = dist.keys()
list(vocab1)[:10]
```

```
['Pierre', 'Vinken', ',', '61', 'years', 'old', 'will', '
```

**Q:** How many times did word “four" appear?

NLTK.ipynb In[13]:

```
dist["four"]
```

```
20
```

**Q:** What are the frequent words?

NLTK.ipynb In[14]:

```
freqwords = [w for w in vocab1 if len(w) > 5 and dist[w] > 100]
print(freqwords)
```

```
['billion', 'company', 'president', 'because', 'market', 'million', 'shares', 'trading
```

# NLTK — Searching Text

A **concordanc view** shows every occurrence of a given word, with some context.

NLTK.ipynb In[15]:

```
text1.concordance("monstrous")
```

```
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```

Find what other words have appeared in a similar range of contexts:

NLTK.ipynb In[16]:

```
text1.similar("monstrous")
```

```
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless
```

# Normalization and Stemming I

**Normalization**

A process that converts a list of words to a more uniform sequence.

- ✔ Useful in preparing text for later processing.
  - Converting to lowercase
  - Removing stopwords
  - Stemming, . . .
- ✔ A standard format, will simplify later other operations — "separation of concerns".
- ✔ Can improve text matching. For example, the term "modem router" can be expressed, such as modem and router, modem & router, modem/router, and modem-router.
- ✘ But can also compromise an NLP task — converting to lowercase letters can decrease the reliability of searches when the case is important.

NLTK.ipynb In[17]:

```
input1 = "List␣listed␣lists␣listing␣listings"

words1 = input1.lower().split()
words1
```

```
['list', 'listed', 'lists', 'listing', 'listings']
```

# Normalization and Stemming II

**Stemming**

A (crude heuristic) process that chops off the ends of words to get a common root.

- Language dependent. For English Porter's algorithm (1980) has repeatedly been shown to be empirically very effective:
  - consists of 5 phases of word reductions,
  - within each phase there are conventions to select rules,

| Rule | | | Example | | |
|---|---|---|---|---|---|
| SSES | → | SS | caresses | → | caress |
| IES | → | I | ponies | → | poni |
| SS | → | SS | caress | → | caress |
| S | → | | cats | → | cat |

NLTK.ipynb In[18]:

```
porter = nltk.PorterStemmer()
[porter.stem(t) for t in words1]
```

```
['list', 'list', 'list', 'list', 'list']
```

But

NLTK.ipynb In[19]:

```
[porter.stem(t) for t in ["see", "saw", "seeing","to see"]]
```

```
['see', 'saw', 'see', 'to se']
```

# Comparison of Stemming Algorithms

***Sample text:*** Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

***Lovins stemmer:*** such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

***Porter stemmer:*** such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

***Paice stemmer:*** such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

# Lemmatisation

**Lemmatisation**

A process that determines the **lemma** of a word. A lemma can be thought of as the dictionary form of a word. For example, the lemma of “was” is “be”.

- ✘ Computational much more expensive than stemming.
- ✔ But will always return a valid word.
- Empericaly:
  - Stemming increases recall while harming precision.
  - Benefit of lemmatisation can be modest — especially for retrieval in Engligh.

# Steaming vs Lemmatisation

NLTK.ipynb In[21]:

```
WNlemma = nltk.WordNetLemmatizer()
udhr = nltk.corpus.udhr.words('English-Latin1')
print(udhr[:20])
```

```
['Universal', 'Declaration', 'of', 'Human', 'Rights', 'Preamble', 'Whereas', 'recognitio
```

Applying stemming and lemmatisation to the first 20 words ...

NLTK.ipynb In[22]:

```
[porter.stem(t) for t in udhr[:20]]
```

```
['univers',
 'declar',
 'of',
 'human',
 'right',
 'preambl',
 'wherea',
 'recognit',
 'of',
 'the',
 'inher',
 'digniti',
 'and',
 'of',
 'the',
 'equal',
```

NLTK.ipynb In[23]:

```
[WNlemma.lemmatize(t) for t in udhr[:2
```

```
['Universal',
 'Declaration',
 'of',
 'Human',
 'Rights',
 'Preamble',
 'Whereas',
 'recognition',
 'of',
 'the',
 'inherent',
 'dignity',
 'and',
 'of',
 'the',
 'equal',
```

# Tokenisation

**Tokenisation**
Splitting a sentence into words / tokens.

Surly, this is easy … could just use the string split method ?

NLTK.ipynb In[24]:

```
text11 = "Children shouldn't drink a sugary drink before bed."
text11.split()
```

```
['Children', "shouldn't", 'drink', 'a', 'sugary', 'drink', 'before', 'bed.']
```

However, NLTK provides a better (semantics wise) split …

NLTK.ipynb In[25]:

```
print(nltk.word_tokenize(text11))
```

```
['Children', 'should', "n't", 'drink', 'a', 'sugary', 'drink', 'before', 'bed', '.']
```

(i.e., separated punctation and “shouldn’t”)

# Sentence Splitting

**Sentence Splitting**

Splitting a body of text into sentences.

This is harder again … using the string `split(".")` method is optimistic at best.

## Example

```
"This is the first sentence. After Brexit milk in the U.K. will cost 9.99. Is this the third sentence? Yes, it is!"
```

NLTK.ipynb In[27]:

```
for s in text12.split("."):
    print(s)
```

```
This is the first sentence
 After Brexit milk in the U
K
 will cost 9
99
 Is this the third sentence? Yes, it is!
```

NLTK.ipynb In[28]:

```
sentences = nltk.sent_tokenize(text12)
for s in sentences:
    print(s)
```

```
This is the first sentence.
After Brexit milk in the U.K. will cost 9.99.
Is this the third sentence?
Yes, it is!
```

# Part of Speech (POS) Tagging

**Part of Speech (POS)**

Is a special label assigned to each token (word) in a text corpus to indicate the part of speech and often also other grammatical categories such as tense, number (plural/singular), case etc.

| Number | Tag | Description |
|---|---|---|
| 1 | CC | Coordinating conjunction |
| 2 | CD | Cardinal number |
| 3 | DT | Determiner |
| 4 | EX | Existential *there* |
| 5 | FW | Foreign word |
| 6 | IN | Preposition or subordinating conjunction |
| 7 | JJ | Adjective |
| 8 | JJR | Adjective, comparative |
| 9 | JJS | Adjective, superlative |
| 10 | LS | List item marker |
| 11 | MD | Modal |
| 12 | NN | Noun, singular or mass |
| 13 | NNS | Noun, plural |
| 14 | NNP | Proper noun, singular |
| 15 | NNPS | Proper noun, plural |
| 16 | PDT | Predeterminer |
| 17 | POS | Possessive ending |
| 18 | PRP | Personal pronoun |

| Number | Tag | Description |
|---|---|---|
| 19 | PRP$ | Possessive pronoun |
| 20 | RB | Adverb |
| 21 | RBR | Adverb, comparative |
| 22 | RBS | Adverb, superlative |
| 23 | RP | Particle |
| 24 | SYM | Symbol |
| 25 | TO | *to* |
| 26 | UH | Interjection |
| 27 | VB | Verb, base form |
| 28 | VBD | Verb, past tense |
| 29 | VBG | Verb, gerund or present participle |
| 30 | VBN | Verb, past participle |
| 31 | VBP | Verb, non-3rd person singular present |
| 32 | VBZ | Verb, 3rd person singular present |
| 33 | WDT | Wh-determiner |
| 34 | WP | Wh-pronoun |
| 35 | WP$ | Possessive wh-pronoun |
| 36 | WRB | Wh-adverb |

# Example

| Tag | Meaning | English Examples |
|---|---|---|
| ADJ | adjective | *new, good, high, special, big, local* |
| ADP | adposition | *on, of, at, with, by, into, under* |
| ADV | adverb | *really, already, still, early, now* |
| CONJ | conjunction | *and, or, but, if, while, although* |
| DET | determiner, article | *the, a, some, most, every, no, which* |
| NOUN | noun | *year, home, costs, time, Africa* |
| NUM | numeral | *twenty-four, fourth, 1991, 14:24* |
| PRT | particle | *at, on, out, over per, that, up, with* |
| PRON | pronoun | *he, their, her, its, my, I, us* |
| VERB | verb | *is, say, told, given, playing, would* |
| . | punctuation marks | *. , ; !* |
| X | other | *ersatz, esprit, dunno, gr8, univeristy* |

N

```
text = "Children shouldn't drink a sugary drink before bed."
text2 = nltk.word_tokenize(text)
nltk.pos_tag(text2)
```

```
[('Children', 'NNP'),
 ('should', 'MD'),
 ("n't", 'RB'),
 ('drink', 'VB'),
 ('a', 'DT'),
 ('sugary', 'JJ'),
 ('drink', 'NN'),
 ('before', 'IN'),
 ('bed', 'NN'),
 ('.', '.')]
```

**/NNP** proper noun
**/MD** Modal
**/RB** Adverb
**/VB** Base verb
**/DT** Determiner
**/JJ** Adjective
**/NN** Noun, singular or mass
**/IN** Preposition

# POS Tagging — How hard is it?

- $\approx$ 89% of English words have only one part of speech (unambiguous).
  - However, many common words in English are ambiguous.
  - But even these can largely be disambiguated by rules or probabilistically.
- Taggers can be rule-based, stochastic (training on a labelled set of words using Hidden Markov Models (HMMs)), or a combination (most popular combination is the “Brill” tagger).

## Example of stochastic tagging

The sentence

“Secretariat is expected to race tomorrow”

has POS tagging:

| NNP | VBZ | VBN | TO | VB | NR |
|---|---|---|---|---|---|
| Secretariat | is | expected | to | race | tomorrow |

| NNP | VBZ | VBN | TO | NN | NR |
|---|---|---|---|---|---|
| Secretariat | is | expected | to | race | tomorrow |

**/NNP**
proper noun
**/VB**
Base verb
**/VBN**
verb, past participle
**/VBZ**
verb, 3rd prsn
**/TO**
to

Looking at transition probabilities (going from **TO** to a **VB** or a **NN**) we have

$$\left.\begin{aligned}\Pr(\mathbf{NN}|\mathbf{TO}) &= 0.0047\\ \Pr(\mathbf{VB}|\mathbf{TO}) &= 0.83\end{aligned}\right\} \Longrightarrow \text{“race” is most likely a verb}$$

What about “The old man the boat”?

# Making Sense of Sentences

Making sense of sentences is easy if they follow a well-defined grammatical structure.

NLTK.ipynb In[32]:

```python
text14 = nltk.word_tokenize("Alice loves Bob")
nltk.pos_tag(text14)
```

```
[('Alice', 'NNP'), ('loves', 'VBZ'), ('Bob', 'NNP')]
```

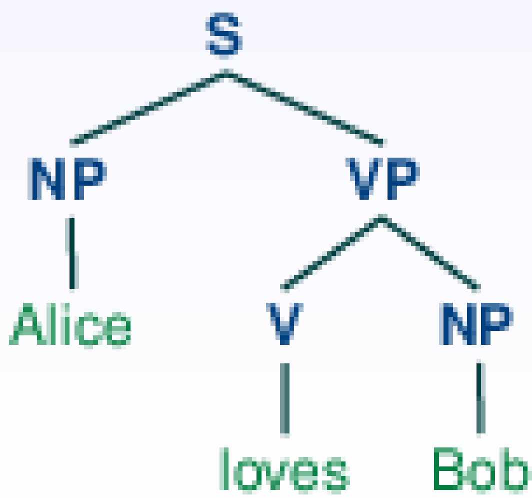
NLTK.ipynb In[33]:

```python
grammar = nltk.CFG.fromstring("""
 S -> NP VP
 VP -> V NP
 NP -> 'Alice' | 'Bob'
 V -> 'loves'
 """)
parser = nltk.ChartParser(grammar)

trees = parser.parse_all(text14)
for tree in trees:
    print(tree)

trees[0]
```

```
(S (NP Alice) (VP (V loves) (NP Bob)))
```



# Term Frequency (TF)

**Term Frequency (TF)**
Number of times the term occurs in a document

## Assumption

- If term occurs more often, it measures something important.
- $2\times$ as many occurrences is $2\times$ as important
  - This can be mitigated if need be — common "fix" is to transform using log transform: ("plus 1" to avoid NaN)

$$\log 10(1 + TF)$$

- Each occurrence is an independent event (not a replicate). Is it true?
  - Information retrieval: probably "yes"
  - Fraud detection, notes, log files: maybe "no"

Since documents vary in length, it is possible that a term would appear much more times in long documents than shorter ones.

- Normalise by dividing by total number of terms in document.

$$\text{TF}(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$$

# Document Frequency (DF)

**Document Frequency (DF)**

Number of documents the term occurs in

## Assumption

- Terms that occur in fewer documents are more specified to a document and more descriptive of the content: rarity matters.
- Terms that occur in most documents are common words, not as descriptive. Is it true?
  - Sometimes "yes"
  - Sometimes just reflect textual variants (synonyms), regional differences, personal style.

Again, normalise with respect to number of documents

$$\mathrm{DF}(t) = \frac{\text{Number of document term } t \text{ appears in}}{\text{Total number of documents}}$$

# Inverse Document Frequency (IDF)

- For DF, smaller is better — we often want a larger number to be “better”.
- Possible transforms:
  - The reciprocal is too severe:

$$\mathrm{IDF}(t) = \frac{1}{\mathrm{DF(t)}}$$

  - Better, more popular definition

$$\mathrm{IDF}(t) = \log_{10}\left(1 + \frac{1}{\mathrm{DF(t)}}\right)$$

    - Again, use of log to “compress” (slows growth rate) an interval $[1, \infty)$.
    - Don’t have to use base 10 logs – natural logs are same up to constant factor.

# TF-IDF

**TD-IDF**

Term frequency–inverse document frequency.

- Separately, DF and IDF can be good features
- Together, they represent a good idea

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

- Assumption: Higher frequency of terms that are rare may indicate a very important concept
- Why multiply? Are these “independent”?
  - No, but multiplying seems to work just fine
- TF-IDF can be successfully used for stop-words filtering in various subject fields including text summarisation and classification.
- Variations of the TF-IDF weighting scheme are often used by search engines as a central tool in scoring and ranking a document’s relevance given a user query.