ENSF 612
Lecture - Natural Language Processing (NLP)
Parts of Speech Tagging,
Stemming, and Lemmatization
Textual Similarity

Gias Uddin, Assistant Professor
Department of Electrical and Software Engineering,
University of Calgary
https://giasuddin.ca/

Parts of Speech (POS)

Wikipedia:
In traditional grammar, a **part of speech** or **Part-of-speech** (abbreviated as **POS** or **PoS**), is a category of words (or, more generally, of lexical items) that have similar grammatical properties. Words that are assigned to the same part of speech generally display similar syntactic behavior—they play similar roles within the grammatical structure of sentences—and sometimes similar morphology in that they undergo inflection for similar properties.

NLTK POS Tagger

```
>>> text = word_tokenize("They refuse to permit us to obtain the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

Source: https://www.nltk.org/book/ch05.html

Parts of Speech (POS) Tagging using NLTK

```
1  import nltk
2  nltk.download('punkt')
3  nltk.download('maxent_treebank_pos_tagger')
4  nltk.download('averaged_perceptron_tagger')
```

```
1  from nltk.tag import pos_tag
2  from nltk.tokenize import word_tokenize
3
4
5  text = "They refuse to permit us to obtain the refuse permit"
6  words = word_tokenize(text)
7  print(words)
8  pos_tag(words)
```

```
['They', 'refuse', 'to', 'permit', 'us', 'to', 'obtain', 'the', 'refuse', 'permit']
Out[45]: [('They', 'PRP'),
 ('refuse', 'VBP'),
 ('to', 'TO'),
 ('permit', 'VB'),
 ('us', 'PRP'),
 ('to', 'TO'),
 ('obtain', 'VB'),
 ('the', 'DT'),
 ('refuse', 'NN'),
 ('permit', 'NN')]
```

POS Tagging

The POS Tagging problem is to determine the parts of speech of a particular word instance in a given sentence

Challenge: Words can have more than one POS: e.g., back

- The *back* door = JJ (adjective)
- On my *back* = NN (noun)
- Win the voters *back* = RB (adverb)
- Promised to *back* the bill = VB (verb)

Input: Plays well with others

Ambiguities:

1. "Plays" can be noun/verb
2. "well" can be adjective/noun/adverb and even interjection

Correct output:

Plays/VBZ (verb) well/RB (adverb) with/IN others/NNS

Source: http://web.stanford.edu/class/cs124/lec/postagging.pdf

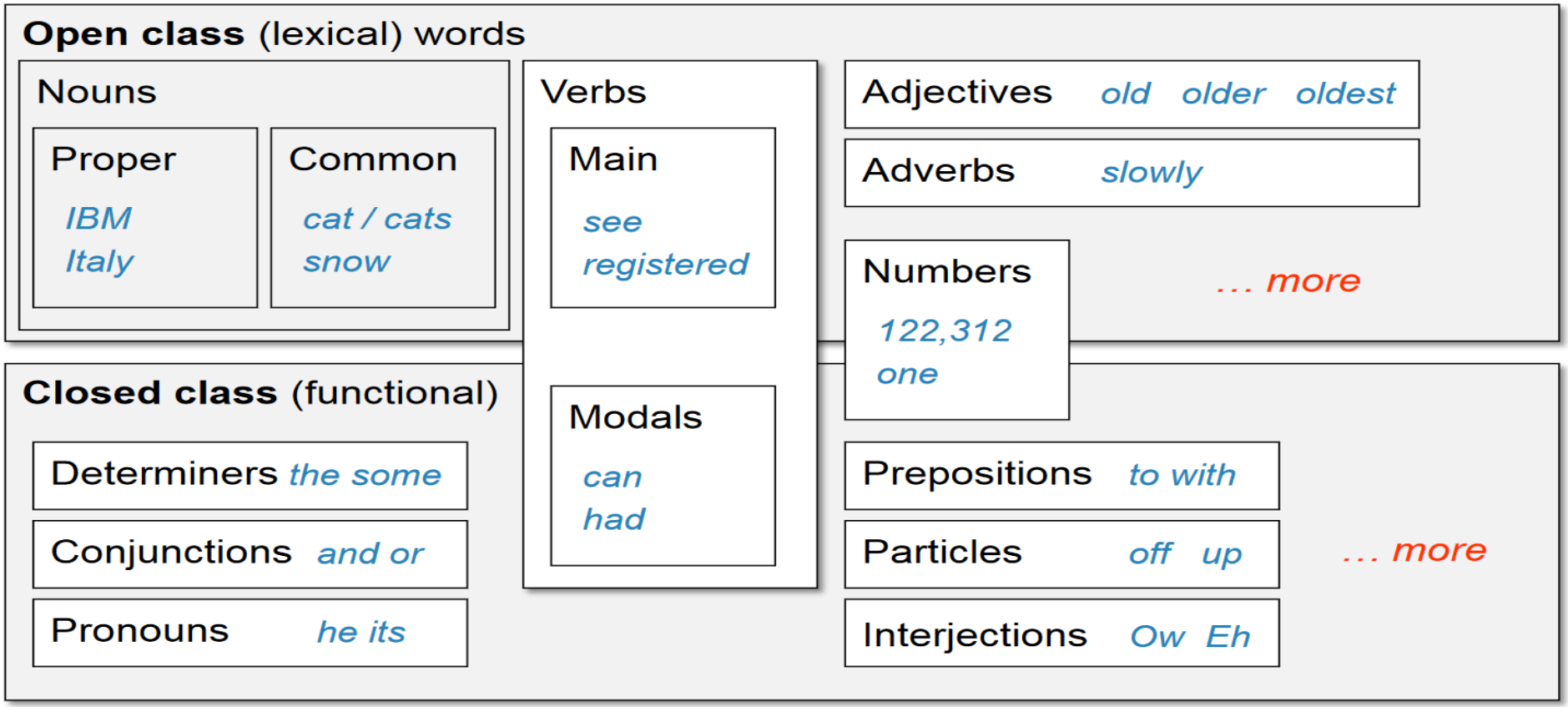
Penn POS Treebank

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Source: http://web.stanford.edu/class/cs124/lec/postagging.pdf

POS Tagging Open vs Closed Classes

- **Open class:**
  - Impossible to completely enumerate the right tag in all instances
  - New words are continuously being added/invented
- **Closed class:**
  - Closed, fixed membership all the time
  - As such, reasonably easy to enumerate the right tag



Closed Class POS

Prepositions

In English, prepositions normally occur before noun phrases to denote relations

e.g., He came ***by*** the office in a hurry!

e.g., ***on*** the shelf, ***before*** noon

Particles

Resembles a preposition, but used with a verb to constitute a phrase

e.g., find ***out,*** turn ***over***, go ***on***

Example	POS
He came by the office in a hurry	(by = preposition)
He came by his fortunes honestly	(by = particle)
We ran up the phone bill	(up = preposition)
We ran up the small hill	(up = particle)
He lived down the block	(down = preposition)
He never lived down the nicknames	(down = particle)

Source: http://www.cs.umd.edu/class/fall2017/cmsc723/slides/slides_09.pdf

Closed Class POS

Determiners

- Reference for a noun
- Examples: a, an, the, that, this, many, …

Pronouns

- Refer to a noun (person, entity): he, she, it
- Possessive pronouns: his, her, its
- Wh-pronous: what, who

Coordinating Conjunctions

- Join two elements of "equal status": cats ***and*** dogs, salad ***or*** soup

Subordinating Conjunctions

- Join two elements of "unequal status": we will do this ***after*** you finish your work
- Complements: I think ***that*** you should finish your assignment

Source: http://www.cs.umd.edu/class/fall2017/cmsc723/slides/slides_09.pdf

Open Class POS

Nouns

- Generally, words for people, places, entities, etc.
- New words are added all the time: muggle (harry potter), webinar, ….

Verbs

- New words are added all the time: google, tweet, ….

Adjectives

- Generally modify nouns, e.g., ***crazy*** world

Adverbs

- Modify verbs: ***worked*** wonderful!
- Modify adjectives : ***very*** well!

Source: http://www.cs.umd.edu/class/fall2017/cmsc723/slides/slides_09.pdf

Word Stemming

Wikipedia

In linguistic morphology and information retrieval, **stemming** is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation.

A computer program or subroutine that stems word may be called a *stemming program*, *stemming algorithm*, or *stemmer*.

Examples:

A stemmer for English operating on the stem ***cat*** should identify such strings as *cats*, *catlike*, and *catty*. A stemming algorithm might also reduce the words *fishing*, *fished*, and *fisher* to the stem ***fish***. The stem need not be a word, for example the Porter algorithm reduces, *argue*, *argued*, *argues*, *arguing*, and *argus* to the stem ***argu***.

https://en.wikipedia.org/wiki/Stemming

Word Stemming

Stemming also called suffix stripping, is a technique to reduce text dimensionality.

Stemming is a type of text normalization that enables to standardize some words into specific expressions called stems.

Word	Stem
Program	Program
Programs	Program
Programmer	Program
Programming	Program
Programmers	Program

https://en.wikipedia.org/wiki/Stemming

Porter Stemming Algorithm

Step 1a.

A *consonant* in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term `consonant' is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a *vowel*.

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:

```
CVCV ... C
CVCV ... V
VCVC ... C
VCVC ... V
```

These may all be represented by the single form

```
[C]VCVC ... [V]
```

```
SSES -> SS                      caresses  ->  caress
IES  -> I                       ponies    ->  poni
                                ties      ->  ti
SS   -> SS                      caress    ->  caress
S    ->                         cats      ->  cat
```

Source: http://facweb.cs.depaul.edu/mobasher/classes/csc575/papers/porter-algorithm.html

Porter Stemming Algorithm

Remove suffixes from words to convert those into "root" forms

Step 1a

```
SSES -> SS                      caresses  ->  caress
IES  -> I                       ponies    ->  poni
                                ties      ->  ti
SS   -> SS                      caress    ->  caress
S    ->                         cats      ->  cat
```

Step 1b

```
(m>0) EED -> EE                 feed      ->  feed
                                agreed    ->  agree
(*v*) ED  ->                    plastered ->  plaster
                                bled      ->  bled
(*v*) ING ->                    motoring  ->  motor
                                sing      ->  sing
```

If the second or third of the rules in Step 1b is successful, the following is done:

```
AT -> ATE                       conflat(ed)  ->  conflate
BL -> BLE                       troubl(ed)   ->  trouble
IZ -> IZE                       siz(ed)      ->  size
(*d and not (*L or *S or *Z))
   -> single letter
                                hopp(ing)    ->  hop
                                tann(ed)     ->  tan
                                fall(ing)    ->  fall
                                hiss(ing)    ->  hiss
                                fizz(ed)     ->  fizz
(m=1 and *o) -> E               fail(ing)    ->  fail
                                fil(ing)     ->  file
```

The rule to map to a single letter causes the removal of one of the double letter pair. The -E is put back on -AT, -BL and -IZ, so that the suffixes -ATE, -BLE and -IZE can be recognised later. This E may be removed in step 4.

Source: http://facweb.cs.depaul.edu/mobasher/classes/csc575/papers/porter-algorithm.html

Porter Stemming Algorithm

Remove suffixes from words to convert those into "root" forms

Step 1c

```
(*v*) Y -> I                    happy        ->  happi
                                sky          ->  sky
```

Step 1 deals with plurals and past participles. The subsequent steps are much more straightforward.

Step 2

```
(m>0) ATIONAL ->  ATE           relational     ->  relate
(m>0) TIONAL  ->  TION          conditional    ->  condition
                                rational       ->  rational
(m>0) ENCI    ->  ENCE          valenci        ->  valence
(m>0) ANCI    ->  ANCE          hesitanci      ->  hesitance
(m>0) IZER    ->  IZE           digitizer      ->  digitize
(m>0) ABLI    ->  ABLE          conformabli    ->  conformable
(m>0) ALLI    ->  AL            radicalli      ->  radical
(m>0) ENTLI   ->  ENT           differentli    ->  different
(m>0) ELI     ->  E             vileli        - >  vile
(m>0) OUSLI   ->  OUS           analogousli    ->  analogous
(m>0) IZATION ->  IZE           vietnamization ->  vietnamize
(m>0) ATION   ->  ATE           predication    ->  predicate
(m>0) ATOR    ->  ATE           operator       ->  operate
(m>0) ALISM   ->  AL            feudalism      ->  feudal
(m>0) IVENESS ->  IVE           decisiveness   ->  decisive
(m>0) FULNESS ->  FUL           hopefulness    ->  hopeful
(m>0) OUSNESS ->  OUS           callousness    ->  callous
(m>0) ALITI   ->  AL            formaliti      ->  formal
(m>0) IVITI   ->  IVE           sensitiviti    ->  sensitive
(m>0) BILITI  ->  BLE           sensibiliti    ->  sensible
```

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Source: http://facweb.cs.depaul.edu/mobasher/classes/csc575/papers/porter-algorithm.html

Porter Stemming Algorithm

Remove suffixes from words to convert those into "root" forms

Step 3

```
(m>0) ICATE ->  IC              triplicate     ->  triplic
(m>0) ATIVE ->                  formative      ->  form
(m>0) ALIZE ->  AL              formalize      ->  formal
(m>0) ICITI ->  IC              electriciti    ->  electric
(m>0) ICAL  ->  IC              electrical     ->  electric
(m>0) FUL   ->                  hopeful        ->  hope
(m>0) NESS  ->                  goodness       ->  good
```

Step 4

```
(m>1) AL    ->                  revival        ->  reviv
(m>1) ANCE  ->                  allowance      ->  allow
(m>1) ENCE  ->                  inference      ->  infer
(m>1) ER    ->                  airliner       ->  airlin
(m>1) IC    ->                  gyroscopic     ->  gyroscop
(m>1) ABLE  ->                  adjustable     ->  adjust
(m>1) IBLE  ->                  defensible     ->  defens
(m>1) ANT   ->                  irritant       ->  irrit
(m>1) EMENT ->                  replacement    ->  replac
(m>1) MENT  ->                  adjustment     ->  adjust
(m>1) ENT   ->                  dependent      ->  depend
(m>1 and (*S or *T)) ION ->     adoption       ->  adopt
(m>1) OU    ->                  homologou      ->  homolog
(m>1) ISM   ->                  communism      ->  commun
(m>1) ATE   ->                  activate       ->  activ
(m>1) ITI   ->                  angulariti     ->  angular
(m>1) OUS   ->                  homologous     ->  homolog
(m>1) IVE   ->                  effective      ->  effect
(m>1) IZE   ->                  bowdlerize     ->  bowdler
```

The suffixes are now removed. All that remains is a little tidying up.

Source: http://facweb.cs.depaul.edu/mobasher/classes/csc575/papers/porter-algorithm.html

Porter Stemming Algorithm

Remove suffixes from words to convert those into "root" forms

Step 5a

```
(m>1) E     ->                  probate         ->  probat
                                rate            ->  rate
(m=1 and not *o) E ->           cease           ->  ceas
```

Step 5b

```
(m > 1 and *d and *L) -> single letter
                                controll        ->  control
                                roll            ->  roll
```

The algorithm is careful not to remove a suffix when the stem is too short, the length of the stem being given by its measure, m. There is no linguistic basis for this approach. It was merely observed that m could be used quite effectively to help decide whether or not it was wise to take off a suffix. For example, in the following two lists:

```
list A          list B
------          ------
RELATE          DERIVATE
PROBATE         ACTIVATE
CONFLATE        DEMONSTRATE
PIRATE          NECESSITATE
PRELATE         RENOVATE
```

-ATE is removed from the list B words, but not from the list A words. This means that the pairs DERIVATE/DERIVE, ACTIVATE/ACTIVE, DEMONSTRATE/DEMONS- TRABLE, NECESSITATE/NECESSITOUS, will conflate together. The fact that no attempt is made to identify prefixes can make the results look rather inconsistent. Thus PRELATE does not lose the -ATE, but ARCHPRELATE becomes ARCHPREL. In practice this does not matter too much, because the presence of the prefix decreases the probability of an erroneous conflation.

Source: http://facweb.cs.depaul.edu/mobasher/classes/csc575/papers/porter-algorithm.html

Porter Stemming Algorithm

An Example using the word "conditional"

Certain word endings only matter for the algorithm, which is used to define the variable "m". Thus, the suffix "ION" or "TION" are only applied on the second and fourth phase within certain conditions. For "conditional", m = 4 (TION).

1. Phase 1. It doesn't match any of the patterns that this step considers like ending with IES or SSES and other rules present in phase 1b and 1c.
2. Phase 2. As m is greater than 0 and the word ends with "tional" the stemmer applies a cropping of the suffix "al" and "conditional" is trimmed to "condition"
3. Phase 3. The input word is now "condition" and we move to Phase 4.
4. Phase 4. "condition" ends with "ion" where m = 3 (i.e., m > 1). Thus, we further trim "condition" to satisfy m = 1 by "condit"
5. Our input word is now "condit" and it does not match any pattern from this phase, so the final output, i.e., stem is "condit"

Porter Stemming Algorithm in NLTK

```
from nltk.stem.api import StemmerI


class PorterStemmer(StemmerI):
    """
    A word stemmer based on the Porter stemming algorit

        Porter, M. "An algorithm for suffix stripping."
        Program 14.3 (1980): 130-137.

    See http://www.tartarus.org/~martin/PorterStemmer/
    of the algorithm.

    Martin Porter has endorsed several modifications to
    algorithm since writing his original paper, and tha
    included in the implementations on his website. Add
    have proposed further improvements to the algorithm
    contributors. There are thus three modes that can b
    passing the appropriate constant to the class const
    attribute:
```

```
>>> from __future__ import print_function
>>> from nltk.stem import *
```

Unit tests for the Porter stemmer

```
>>> from nltk.stem.porter import *
```

Create a new Porter stemmer.

```
>>> stemmer = PorterStemmer()
```

Test the stemmer on various pluralised words.

```
>>> plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
...            'died', 'agreed', 'owned', 'humbled', 'sized',
...            'meeting', 'stating', 'siezing', 'itemization',
...            'sensational', 'traditional', 'reference', 'colonizer',
...            'plotted']

>>> singles = [stemmer.stem(plural) for plural in plurals]

>>> print(' '.join(singles))  # doctest: +NORMALIZE_WHITESPACE
caress fli die mule deni die agre own humbl size meet
state siez item sensat tradit refer colon plot
```

Source: https://www.nltk.org/howto/stem.html

Snowball Stemming

Also called porter2 stemmer

The logic and process is exactly similar to Porter with five phases, but some stemming rules are different / updated in the snowball stemming process to make better stem.

1. Better handling of adverbs

```
from nltk.stem import SnowballStemmer

snowball = SnowballStemmer(language = 'english')
porter = PorterStemmer()
```

```
def printStem(word):
  ps = porter.stem(word)
  sb = snowball.stem(word)
  print("Word = %s. Stem using Porter Algor = %s, Snowball algo = %s"%(word, ps, sb))
```

```
word = "fairly"
printStem(word)
```

```
Word = fairly. Stem using Porter Algor = fairli, Snowball algo = fair
```

Snowball Stemming

Also called porter2 stemmer

1. Better handling of adverbs

```
word = "loudly"
printStem(word)
```

```
Word = loudly. Stem using Porter Algor = loudli, Snowball algo = loud
```

2. Prevention of some overstemming (i.e., cropping too much suffix from the word)

```
word = "generically"
printStem(word)
```

```
Word = generically. Stem using Porter Algor = gener, Snowball algo = generic
```

```
word = "generous"
printStem(word)
```

```
Word = generous. Stem using Porter Algor = gener, Snowball algo = generous
```

Lancaster Stemming

Developed by Chris Price from Lancaster University.

1. Stemming rules are more aggressive than porter and snowball
2. It is one of the most aggressive stemmer as it tends to overstem lots of words
3. The rule here is to find the shortest possible stem for a given word

```
from nltk.stem import LancasterStemmer

lanc = LancasterStemmer()
```

```
def printStemPSL(word):
  ps = porter.stem(word)
  sb = snowball.stem(word)
  lc = lanc.stem(word)
  print("Word = %s. Stem using Porter Algor = %s, Snowball algo = %s, Lancaster algo = %s"%(word, ps, sb, lc))
```

```
word = "salty"
printStemPSL(word)
```

```
Word = salty. Stem using Porter Algor = salti, Snowball algo = salti, Lancaster algo = sal
```

```
word = "sales"
printStemPSL(word)
```

```
Word = sales. Stem using Porter Algor = sale, Snowball algo = sale, Lancaster algo = sal
```

Impact of Stemming

Useful in applications like Machine learning (ML) where we need to keep the root words to ensure the ML algorithms can learn word-based features in text classification. This means we want to achieve generalization of our ML models across textual data (i.e., from multiple similar platforms)

```
eu_desc = """
The European Union (EU) is a political and economic union of 27 member states that are located
primarily in Europe. Its members have a combined area of 4,233,255.3 km2 (1,634,469.0 sq mi)
and an estimated total population of about 447 million. The EU has developed an internal
single market through a standardised system of laws that apply in all member states in those
matters, and only those matters, where members have agreed to act as one. EU policies aim to
ensure the free movement of people, goods, services and capital within the internal market;
enact legislation in justice and home affairs; and maintain common policies on trade,
agriculture, fisheries and regional development. Passport controls have been abolished for
travel within the Schengen Area. A monetary union was established in 1999, coming into full
force in 2002, and is composed of 19 EU member states which use the euro currency. The EU has
often been described as a sui generis political entity (without precedent or comparison).The
EU and European citizenship were established when the Maastricht Treaty came into force in
1993. The EU traces its origins to the European Coal and Steel Community (ECSC) and the
European Economic Community (EEC), established, respectively, by the 1951 Treaty of Paris and
1957 Treaty of Rome. The original members of what came to be known as the European Communities
were the Inner Six: Belgium, France, Italy, Luxembourg, the Netherlands, and West Germany. The
Communities and their successors have grown in size by the accession of new member states and
in power by the addition of policy areas to their remit. The United Kingdom became the first
member state to leave the EU on 31 January 2020. Before this, three territories of member
states had left the EU or its forerunners. The latest major amendment to the constitutional
basis of the EU, the Treaty of Lisbon, came into force in 2009.
"""
```

Impact of Stemming

```
def checkInfoReduction(text):
  words = word_tokenize(text)
  chars_original = len(''.join(word_tokenize(text)))
  eu_porter = [porter.stem(word) for word in words]
  chars_porter = len(''.join(eu_porter))
  eu_snowball = [snowball.stem(word)  for word in words]
  chars_snowball = len(''.join(eu_snowball))
  eu_lanc = [lanc.stem(word) for word in words]
  chars_lanc = len(''.join(eu_lanc))

  print("Total Chars in Original = %d.\n Porter = %d (Normalized = %.2f).\n Snowball = %d (Normalized = %.2f).\n Lancaster = %d (Normalized = %.2f)"%(
  chars_original,
  chars_porter,
  chars_porter*100/chars_original,
  chars_snowball,
  chars_snowball*100/chars_original,
  chars_lanc,
  chars_lanc*100/chars_original))
```

```
1  checkInfoReduction(eu_desc)
```

```
Total Chars in Original = 1591.
 Porter = 1430 (Normalized = 89.88).
 Snowball = 1446 (Normalized = 90.89).
 Lancaster = 1248 (Normalized = 78.44)
```

Lancaster is the most aggressive in normalization but may still not be best choice in ML process, when it may stem multiple different words into same stem.

Lemmatization

Wikipedia

Lemmatisation (or **lemmatization**) in linguistics 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.[1]

In computational linguistics, lemmatisation is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. As a result, developing efficient **lemmatisation** algorithms is an open area of research.

Stemming vs Lemmatization

1. Stemming cuts off the end or beginning of a word to find root form. Lemmatization queries a database by taking into account the word and its POS to find the root form based on the given context.
2. A stem can be the same for the inflected forms of different lemmas. The same lemma can correspond to different forms of lemma.

https://en.wikipedia.org/wiki/Lemmatisation

Lemmatization

In [2]:
```
from nltk.stem.porter import *
from nltk.stem import WordNetLemmatizer
```

In [3]:
```
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
```

In [18]:
```
words = [('better', 'a'), ('corpora', 'n'), ('corpora', 'v'), ('rocks', 'v') ]
for item in words:
    word = item[0]
    pos = item[1]
    print word
    print "stem: ", stemmer.stem(word)
    print "lemma (without POS): ", lemmatizer.lemmatize(word)
    print "lemma (with POS = "+ pos +"): ", lemmatizer.lemmatize(word, pos=pos)
    print "........."
```

```
better
stem:  better
lemma (without POS):  better
lemma (with POS = a):  good
.........
corpora
stem:  corpora
lemma (without POS):  corpus
lemma (with POS = n):  corpus
.........
corpora
stem:  corpora
lemma (without POS):  corpus
lemma (with POS = v):  corpora
.........
rocks
stem:  rock
lemma (without POS):  rock
lemma (with POS = v):  rock
.........
```

https://en.wikipedia.org/wiki/Lemmatisation

Lemmatization

In [2]:
```
from nltk.stem.porter import *
from nltk.stem import WordNetLemmatizer
```

In [3]:
```
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
```

In [18]:
```
words = [('better', 'a'), ('corpora', 'n'), ('corpora', 'v'), ('rocks', 'v') ]
for item in words:
    word = item[0]
    pos = item[1]
    print word
    print "stem: ", stemmer.stem(word)
    print "lemma (without POS): ", lemmatizer.lemmatize(word)
    print "lemma (with POS = "+ pos +"): ", lemmatizer.lemmatize(word, pos=pos)
    print "........."
```

```
better
stem:  better
lemma (without POS):  better
lemma (with POS = a):  good
.........
corpora
stem:  corpora
lemma (without POS):  corpus
lemma (with POS = n):  corpus
.........
corpora
stem:  corpora
lemma (without POS):  corpus
lemma (with POS = v):  corpora
.........
rocks
stem:  rock
lemma (without POS):  rock
lemma (with POS = v):  rock
.........
```

https://en.wikipedia.org/wiki/Lemmatisation

Textual Similarity

Problem: How can we compute the similarity between texts?
Text1 = “This is example is about similarity”
Text2 = “This is example is about similarity too”
Text3 = “This is a sports news not example”

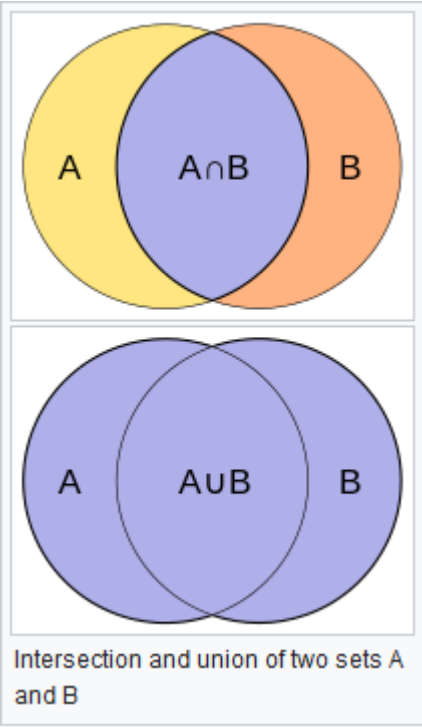
Many metrics are proposed:

1. Jaccard Index:
2. Cosine Similarity
3. Euclidean distance
4. Manhattan distance
5. Chebyshev distance
6. Minkowski distance

Textual Similarity – Jaccard Index

Wikipedia:
https://en.wikipedia.org/wiki/Jaccard_index
The **Jaccard index**, also known as **Intersection over Union** and the **Jaccard similarity coefficient** (originally given the French name *coefficient de communauté* by Paul Jaccard), is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Intersection and union of two sets A and B

Problem: How can we compute the similarity between texts?
Text1 = “This is example is about similarity”
Text2 = “This is example is about similarity too”
Text3 = “This is a sports news not example”

Textual Similarity – Jaccard Index

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Problem: How can we compute the similarity between texts?
T1 = “This is an example about similarity”
T2 = “This is an example about similarity too”
T3 = “This is a sports news not example”

J(T1, T2) = 6/7 = 0.857
J(T1,T3) = |(this, is, example)| / |(this, is, an, example, about, similarity, a, sports, news, not)| = 3/10 = 0.3

Textual Similarity – Cosine Similarity

Wikipedia: https://en.wikipedia.org/wiki/Cosine_similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1. The cosine of 0° is 1, and it is less than 1 for any angle in the interval $(0, \pi]$ radians. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at 90° relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. The cosine similarity is particularly used in positive space, where the outcome is neatly bounded in $[0, 1]$. The name derives from the term "direction cosine": in this case, unit vectors are maximally "similar" if they're parallel and maximally "dissimilar" if they're orthogonal (perpendicular). This is analogous to the cosine, which is unity (maximum value) when the segments subtend a zero angle and zero (uncorrelated) when the segments are perpendicular.

$$Cosine\ Similarity\ (A, B)$$

$$= \cos(\theta) = \frac{A.B}{||A||||B||} = \sum_{\{i=1\}}^{n} \frac{A_i B_i}{\sqrt{(\sum_{\{i=1\}}^{n} A_i^2 * \sum_{\{i=1\}}^{n} B_i^2)}}$$

Textual Similarity – Cosine Similarity

$$Cosine\ Similarity\ (A,B) = \cos(\theta) = \frac{A.B}{||A||||B||} = \sum_{\{i=1\}}^{n} \frac{A_i B_i}{\sqrt{(\sum_{\{i=1\}}^{n} A_i^2 * \sum_{\{i=1\}}^{n} B_i^2)}}$$

Problem: How can we compute the similarity between texts?
T1 = “This is an example”
T2 = “This is an example too”

Step 1. First create a list of all distinct words in T1 and T2:
D: {This, is, an, example, too}
Step 2. Create a DTM for each Ti by computing the occurrences of words in D in Ti:
DT1 = [1, 1, 1, 1, 0]
DT2 = [1, 1, 1, 1, 1]

Step 3. Compute similarity by using DT1 and DT2:
Similarity = (1*1 + 1*1 + 1*1 + 1*1 + 1*1 + 0*1)/sqrt((1^2 + 1^2 + 1^2 + 1^2 +1^2 + 0^2)* (1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2))
= 5/sqrt(5*6) = 0.91