Extracción de información

Tecnologías de búsqueda en la web

Marcelo Mendoza

Departamento de Informática
Universidad Técnica Federico Santa María

Preprocesamiento de texto

Preprocesamiento en NLTK

- Proyecto de Edward Loper, Ewan Klein, Steven Bird para python (Stanford).
- Sitio: http://www.nltk.org/
- Considera clases para NLP, soporte para POS, named-entity recognition, collocations.
- Clases más usadas: grammar, collocations, tag.
- Orientado hacia procesamiento NLP.

Natural Language Toolkit

Acceso a Corpus en NLTK

Procesamiento básico:

```
> import nltk
> nltk.download()
> from nltk.book import *
```

Búsqueda de texto:

```
> text1.concordance("whales")
```

Concordance muestra el contexto para "whales". Podemos buscar palabras usadas en contextos similares a "whales":

```
> text1.similar("whales")
```

Podemos buscar contextos similares entre pares de palabras:

```
> text1.common_contexts(["whales","pictures"])
```

Análisis Léxico en NLTK

Dispersión de palabras en un texto (usa numpy y matplotlib):

```
> text4.dispersion_plot(["freedom","democracy","America"])
```

Síntesis de texto usando estadísticas (frecuencias de palabras).

```
> text3.generate()
```

Consolidación de vocabulario:

```
> sorted(set(text3))
> len(set(text3))
```

Cuántas veces (en promedio) es usada cada palabra en el texto (lexical diversity)?:

```
> def lexical_diversity(text):
... return len(text) / len(set(text))
...
```


Python y texto

En Python el texto es una lista de palabras:

```
> sent1 = ["To","be","or","not","to","be"]
> len(sent1)
> lexical_diversity(sent1)
> sent2 = ["this","is","the","question"]
> sent1 + sent2
> sent2.append("or")
> sent1.count("be")
```

Buscar índice de palabra en lista:

```
> text4.index('freedom')
> text4[5740]
> text4[:10]
> text4[11:50]
```

El vocabulario es el conjunto de palabras del texto:

```
> vocab = sorted(set(text3))
> len(vocab)
```


Conteo de palabras en NLTK

Distribución de frecuencias.

```
> fdist1 = FreqDist(text1)
```

Fdist es una tabla <clave,valor>:

```
> vocab = fdist1.keys()
> vocab[:40]
> fdist1["whale"]
```

Visualizando la distribución de frecuencias (top-k).

```
> fdist1.tabulate(100)
> fdist1.plot(100)
```

Viendo la cola de la distribución (palabras con frecuencia 1):

```
> fdist1.hapaxes()
```


Normalización de texto

Minúsculas (Pure Python):

```
> vocab3 = sorted(set(text3))
> for w in vocab3:
... print w.lower()
...
```

Puntuación (Pure Python):

```
> sent3 = "What are you doing?"
> puncts = ".,?!"
> for sym in puncts:
... sent3 = sent3.replace(sym, ' ')
...
> sent3 = sent3.strip()
```

Stopwords (Python + NLTK).

```
> from nltk.corpus import stopwords
> stop = stopwords.words('english')
> tokens = sent3.split(' ')
> filtered = [token.lower() for token in tokens if token not in stop]
```

Bigramas, n-gramas

Bigramas:

```
> bigrams = bigrams(tokens)
```

N-gramas.

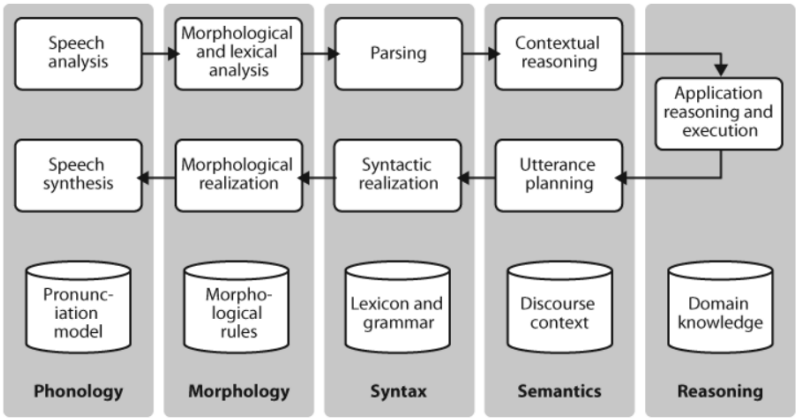
```
> from nltk.util import ngrams
> ngrams(tokens, 3)
```

N-gramas

Secuencias de n palabras. Son relevantes ya que algunas veces un n-grama tiene un significado distinto al de los unigramas que lo componen.

Algunos métodos para NLP

NLP pipeline



POS tagging

- Etiquetar cada término de acuerdo a la función que este cumple en el texto.
- Puede ayudarnos en tareas como detección de estilo, parsing, detección de colocaciones.
- Tarea importante en NLP.

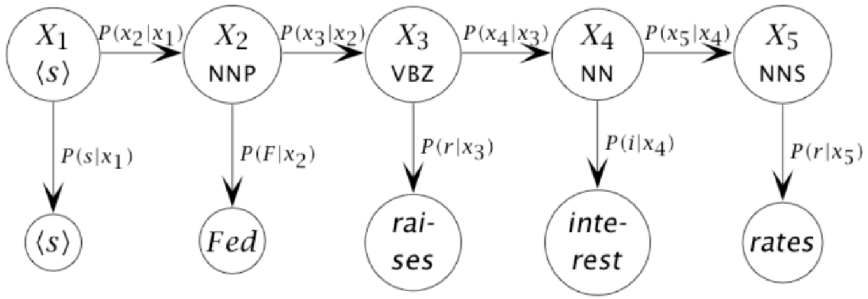
POS tags

PART-OF-SPEECH	TAG	EXAMPLES
Adjective	JJ	happy, bad
Adjective, comparative	JJR	happier, worse
Adjective, cardinal number	CD	3, fifteen
Adverb	RB	often, particularly
Conjunction, coordination	CC	and, or
Conjunction, subordinating	IN	although, when
Determiner	DT	this, each, other, the, a, some
Determiner, postdeterminer	JJ	many, same
Noun	NN	aircraft, data
Noun, plural	NNS	women, books
Noun, proper, singular	NNP	London, Michael
Noun, proper, plural	NNPS	Australians, Methodists
Pronoun, personal	PRP	you, we, she, it
Pronoun, question	WP	who, whoever
Verb, base present form	VBP	take, live

POS tagger basado en HMM

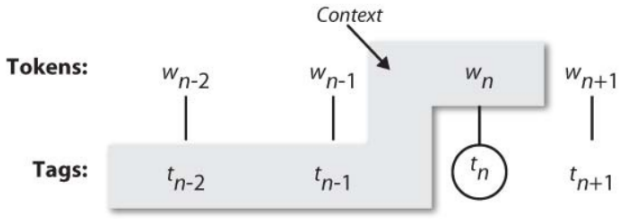
- Se dispone de un corpus etiquetado.
- La secuencia de tags es interpretada como una cadena de Markov: $P(x_{t+1} \mid x_t, \ldots, x_1) = P(x_{t+1} \mid x_t)$, $x_1, \ldots, x_{t+1}$ representan tags
- Usamos un modelo generativo para términos, con tags como estados ocultos: $P(t \mid x_1, \ldots, x_{t+1}) = P(t \mid x_{t+1})$

POS tagger basado en HMM



- En general muestran buena precisión (sobre 90%).

POS tagger basado en bigramas



- Considera los tags de las dos palabras precedentes.
- En general muestra mejor precisión que HMM.

Definiciones

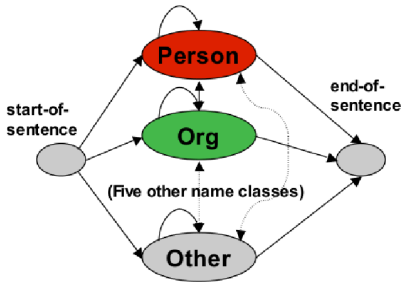
- Desambiguación de términos: un término, muchos significados (polisemia).
- Clustering de términos: varios términos, un significado.
- Collocations: términos que co ocurren tienen un significado distinto.

Colocaciones

- El significado conjunto es más que la suma de las partes (compositionality)
  1. Armas de destrucción masiva
  2. Strong tea
  3. Libre de sodio
  4. Intel inside
  5. Fast food
  6. Nuclear war
- Detectar colocaciones mejora la representación del contenido.
- Cada colocación puede ser procesada como un término.
- Se pueden detectar analizando co ocurrencias, etiquetando el par como una colocación si su co ocurrencia es mucho mayor que la esperada (azar, equiprobable).

Reconocimiento de entidades

- Tarea: Identificar entidades en texto (personas, organizaciones, etc.)
- Separa el text en chunks, y para cada cual asocia una NE. Opera sobre texto tagged.
- NER types: organization, person, location, date, time, money, percent, facility (human made artifacts), gpe (geo-political ents).
- POS tagging puede ayudar, agregando *entity* como un estado mas.



NLP en NLTK

Ejemplos simples NLP en NLTK

- Collocations (sobre texto completo)

```
> from nltk.book import *
> text3.collocations()
```

- POS tagger (a nivel de sentencia):

```
tagged = nltk.pos_tag(tokens)
```

- NER (sobre sentencia POS-tageada):

```
> nltk.chunk.ne_chunk(tagged)
```


Ejemplos simples NLP en NLTK

- Unigram Tagger Training

```
> from nltk.corpus import brown
> brown_tagged_sents = brown.tagged_sents()
> unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
> tagged = unigram_tagger.tag(text)
```

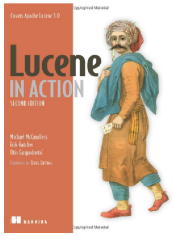
- Bigram Tagger Training

```
> bigram_tagger = nltk.BigramTagger(brown_tagged_sents)
> bigram_tagger.tag(text)
```


Herramientas para NLP

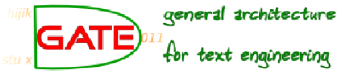
Lucene

- Proyecto apache para RI para desarrollo en Java (libre).
- Sitio: http://lucene.apache.org
- Tokenización con *Analyzer*.
- Clases más usadas: PorterStemmer (y en varios idiomas!!!), HTMLParser, entre otras.
- Orientado hacia procesamiento estándar IR.
- Libro:



Gate

- Proyecto de la University of Sheffield para RI para desarrollo en Java (ver licencias).
- Sitio: http://gate.ac.uk/
- Integra el core de Lucene y agrega clases para NLP.
- Clases más usadas: SentenceSplitter, Parser (con XML), POSTagger.
- Packages muy usados: wordnet, onthology, ontomatcher,
- Orientado hacia procesamiento IR + NLP.



LingPipe

- Proyecto de alias-i para desarrollo en Java (libre).
- Sitio: http://alias-i.com/lingpipe/index.html
- Considera clases para NLP, soporte para POS, named-entity recognition, collocations.
- Clases más usadas: HiddenMarkovModel, StringTagging, Tagger.
- Orientado hacia procesamiento NLP.

SOLR

- Proyecto apache para RI para desarrollo en Java (libre).
- Sitio: http://lucene.apache.org/solr/
- Con el foco en high performance, permite facets, acoplamiento con BDRs, geo tagging.
- Construido como una extensión de Lucene.

Sphinx

- Proyecto open source para RI para desarrollo en C++ (libre).
- Sitio: http://sphinxsearch.com/about/sphinx/
- Con el foco en acoplamiento con BDRs, permite índices en lotes y real-time, acoplamiento con BDRs mediante indexamiento de datos estructurados, consultas al estilo SQL.
- Es escalable (killer app. con 5 billones de docs).

Terrier

- Proyecto open source para RI para desarrollo en Java (libre).
- Sitio: http://terrier.org
- Con el foco en implementación de varios esquemas de ranking, permite evaluación de rendimiento, uso de becnhmarks (TREC).
- Es escalable.

Terrier

Libros