CSC 5930/9010: Text Mining

ANNIE Introduction

Dr. Paula Matuszek
Paula.Matuszek@villanova.edu
Paula.Matuszek@gmail.com
(610) 647-9789
Recap

• The goal of information extraction is to pull some well-defined information out of a large corpus of unstructured documents and put it in a more structured form, for easier access and understandability.

• Typically you have
  • a domain model
  • a knowledge model
  • an extraction engine
Knowledge-Based Approaches

- Systems like GATE use an approach based in natural language processing and modeling knowledge
  - domain model
  - knowledge model
- There are also machine-learning approaches using classifiers and sequence models
- And, of course, hybrid approaches. GATE includes some machine-learning based tools (not in ANNIE)
Domain Model

• Terms: enumerated strings which are members of some class

• Classes: Categories of terms, such as “locations”, “proteins”, “diseases

Both are often organized into an ontology

• Extraction rules
Terms, Classes, Ontologies

- MeSH is an example of a rich ontology in the medical domain; primarily used for searching, but it could also be used as the basis for an IE system.

- In ANNIE these are represented by gazetteer lists and by an index file which describes all the gazetteers, including their major type and possibly minor type and language.

- GATE also has a number of ontology plugins and tools.
Domain Rules

- LHS matches some pattern in the domain, RHS does something
- In ANNIE, RHS typically adds an annotation
Knowledge Model

- What you’re trying to extract
- Can be pretty simple and generic:
  - tuples of the form entity-relation-entity
  - entity:class relations
- Can be more detailed and specific:

<table>
<thead>
<tr>
<th>Org Name</th>
<th>Villanova University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Org Alias</td>
<td>Nova, Villanova</td>
</tr>
<tr>
<td>Org Description</td>
<td>“The local university”</td>
</tr>
<tr>
<td>Org Type</td>
<td>Educational Institution</td>
</tr>
<tr>
<td>Org Locale</td>
<td>Villanova, PA</td>
</tr>
</tbody>
</table>
Knowledge Model in ANNIE

- The knowledge model isn’t specified explicitly; it can basically be considered as all of the annotations which the various processing resources can produce. Each processing resource has an implicit knowledge model.

- Additional models are defined in JAPE, which can create additional annotations.
Extraction Engine

- Tool which applies rules to text and extracts matches
  - Tokenizer
  - Part of Speech (POS) Tagger
  - Term and class tagger
  - Rule engine: match LHS, execute RHS
- Rule engine is iterative
- May include an interactive component which is essentially a query engine against already extracted information
The entire GATE system can be considered as an extraction engine (although it does a lot more)

Most of the processing resources in ANNIE and many others available in CREOLE are specific extraction engines

Some systems (eg, AeroText, OpenCalais) make an explicit distinction among the domain model, the knowledge model, and the extraction engine.

The distinction in ANNIE is less clear
On to ANNIE:

• A family of Processing Resources for language analysis included with GATE

• Stands for A Nearly-New Information Extraction system.

• Using finite state techniques to implement various tasks: tokenization, semantic tagging, verb phrase chunking, and so on.

• (LaSIE is the forerunner of ANNIE, focused specifically on information extraction for the TREC conferences)
ANNIE IE Modules

http://gate.ac.uk/sale/tao/splitch6.html#chap:annie
ANNIE Standard Components

- These are what is loaded when you load ANNIE and run the default application
  - Document Reset
  - Tokenizer
  - Gazetteer: lists of entities
  - Sentence Splitter/Regex sentence splitter
  - Part of Speech Tagger
  - Named Entity Transducer
  - Orthomatcher
Document Reset

• Task: reset a document to its original state. Releases old annotations for garbage collection.

• Parameters:
  • annotationTypes: specify annotations to remove. Default is all.
  • setsToKeep:
  • setsToRemove:
  • keepOriginalMarkupsAS: Boolean.
ANNIE Component: Tokenizer

- Task: split text into simple tokens
- In ANNIE, used as one piece of the English Tokenizer
- Five types of tokens, some of which have attributes or subtypes
- Uses tokenizer rules
  - left hand side (LHS): pattern to be matched
  - right hand side (RSH): action to be taken
Default Token Types

- **Token Types.** All have attributes string and length

- **Tokens**
  - **word.** Any contiguous set of upper or lowercase letters, including a hyphen.
    - orthography attribute (orth): upperInitial, allCaps, lowerCase, mixedCaps
  - **number.** Any combination of successive digits.
  - **symbol.** Currency symbols, ^, +, =, etc..
    - symbolkind attribute: currency
  - **punctuation.**
    - position attribute (position) startpunct, endpunct
    - subkind attribute (subkind) dashpunct

- **SpaceTokens**
  - **space.** any non-control character
  - **control.** any control character
Tokenizer Rule

- Operations used on the LHS:
  - \(|\) (or)
  - \(\ast\) (0 or more occurrences)
  - \(\?\) (0 or 1 occurrences)
  - \(+\) (1 or more occurrences)

- The RHS uses ‘;’ as a separator, and has the following format:
  \{LHS\} > \{Annotation type\};
  \{attribute 1\}={value 1};...;\{attribute  n\}={value  n}
Example Tokenizer Rule

"UPPERCASE_LETTER" "LOWERCASE_LETTER"*

> 

Token;orth=upperInitial;kind=word;

- The sequence must begin with an uppercase letter, followed by zero or more lowercase letters. This sequence will then be annotated as type “Token”. The attribute “orth” (orthography) has the value “upperInitial”; the attribute “kind” has the value “word”.
English Tokenizer

• Task: adapt generic tokenizer for POS tagger by dealing with constructs involving apostrophes
  - don’t --> do + n’t
  - you’ve --> you + ‘ve

• Uses a JAPE transducer: Java Annotation Patterns Engine.
  - LHS: annotation pattern description
  - RHS: annotation manipulation statement

• If you’re going to use the POS tagger, always use this tokenizer
ANNIE Component: Gazetteer

- **Task:** identify entity names based on lists
  - Located in `<GATEhome>/plugins/ANNIE/resources/gazetteer`
  - Defined in `lists.def`, which gives name, major type, minor type, language

- **Example lists.def entries:**
  - `airports.lst:location:airport`
  - `city.lst:location:city`
  - `festival.lst:date:festival`
  - `day.list:date:day`
Gazetteer lists

- The gazetteer lists used are plain text files, with one entry per line.

- Typical lists include
  - named entities, such as locations, organizations, dates, names,
  - grammatical entities such as determiners
  - components of entities, such location prefixes
  - anything else that can usefully be enumerated

- Don’t always have a minor type or language. Many are one-off. (spur_ident)
Gazetteer Parameters

• **Init-time parameters**
  - listsURL: index file. Default is lists.def
  - encoding: default is UTF-8
  - gazetteerFeatureSeparator: Not required.
  - caseSensitive (Boolean). Default is true.

• **Run-time parameters**
  - document to process
  - annotationSetName: what to call the Lookup annotation set. Not required.
  - wholeWordsOnly: Default is true
  - longestMatchOnly: Default is true. EG: Amazon UK
Gazetteer Editor

- You can add or modify
  - the name of a list
  - the major and minor categories in the list
  - an entry in the list
- You can delete a list by right-clicking the name
- You can also add a new list
- Or edit everything outside of GATE with a text editor.
ANNIE Component: Sentence Splitter

• Task: just what it says: segments the text into sentences.

• The splitter uses a gazetteer list of abbreviations to help distinguish sentence-marking full stops from other kinds.

• This module is required for the POS tagger.
POS Tagger

- Modified version (Hepple) of the Brill tagger, probably the best-known of the POS taggers
  - produces a POS tag as an annotation on each word or symbol
  - Uses a default lexicon and rule set
    - alternate lexicons for all upper-case and all lower-case corpora.
- Must run English Tokenizer and Sentence Splitter first
Partial List of Hepple (Penn treebank) Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>cardinal number</td>
</tr>
<tr>
<td>IN</td>
<td>preposition or subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>adjective: Hyphenated compounds that are used as modifiers; happy-go-lucky.</td>
</tr>
<tr>
<td>JJR</td>
<td>adjective - comparative: Adjectives with the comparative ending ‘-er’.</td>
</tr>
<tr>
<td>JJS</td>
<td>adjective - superlative: Adjectives with the superlative ending ‘-est’ (and ‘worst’).</td>
</tr>
<tr>
<td>NN</td>
<td>noun - singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>noun - plural</td>
</tr>
<tr>
<td>NP</td>
<td>proper noun - singular</td>
</tr>
<tr>
<td>NPS</td>
<td>proper noun - plural</td>
</tr>
<tr>
<td>POS</td>
<td>possessive ending: Nouns ending in ‘s’ or ‘’</td>
</tr>
<tr>
<td>PP</td>
<td>personal pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>adverb: most words ending in ‘-ly’. Also ‘quite’, ‘too’, ‘very’, others</td>
</tr>
<tr>
<td>RBR</td>
<td>adverb - comparative: adverbs ending with ‘-er’ with a comparative meaning.</td>
</tr>
<tr>
<td>RBS</td>
<td>adverb - superlative</td>
</tr>
<tr>
<td>SYM</td>
<td>symbol: technical symbols or expressions that aren’t English words.</td>
</tr>
<tr>
<td>VBD</td>
<td>verb - past tense</td>
</tr>
<tr>
<td>VBG</td>
<td>verb - gerund or present participle</td>
</tr>
<tr>
<td>VBN</td>
<td>verb - past participle</td>
</tr>
<tr>
<td>VBP</td>
<td>verb - non-3rd person singular present</td>
</tr>
<tr>
<td>VB</td>
<td>verb - base form: subsumes imperatives, infinitives and subjunctives.</td>
</tr>
<tr>
<td>VBZ</td>
<td>verb - 3rd person singular present</td>
</tr>
</tbody>
</table>
Example Lexicon Entries

- '30s CD NNS
- & CC SYM
- 1995-1999 CD
- Announcement NN
- Figuring VBG
- Frequently RB
- ...
- A total of over 17,000 entries
Brill Tagger

• A Brill tagger starts by assigning all the known words from the lexicon.

• It next assigns NNP to capitalized unknown words and NN to everything else unknown.

• Finally, a bunch of replacement rules change the assignment based on various features and context.
Example Brill Tagger Rules

- replace(pos 'NN' 'JJ') # [suffix#less#[0]]
  "replace tag NN with JJ if the word in question ends in "less"."

- replace(pos 'VB' 'NN') # [canHave#'NN'#[0] pos#'DT'#[~1]]
  "replace tag VB with NN if the the word in question can
  have tag NN (according to the lexicon) and if the previous
  word is tagged DT"

LOTS of Parameters

- **Init-Time:** all optional
  - encoding - encoding to be used for reading rules and lexicons
  - lexiconURL - The URL for the lexicon file
  - rulesURL - The URL for the ruleset file

- **Run-time:**
  - document - The document to be processed. Required
  - inputASName - Input annotation set. Optional
  - outputASName - Output annotation set. Optional
  - baseTokenAnnotationType - name of annotation type for tokens (default = Token). Required.
  - baseSentenceAnnotationType - name of annotation type for sentences (default = Sentence). Required.
  - outputAnnotationType - default = Token. Required.
  - failOnMissingInputAnnotations - Default is true.
Named Entity Transducer

• Task: find terms that suggest entities

• Hand-crafted grammars define patterns over the annotations

• Written in JAPE

• Doesn’t require that you run anything else, but doesn’t do much if you don’t already have annotations.

• Adds new token types
Required Parameters

• Initialization
  • encoding. default is UTF-8
  • grammar. default is `<GATEhome>/plugins/ANNIE/resources/NE/main.jape` (which loads a bunch more)

• Runtime
  • Corpus name (will default if there’s only one)
Example JAPE rules

Phase: Unknown

Input: Location Person Date Organization Lookup

Rule: Known
Priority: 100
( {Location} | {Person} | {Date} | {JobTitle}) : known
--> 
{}

Rule: Unknown
Priority: 50
( {Token.category == NNP}) : unknown
--> 
: unknown.Unknown = {kind = "PN", rule = Unknown}

(This is only a part of the actual rules)
Orthomatcher

- Task: add co-reference information to Named Entities based on orthographic information. In a typical text, a NE may be referred to multiple times, with abbreviations and acronyms
  - Creates match lists for tagged NEs
  - May assign an entity type to previously unclassified entities
  - Won’t reassign already classified entities
- Without NE Transducer annotations doesn’t actually do anything.
Required Parameters

- **Required Init-time**
  - definitionFileURL. Default is `<GATEhome>/plugins/ANNIE/resources/othomatcher/listsNM.def`.
  - encoding. Default is UTF-8.
  - minimumNicknameLikehihood: default is 0.5.

- **Run-time**
  - corpus or document
  - Can also have: AnnotationTypes: list of types to be processed. Default: Organization, Person, Location, Date. Not required.
Co-reference Editor

- Can modify the reference chains within a document
  - Delete an existing member of the chain
  - Add a new item to a chain
- This will change all relevant chains: adding an item to a chain will change the chain for all the items in it
- This will not change NE types or other annotations; it doesn’t propagate.
Some Additional ANNIE Modules

- These modules are not included in the default ANNIE load, but they are available in processing resources
  - Morphological analyzer (stemmer). Add “root” annotation
  - Pronominal Coreference. Add “coreference” annotation
- May need to load tools from CREOLE manager first
Summary

- The default ANNIE application uses a sequence of processing resources to create annotations over unstructured text.

- The tokenizer, sentence splitter, POS tagger and orthomatcher are (mostly) domain-independent.

- The gazetteer lists and Named Entity Transducer are domain-dependent; ANNIE comes with a reasonably good default set.

- The domain information is largely captured in gazetteer lists and JAPE rules.