Information Extraction and Question-Answering Systems

Foundations and methods

Dr. Günter Neumann
LT-Lab, DFKI
neumann@dfki.de

What the lecture will cover

- Machine Learning for IE
- Lexical processing
- Evaluation Methods
- Basic Terms & Examples
- Parsing of Unrestricted Text
- Generic NL Core system
- Domain Modelling
- Question/Answering Core components
- Advanced Topics

Machine Learning for IE

- Recently, the problem of using machine learning methods to induce IE routines has received more attention
- The majority of current approaches are variants of inductive supervised learning
- Goal: given a set of annotated text documents induce automatically template filler rules by successively generalizing the initialized instantiated rules computed from the tagged examples

Machine Learning for IE

- Usually, the documents are preprocessed by NL components
  - tokenization (Freitag, 98)
  - POS tagging (Califf&Mooney,98)
  - phrase recognition (Hufman,96)
  - shallow sentence parsing (Riloff,96a;Soderland,97)
- most approaches learn slot-filler rules, some newer learn relational structures (Califf&Mooney,98)
- current trend is towards minimally supervised strategies (Riloff,96b)

An initial example

<PNG> Sue Smith </PNG>, 39, of Menlo Park, was appointed <TNG> president </TNG> of <CNG> Foo Inc. </CNG>

n_was_named_t_by_c:

noun-group(PNG, head(isa(person-name))),
noun-group(TNG, head(isa(title))),
noun-group(CNG,head(isa(company-name))),
verb-group(VG, type(passive), head(named or elected or appointed)),
prep(PREP, head(of or at or by)),
subject(PNG,VG), object(VG,TNG), post_nominal_prep(TNG,PREP), prep_obj(PREP,CNG)

⇒ management_appointment(M,person(PNG), title(TNG), company(CNG))

Machine Learning for IE

- Current approaches show already impressive results (for flat sentence-based templates):
  - Huffman,96: management changes task => 85.2% F (89.4%)
  - Califf&Mooney,98 : computer-related job postings => 87.1% P & 58.8% R

Overview of different approaches

- We review extraction patterns that are used only to process documents that contain grammatical, plain text.
- Such extraction rules are based on syntactic and semantic constraints that help identify the relevant information within a document.
- Consequently, in order to apply the extraction patterns below, one has to pre-process the original text with a syntactic analyzer and a semantic tagger.

AutoSlog (E. Riloff, 1996)

- AutoSlog builds a dictionary of extraction patterns that are called concepts or concept nodes.
- Each concept C consists of:
  - conceptual anchor (triggering word) that activates C
  - linguistic pattern, which, together with the set of enabling conditions, guarantees C's applicability.
    - Enabling conditions represent constraints on the components of the linguistic pattern.

AutoSlog example

- Goal: extract the target of the terrorist attack from the sentence
  - The Parliament was bombed by the guerrillas.
- Concept:
  - triggering word: *bombed*
  - linguistic pattern: *<subject> passive-verb*
- Application:
  - Parse sentence (determines linguistic categories like subject, object, PP, verb form)
  - Activate concept by matching the verb form *bombed*
  - Match linguistic pattern against sentence and the *subject* is extracted as the *target* of the terrorist attack

Determination of concept dictionary

1. Run a text through the CIRCUS (UMass) parser and identify information that should have been extracted but was not (the „targeted" information).
2. Determine whether the targeted information was the subject of a clause, the direct object, or a PP.
3. Determine which word in the sentence was the strongest indicator that the information should have been extracted. Use this word as the trigger word for a concept word.
4. Create a concept node that is activated by the triggering word in the same immediate context, and extract information from the syntactic constituent identified in step (2).

Step 3 seems like the most difficult to automate

- Trigger words can be reliabley identified using simple linguistic rules.
- Based on linguistic category of targeted NP, identify restriction on linguistic category of triggering word.
- Predefine generic linguistic pattern rules.

Predefined linguistic patterns

Each rule generates an expression that likely defines the conceptual role of the NP.
An extraction pattern is created by instantiating the rule with the specific words that it matched in the sentence.
The rules are ordered so the first one that is satified generates an extraction pattern with the longest patterns being tested before the shorter ones.

Pattern	**Example**
<subj> passive-verb	<victim> was murdered
<subj> active-verb	<perp> bombed
<subj> verb infin.	<perp> attemped to kill
<subj> aux noun	<victim> was victim
Passive-verb <dobj>	Killed <victim>
Active-verb <dobj>	Bombed <target>
infin. <dobj>	To kill <victim>
Verb infin. <dobj>	Tried to attack <target>
Gerund <dobj>	Killing <victim>
Noun aux <dobj>	Fatality was <victim>
Noun prep <np>	Bomb against <target>
Active-verb prep <np>	Killed with <target>
Passive-verb prep <np>	Was aimed at <target>

AutoSlog is a supervised ML approach

- Example text annotations for Autoslog (adapted)
  It was officially reported that a policeman was wounded today when urban guerrillas attacked the guards at a power substation located in downtown San Salvador.
  - A policeman=injury victim
  - urban guerrillas=attack perpetrator
  - the guards=attack victim
  - San Salvador=attack location

Overview of training

- For each targeted NP in the training corpus, Autoslog identifies the sentence from which it should be extracted.
- Autoslog makes the assumption that the first sentence containing the NP is the one from which ist should have been extracted.
- Parse sentence with CIRCUS parser
  - Assigns each NP to one of three syntactic categories: subject, direct object, PP
- Apply heuristics to identify triggering words
  - Should be the word that determines the conceptual role of NP
  - Apply predefined linguistic patterns according to longest match
  - If NP is part of PP apply specialized PP-attachment algorithm to attach PP to preceding noun or verb

Examples of learned concept node

CONCEPT NODE:

Name: target-subject-passive-verb-bombed
Trigger: bombed
Variable Slots: (target (Subj 1))
Constraints: (class phys-target Subj)
Constant Slots: (type bombing)
Enabling Conditions: (passive)

Example sentence:

In La Oroya, Junin department, in the central Peruvian mountain range, public buildings were bombed and a car-bomb was detonated.

Examples of learned concept node

CONCEPT NODE:

Name: perpetrator-subject-verb-infinitive-threatened-to-murder
Trigger: murder
Variable Slots: (perpetrator(Subj 1))
Constraints: (class perpetrator Subj)
Constant Slots: (type perpetrator)
Enabling Conditions: ((active)
(trigger-preceded-by `threatened `to)

Example sentence:

The Salvadoran guerrillas today threatened to murder individuals involved in 19 March presidential elections if they do not resign from their posts.

Evaluation on MUC-4 corpus

- Created a concept dictionary for the MUC-4 terrorism domain using AutoSlog and compare it with the hand-crafted dictionary used in MUC-4.
- 722 texts (marked with answer keys) used for training
- Contained 4780 tagged NPs corresponding to six types
  1. Human target description, „a security guard“
  2. Human target name, „Ricardo Castellar“
  3. Instrument id, „car-bomb“
  4. Perpetrator individual, „a group of subversives“
  5. Perpetrator organisation, „the FMLN“
  6. Physical target id, „car dealership“
- AutSlog generated 1237 unique concept node definition
- Human in the loop filtered out 787 unreliable concept nodes (easy job, took only 5 hours for untrained student)

Comparision with hand-made system

- Official Umass/MUC-4 system
  - One instance using orginal dictionary
  - One instance using AutSlog dictionary
- Perform evaluation of both systems on two blind test sets of 100 text each

System/Test set	Recall	Precision	F-measure
MUC-4/TST3	46	56	50.51
AutoSlog/TST3	43	56	48.65
MUC-4/TST4	44	40	41.90
AutoSlog/TST 4	39	45	41.79

AutSlog achieved 98% of the performance of a dictionary that was built manually, with substantially less time required for knowledge enineering

LIEP (S. Huffman, 1995)

- LIEP is a learning system that generates multi-slot extraction rules
  - Rather then learning one extraction pattern for each item of interest in a sentence (e.g., target and perpetrator), LIEP generates a single rule for all items of interest.
  - An extraction rule corresponds to an event, basically consisting of a group of entities (realized as noun groups) which stand in a certain relationship (expressed via verb group)
- Example:

  The Parliament was bombed by the guerrillas.

```
TARGET-was-bombed-by-PERPETRATOR:
      noun-group( TRGT, head( isa(physical-target) ) ),
      noun-group( PERP, head( isa(perpetrator) ) )
      verb-group( VG, type(passive), head(bombed) )
      preposition( PREP, head(by) )
      subject( TRGT, VG ), post-verbal-prep(VG, PREP ),
      prep-object( PREP, PERP )
=> bombing-event( BE, target(TRGT), agent(PERP) )
```


The extraction system

- The extraction system that LIEP learns extraction patterns for is called ODIE (for On Demand Information Extractor).
- Given an input text
  - tokenizes the text and breaks it into sentences.
  - For each sentence, ODIE checks whether the sentence contains any of a set of keywords that indicate the possibility that the sentence expresses an event of interest.
  - If so, the words in the sentence are tagged with their POS.
  - Next, a set of pattern-matchers run over the sentence to identify entities of interest (for management changes, this is people, company names, and management titles) and contiguous syntactic constituents (noun groups, verb groups, and prepositions).

Continue

- Next, ODIE applies a set of information extraction patterns
- Patterns match syntactic constituents by testing their head words/entities and other simple properties (e.g. active/passive for verb groups), and attempt to verify syntactic relationships between the constituents. If all of the syntactic relationships are verified, an event is logged.
- ODIE performs a partial parsing: it verifies the plausibility of specific syntactic relationships between pairs of constituents tested in extraction patterns. For instance,
  - subject(ng,vg) if ng is directly to the left of vg, or if ng is further to the left, and everything in between ng and vg could possibly be a right-modifier of ng (e.g., prepositional phrases, comma-delimited strings of words like relative clauses, parentheticals, etc.)
  - Note that locallity of test are unsecure (but robust)

Learning extraction patterns

- LIEP is a supervised learning method using a set of training exambles annotated with corresponding template tags.
- LIEP tries to build a set of extraction patterns that will maximize the number of extractions of positive examples and minimize spurious extractions.
- Given a new example that is not already matched by a known pattern, LIEP first attempts to generalize a known pattern to cover the example.
- If generalization is not possible or fails to produce a high-quality pattern, LIEP attempts to build a new pattern based on the example.

Building extraction patterns

- LIEP creates potential patterns from an example sentence/ event by searching for sets of relationships that relate all of the role-filling constituents in the event to one another.
- Example „Bob was named CEO of Foo Inc.“
  - Three constituents: Bob, CEO, Foo Inc.
  - LIEP attempts to find paths of relationships between each pair of consituents, (Bob, CEO), (Bob, Foo Inc), (CEO, Foo Inc)
  - then merges those paths to create sets of relationships relating all three
- Relationship can be direct, e.g., subject(ng,vg), or
- Indirect where constituents start/end of a path, e.g., (subject(Bob,named),object(named,CEO))

Algorithm for identifying relationships between entities

```
Find_relationships(C1,C2) {
    if direct_relationship(C1, C2, R) then return(R)
    else
    while (choose_next_intermediate_constituent(CIntermediate)) {
          Rels1 = find_relationships(C1,CIntermediate)
          Rels2 = find_relationships(C2,CIntermediate)
          return(Rels1 + Rels2)}
        else failure}
```

choose_next_intermediate_constituent(CIntermediate):
selects intermediate constituents to use, starting from the rightmost constituent between the two being related, and moving left to the beginning of the sentence.

Multiple paths

- In many cases, there are multiple paths of relationships between a pair of constituents. The multiple paths very roughly correspond to multiple syntactic parses of the sentence.
- Example above „of Foo Inc." could modify the verb named or the noun CEO. Thus, Bob and Foo Inc. are related by both:
  - [subject(Bob,named),object(named,CEO), post_verbal_post_object_prep(named,of), prep_object(of,Foo Inc.)]
  - and: [subject(Bob,named),object(named,CEO), post_nominal_prep(CEO,of), prep_object(of,Foo Inc.)]
- LIEP does not reason about what „Foo Inc." modifies; it simply generates both of the possibilities because ODIE's plausible syntactic knowledge indicates that both postverbal post object prep(named,of) and post nominal prep(CEO,of) hold.

Algorithm for building new patterns

```
Build_new_pattern(Example) {
    HighestAccuracy = 0, Result = failure
    do 3 times {
    Rels = find_relationships_between_role_fillers(Example)
    if (Rels != failure) then {
        Pattern = create_pattern_from_relationships(Rels)
        Acc = compute_f_score_on_old_examples(Pattern)
        if Acc > HighestAccuracy then {
            HighestAccuracy = Acc
            Result = Pattern }}}
    return(Result)}
```

Calls find_relationships for each pair of roles in the example event, and merges the resulting sets of relationships. Calling it multiple times causes Find_relationships to backtrack and find multiple paths between constituents if they exist.

Example
„Bob was named CEO of Foo Inc."

- First set of relationships Find_relationships_between_role_fillers() finds relating Bob, CEO, and Foo Inc. is:
  - [subject(Bob,named),object(named,CEO), post_verbal_post_object_prep(named,of), prep_object(of,Foo Inc.)]
- Create_pattern_from_relationships() creates the pattern: LIEP_pattern1:
    - noun-group(PNG,head(isa(person-name))),
    - noun-group(TNG,head(isa(title))),
    - noun-group(CNG,head(isa(company-name))),
    - verb-group(VG,type(passive),head(named)),
    - preposition(PREP,head(of)),
    - subject(PNG,VG),
    - object(VG,TNG),
    - post_verbal_post_object_prep(VG,PREP),
    - prep_object(PREP,CNG)
  - => management_appointment(M, person(PNG),title(TNG),company(CNG)).
- After up to three such patterns are constructed, they are compared by running them on all the example sentences LIEP has seen so far. The pattern with the highest F-measure is returned and added to the system's dictionary.

Generalizing patterns

- Often, later training examples have the same syntactic relationships as previously learned pattern, but with dierent constituent head words or properties.
- LIEP assumes that non-role-filler constituents' head words and properties within a pattern can be generalized, but that constituents' syntactic types and relationships (syntactic footprint) should not be generalized.
- LIEP makes use of special versions of the patterns that test only the „ syntactic footprint“, i.e., the non-generalizable parts.

Specialized patterns

- LIEP_pattern1 (NON-GENERALIZABLE-PORTION)
  - noun-group(PNG,head(isa(person-name))),
  - noun-group(TNG,head(isa(title))),
  - noun-group(CNG,head(isa(company-name))),
  - verb-group(VG), preposition(PREP)

  - subject(PNG,VG),
  - object(VG,TNG),
  - post_verbal_post_object_prep(VG,PREP),
  - prep_object(PREP,CNG)
  - => matches_positive_example(
    person(PNG),title(TNG),company(CNG)).

Example

- „Joan has been appointed vp, finance, at XYZ Company.“
  - Same syntactic relationships between person, title, company NGs
  - Create generalization of pattern1 by inserting disjunctive values within each generalizable test in the pattern
- Generalized version of pattern1: next slide

Generalized pattern

```
Gen1_LIEP_pattern1:
      noun-group(PNG,head(isa(person-name))),
      noun-group(TNG,head(isa(title))),
      noun-group(CNG,head(isa(company-name))),
      verb-group(VG, type(passive),
      head(member(genclass1))),
      preposition(PREP, head(member(genclass2))),
      subject(PNG,VG),
      object(VG,TNG),
      post_verbal_post_object_prep(VG,PREP),
      prep_object(PREP,CNG)
==> management_appointment(M,
            person(PNG),title(TNG),company(CNG)).
```

genclass1=(named,appointed)
genclass2 = (of,at)

Evaluation

- Test corpus of 300 naturally-occurring texts reporting management changes. Each text contained 1 or 2 complex sentences.
- ODIE's average F-measure using a hand-built set of patterns (100 test texts):
  - 89.4% (recall 85.9%; precision 93.2)
- LIEP's average f-measure (150 test texts):
  - 85.2% (recall 81.6%; precision 89.4%)

RAPIER (CaliffMooney,2002)

- Approaches so far are used as part of a larger IE system
- Rapier learns rules for complete IE tasks directly from document without prior (shallow) parsing
- Of course, so far only used for English. Open question:
  What about free word order languages?

What does Rapier learn?

- Shallow patterns that make very limited syntactic and semantic information, basically POS taggers & lexica
- The rules built from these patterns can consider an unbounded context, giving them an advantage over more limited representations which consider only a fixed number of words.
- In order to do so, Rapier employs a relational learning algorithm which uses techniques from several Inductive Logic Programming (ILP) systems.
- Core approach: primarily a specific to general (bottom-up) search.

Domain: job posting from newsgroups

Subject: US-TN-SOFTWARE PROGRAMMER

Date: 17 Nov 1996 17:37:29 GMT

Organization: Reference.Com Posting Service

Message-ID: <56nigp$mrs@bilbo.reference.com>

SOFTWARE PROGRAMMER

Position available for Software Programmer experienced in generating software for PC-Based Voice Mail systems. Experienced in C Programming. Must be familiar with communicating with and controlling voice cards; preferable Dialogic, however, experience with others such as Rhetorix and Natur Microsystems is okay. Prefer 5 years or more experience with PC Based Voice Mail, but will consider as little as 2 years. Need to find a Senior level person who can come on board and pick up code with very little training. Present Operating System is DOS. May go to OS-2 or UNIX in future.

Please reply to:

Kim Anderson

AdNET

(901) 458-2888 fax

kimander@memphisonline.com

Filled template

computer_science_job
id: 56nigp$mrs@bilbo.reference.com
title: SOFTWARE PROGRAMMER
salary:
company:
recruiter:
state: TN
city:
country: US

language: C
platform: PC \ DOS \ OS-2 \ UNIX
application:
area: Voice Mail
req_years_experience: 2
desired_years_experience: 5
req_degree:
desired_degree:
post_date: 17 Nov 1996

By way:
domain is evident;
see FlipDog, a job posting webside, developed by WhizBang!
www.whizbanglabs.com

Relational Learning

- In order to accurately estimate probabilities from limited data, most statistical techniques base their decisions on a very limited context, such as bigrams or trigrams (2 or 3 word contexts). However, NLP decisions must frequently be based on much larger contexts that include a variety of syntactic, semantic, and pragmatic cues.
- Relational learning methods allow induction over structured examples that can include first-order logical predicates and functions and unbounded data structures such as lists and trees. Inductive logic programming (ILP) studies the induction of rules in first-order logic (Prolog programs).
- While Rapier is not an ILP system, it is a relational learning algorithm learning a structured rule representation, and its algorithm was inspired by ideas from ILP systems.

Two sorts of systems

- compression: Systems begin by creating an initial set of highly specific rules, typically one for each example.
  - At each iteration a more general rule is constructed, which replaces the rules it subsumes, thus compressing the rule set.
  - At each iteration, all positive examples are under consideration to some extent, and the metric for evaluating new rules is biased toward greater compression of the rule set.
  - Rule learning ends when no new rules to compress the rule set are found.
- Covering: Systems begin with a set of positive examples.
  - Then, as each rule is learned, all positive examples the new rule covers are removed from consideration for the creation of future rules.
  - Rule learning ends when all positive examples have been covered.

Rapier: Rule representation

- The extraction rules are indexed by template name and slot name and consist of three parts:
  - a pre-filler pattern: matches text immediately preceding the filler,
  - a pattern: must match the actual slot filler,
  - a post-filler pattern: must match the text immediately following the filler
- Each pattern is a sequence (possibly of length zero in the case of pre- and post-filler patterns) of pattern elements.
- two types of pattern elements: pattern items and pattern lists
- A pattern list species a maximum length N and matches 0 to N words or symbols from the document, each of which must match the list's constraints
- Constraints: POS or semantic class of words in a pattern

Example

- A rule for extracting the transaction amount from a newswire concerning a corporate acquisition:
  - sold to the bank for an undisclosed amount
  - paid Honey- well an undisclosed price".

Pre-filler pattern:	**Filler pattern:**	**Post-filler pattern:**
1) syntactic: {nn,nnp}	**1) word: undisclosed**	**1) semantic: prices**
2) list: length 2	**syntactic: jj**	

- Rapier uses Brill‘s tagger and WordNet synsets as semantic classes. Both are freely available.

Rapier's design decisions

- Compression-based and primarily consists of a specific to general (bottom-up) search.
- Bottom-up approach: prefer overly specific rules to overly general ones
  - A bottom-up approach will tend to produce specific rules, which also tend to be precise rules.
- Rapier begins with a most specific definition and then attempts to compact that definition by replacing rules with more general rules.
- Since in Rapier's rule representation rules for the different slots are independent of one another, the system actually creates the most specific definition and then compacts it separately for each slot in the template (order independence wrt. Slots)

Initial Rulebase construction

- Most-specific patterns for each slot are created for each example, specifying words and tags for the filler and its complete context.
- Pre-filler contains an item for each word from the beginning of the document to the word immediately preding the filler
- Filler has one item from each word in the filler (tagged text fragment).
- Post-filler has one item for each word from the end of the filler to the end of the document

Generalization

- New rules are created by selecting pairs of existing rules and creating generalizations.
- Assumption: the relevant information for extracting a slot-filler will be closer to that filler in the document.
- Rapier begins by generalizing the two filler patterns and creates rules with the resulting generalized filler patterns and empty pre-filler and post-filler patterns.
- It then specializes those rules by adding pattern elements to the pre-filler and post-filler patterns, working outward from the filler.
- The elements to be added to the patterns are created by generalizing the appropriate portions of the pre-fillers or post-fillers of the pair of rules from which the new rule is generalized.
- Advantage: take into account of NL locality without disregarding fairly distant elements

Algorithm for rule induction

```
For each slot, S in the template being learned
    SlotRules = most specific rules for S from example documents
    while compression has failed fewer than CompressLim times
        initialize RuleList to be an empty priority queue of length k
        randomly select M pairs of rules from SlotRules
        find the set L of generalizations of the fillers of each rule pair
        for each pattern P in L
            create a rule NewRule with filler P and empty pre- and post-fillers
            evaluate NewRule and add NewRule to RuleList
        let n = 0 loop
            increment n
            for each rule, CurRule, in RuleList
                NewRuleList = SpecializePreFiller (CurRule, n)
                evaluate each rule in NewRuleList and add it to RuleList
            for each rule, CurRule, in RuleList
                NewRuleList = SpecializePostFiller (CurRule, n)
                evaluate each rule in NewRuleList and add it to RuleList
            until best rule in RuleList produces only valid fillers or the value of the best rule in
                RuleList has failed to improve over the last LimNoImprovements iterations
            if best rule in RuleList covers no more than an allowable percentage of spurious fillers
                add it to SlotRules and remove empirically subsumed rules
```

Rule creation: example for City slot

Ex 1: „located in Atlanta, Georgia.“

Pre-filler pattern:	Filler pattern:	Post-filler pattern:
1) word: located tag: vnb	1) word: atlanta tag: nnp	1) word: , tag: , 2) Word: georgia tag: nnp
2) word: in tag: in		3) Word: . tag: .

Ex 2: „offices in Kansas City, Missouri.“

Pre-filler pattern:	Filler pattern:	Post-filler pattern:
1) word: offices tag: nns	1) word: kansas tag: nnp	1) word: , tag: , 2) Word: missouri tag: nnp
2) word: in tag: in	2) Word: city tag: nnp	3) Word: . tag: .

Example continued

Fillers are generalized to produce two possible rules with empty pre/post fillers, one rule with *words disjuncted* and one rule with *words dropped*. Because one rule has two elements and the other one, a list of length 2 is created.

Pre-filler pattern:	Filler pattern:	Post-filler pattern:
	1) list: len: 2 word: [atlanta, kansas, city] tag: nnp	

Pre-filler pattern:	Filler pattern:	Post-filler pattern:
	1) list: len: 2 tag: nnp	

Example continued

The new rules are likely to cover spurious example (mainly because empty pre/post fillers), so further specialization:

Pre-filler pattern:	Filler pattern:	Post-filler pattern:
1) word: in tag: in	1) list: len: 2 tag: nnp	1) word: , tag: , 2) tag: nnp semantics: state

Prep IN produced because identical in both rules, so a good constraint. In case of post, specific word is underspecifed (dropped) and generalized by means of a semantic class (if available).

Evaluation 1

- Results from 300 computer related job postings, containing 17 different slots
- Precision: ~89%, recall: ~65%
- Hints:
  - words & POS constraint best
  - wordnet does not improve performance

Evaluation 2

- 485 seminar announcements from CMU (half used for training/testing)

stime	etime	loc	speaker
Prec/Rec	Prec/Rec	Prec/Rec	Prec/Rec
93.9/92.9	95.8/94.6	91.0/60.5	80.9/39.4