SEMINAR: INFORMATION EXTRACTION

TOPIC: TEXTRUNNER & KNOWITALL

Haixing Gu

- Part I: Overview of KnowItAll
- Part II: TextRunner
- Part III: A Short Introduction to O-CRF

Part I: Overview of KNOWITALL

Motivation

- Traditional Information Extraction (IE)
  - focus on satisfying precise , narrow, pre-specified requests from small homogeneous corpora.
  - e.g. extract location and time of seminars from a set of announcement
- KNOWITALL system
  - automate the process of extracting large collections of facts from the Web in an unsupervised , domain-independent, and scalable manner.

Part I: Overview of KNOWITALL

Structure

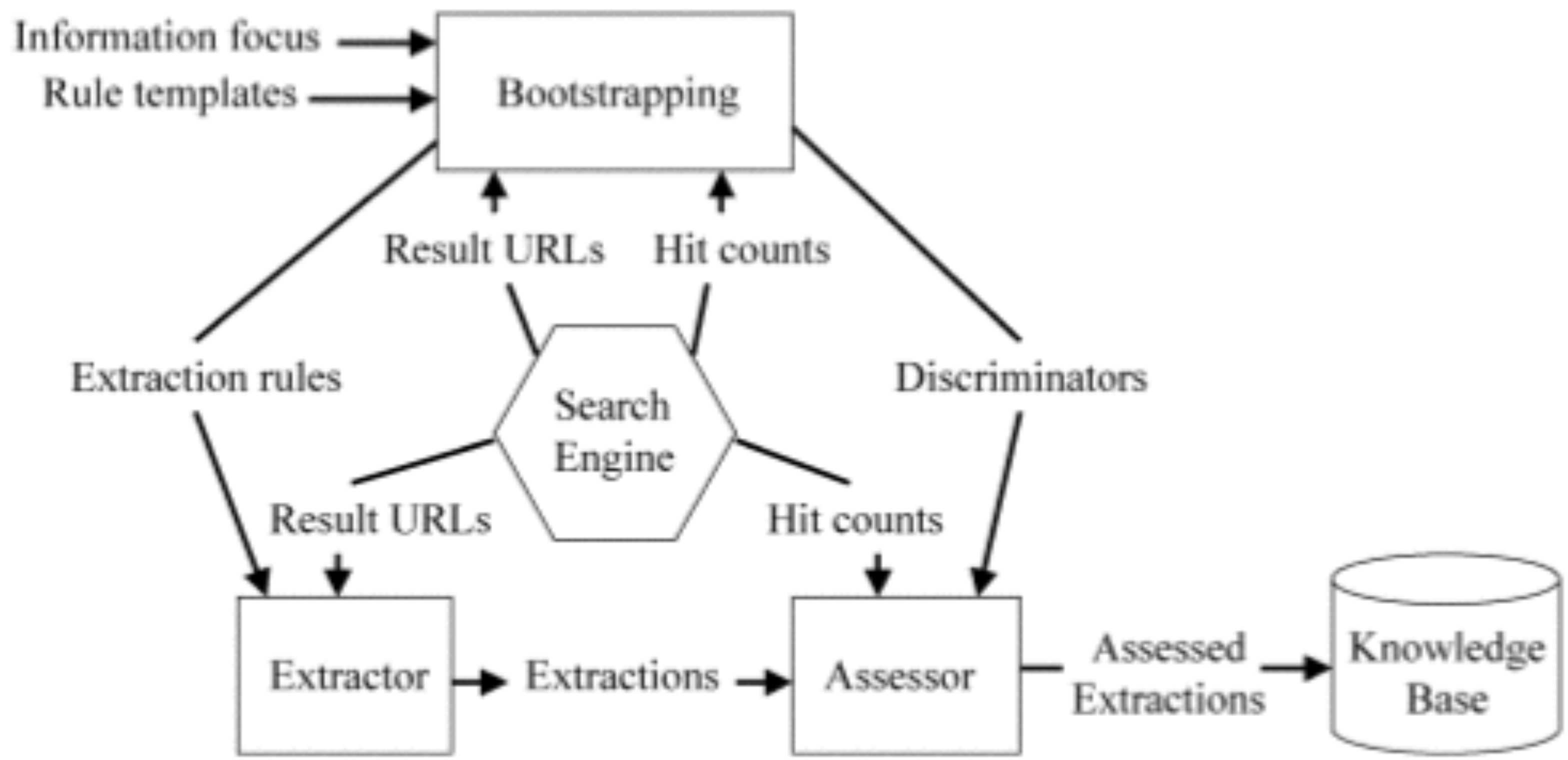

Figure 1

- **Bootstrapping** uses the *two inputs* to produce a set of *extraction rules* and *discriminator phrases* and then trains the discriminators.

- **Extractor** sends queries to search engines and applies extraction rules to extract information from resulting Web pages.

- **Assessor** utilizes discriminators to compute the *probability* that each extraction is correct and adds it to the knowledge base.

Part I: Overview of KNOWITALL

Input

- **Information focus**: a set of predicates that represent classes or relationships.
  - The predicates also give one or more labels for each class
- **Rule templates**: a set of domain-independent extraction patterns

Predicate: City labels: “city”, “town”	Predicate: Film labels: “film”, “movie”
Predicate: Country labels: “country”, “nation”	Predicate: MovieActor labels: “actor”, “movie star”
Predicate: capitalOf(City,Country) relation labels: “capital of” class-1 labels: “city”, “town” class-2 labels: “country”, “nation”	Predicate: starsIn(MovieActor,Film) relation labels: “stars in”, “star of” class-1 labels: “actor”, “movie star” class-2 labels: “film”, “movie”

Figure 2

Predicate:	Class1
Pattern:	NP1 “such as” NPList2
Constraints:	head(NP1)= plural(label(Class1)) & properNoun(head(each(NPList2)))
Bindings:	Class1(head(each(NPList2)))

Figure 3

NP “and other” <class1>
NP “or other” <class1>
<class1> “especially” NPList
<class1> “including” NPList
<class1> “such as” NPList
“such” <class1> “as” NPList
NP “is a” <class1>
NP “is the” <class1>

<class1> “is the” <relation> <class2>
<class1> “,” <relation> <class2>

Figure 4

Part I: Overview of KNOWITALL

Bootstrapping

Predicate:	City
Pattern:	NP1 "such as" NPList2
Constraints:	head(NP1)= "cities"
	properNoun(head(each(NPList2)))
Bindings:	City(head(each(NPList2)))
Keywords:	"cities such as"

Figure 5

Predicate:	CeoOf(Person,Company)
Pattern:	NP1 "," P2 NP3
Constraints:	properNoun(NP1)
	P2 = "CEO of"
	NP3 = "Amazon"
Bindings:	CeoOf(NP1,NP3)
Keywords:	"CEO of Amazon"

Figure 6

- Use the labels to instantiate extraction rules for the predicate.
- Labels are the surface form in which a class may appear in an actual sentence.
- Keywords will be sent to search engines as **queries.**

Part I: Overview of KNOWITALL

Bootstrapping

- Generate a set of discriminator phrases for the predicate based on class labels and on keywords in extraction rules.

- Then train them

Discriminator for: City
"city X"

Discriminator for: CeoOf(Person,Company)
"X CEO of Y"

Figure 7

- Use queries and extraction rules to find some candidate seeds for each predicate.

  - Each seed must have a minimum number of hit counts for the instance itself.

  - We have now: untrained seeds & untrained discriminators

Part I: Overview of KNOWITALL

Bootstrapping

- Compute PMI(c, u) for each seed c and each untrained discriminator u.
    - PMI: pointwise mutual information

$$\mathrm{PMI}(I, D) = \frac{|\mathrm{Hits}(D + I)|}{|\mathrm{Hits}(I)|}$$

I: instance, e.g. New York
D: discriminator, e.g. <I> is a city

Equation 1

- Rank Candidate seeds by average PMI and select the best m seeds.
    - m/2 seeds are used to find the PMI threshold for each discriminator and the other half are used to estimate conditional probabilities.
    - We have now: trained seeds & untrained discriminators

Part I: Overview of KNOWITALL

Bootstrapping

- Select the best k discriminators.
  - <u>We have now</u>: trained seeds & trained discriminators
- Repeat the process to train again the selected seeds and discriminators
  - <u>We have now</u>: retrained seeds & retrained discriminators

Discriminator: <I> is a city Learned Threshold T: 0.000016 P(PMI > T | class) = 0.83 P(PMI > T | ¬class) = 0.08	Discriminator: cities such as <I> Learned Threshold T: 0.0000053 P(PMI > T | class) = 0.75 P(PMI > T | ¬class) = 0.08
Discriminator: <I> and other towns Learned Threshold T: 0.00000075 P(PMI > T | class) = 0.83 P(PMI > T | ¬class) = 0.08	Discriminator: cities including <I> Learned Threshold T: 0.0000047 P(PMI > T | class) = 0.75 P(PMI > T | ¬class) = 0.08
Discriminator: cities <I> Learned Threshold T: 0.00044 P(PMI > T | class) = 0.91 P(PMI > T | ¬class) = 0.25	

Figure 8

Part I: Overview of KNOWITALL

Extractor

Predicate:	City
Pattern:	NP1 "such as" NPList2
Constraints:	head(NP1)= "cities" properNoun(head(each(NPList2)))
Bindings:	City(head(each(NPList2)))
Keywords:	"cities such as"

- Receive the Extraction Rules from Bootstrapping and sends the keywords to a search engine as queries.

- Match the rule to sentences in Web pages returned for the query.

- If all constraints are met, the Extractor creates one or more **extractions**.

  - *e.g. 1: He has visited almost all major European cities such as London, Paris, and Berlin.*

  - *e.g. 2: Detailed maps and information for several cities such as airport maps, city and downtown maps.*

Part I: Overview of KNOWITALL

Assessor

- Assess the likelihood that the Extractor 's conjectures are correct.
- Compute the PMI between each extracted instance and the retained discriminators.
- These PMI statistics are treated as features that are input to a Naive Bayes Classifier (NBC).

$$P(\phi \mid f_1, f_2, \ldots, f_n) = \frac{P(\phi)\prod_i P(f_i \mid \phi)}{P(\phi)\prod_i P(f_i \mid \phi) + P(\neg\phi)\prod_i P(f_i \mid \neg\phi)}$$

Equation 2

Part I: Overview of KNOWITALL

Analysis

- Advantages
  - domain-independent
  - Does not require any manually-tagged training data
- shortcoming
  - Require large numbers of search engine queries and Web page downloads, which means inefficient.
- Experimental results
  - Will be shown along with TEXTRUNNER

Part II: TEXTRUNNER

Motivation

- Goal: Build an **OpenIE**(OIE) system
  - OIE: domain-independent, readily scales to the diversity and size of the Web corpus
  - idea: retain KNOWITALL's benefits but eliminates its inefficiencies
  - implementation: **TEXTRUNNER**

Part II: TEXTRUNNER

Structure

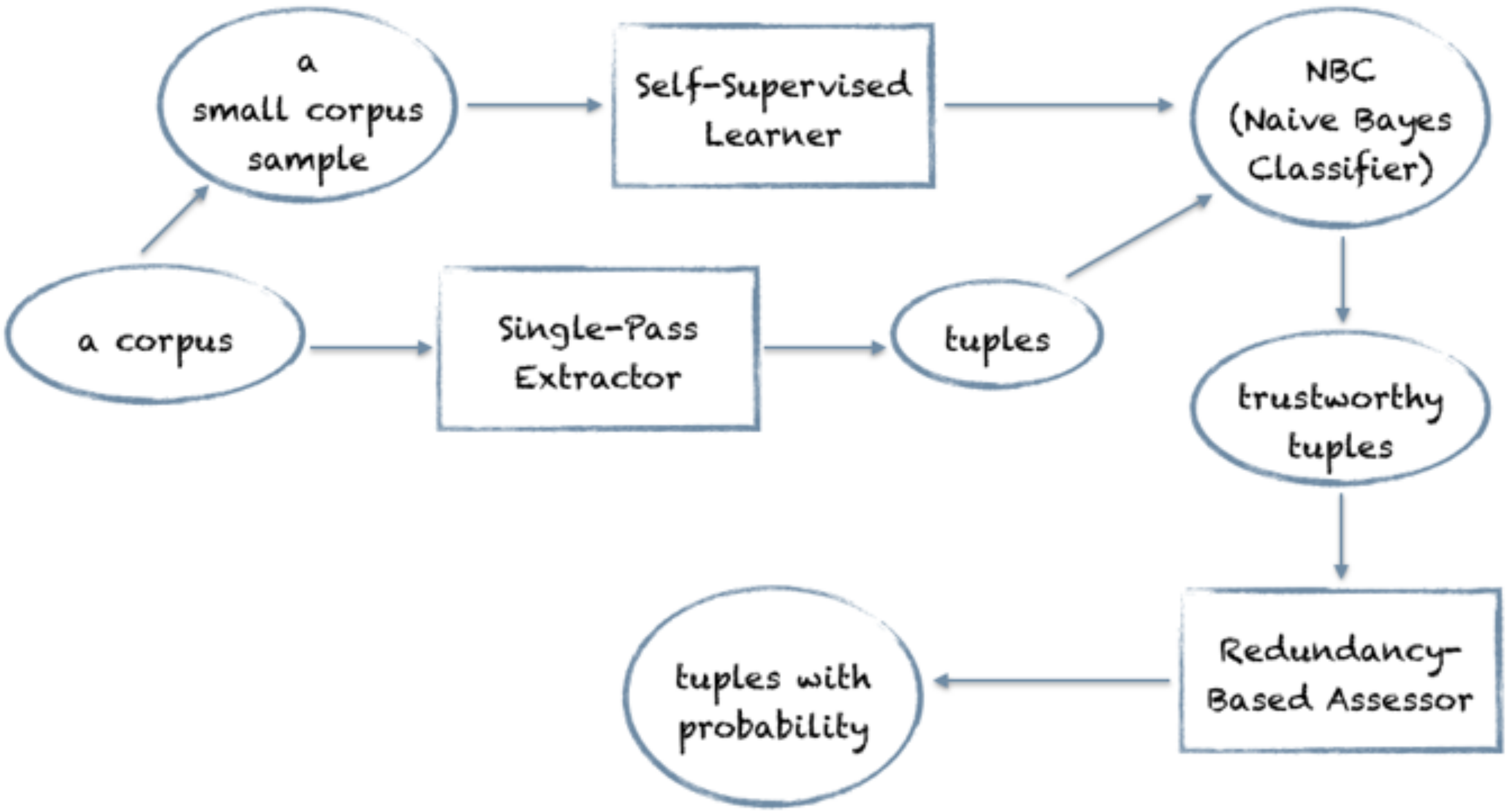

Figure 9

- **Self-Supervised Learner**: output a classifier that labels candidate extractions as trustworthy or not, extractions take the form of the tuple t = (ei, ri,j, ej)
- **Single-Pass Extractor**: generate candidate tuples from each sentence and send them to the classifier
- **Redundancy-Based Assessor**: assign a probability to each retained tuple

Part II: TEXTRUNNER

Self-Supervised Learner

- Parse several thousand sentences to obtain their dependency graph representations. [Klein and Manning, 2003]
- For each parsed sentence, find all base noun phrases constituent ei
  - base noun phrases: e.g. “to be solved problem” ➤ “problem”
- For each pair (ei, ej), locate a relation ri,j in the tuple t
  - e.g. Tokyo is the capital of Japan. ➤ Relation: CapitalOf(X, Y)

Part II: TEXTRUNNER

Self-Supervised Learner

- Label t as positive if certain constraints on the syntactic structure shared by ei and ej are met.
  - e.g. Path from ei to ej should cross no sentence-like boundaries
    - Before the trading of wild animals was abandoned, many species disappeared forever.
    - t(trading, is abandoned, species) ➤ negative
- Map each tuple to a feature vector representation
- Use these features as input to a Naive Bayes Classifier

Part II: TEXTRUNNER

Single-Pass Extractor

- Tag each word in each sentence with its most probable part-of-speech. (maximum-entropy models)
- Use these tags to find entities by identifying noun phrases(noun-phrase chunker[Ratnaparkhi, 1998])
  - The chunker also provides a probability with which each word is believed to be part of the entity.
  - These probabilities are subsequently used to discard tuples containing entities found with small probability.
- Find relations by examining the text between the noun phrases and eliminating non-essential phrases.
  - e.g. “definitely developed” ➤ “developed”
- Present candidate tuple t to the classifier. If t is labeled as trustworthy, it will be extracted and stored.

Part II: TEXTRUNNER

Redundancy-Based Assessor

- Merge tuples where both entities and normalized relations are identical and count the number N of distinct sentences from which each extraction was found.
    - "Little Jack is reading a book."
    - "Jack has read a lot of comic books."
    - "Jack will read another new book."
    - ➤ t(Jack, read, book)   N = 3
- Use N to assign a probability to each tuple using the probabilistic model used in KNOWITALL system.

Part II: TEXTRUNNER

Query Processing

- Each relation found during tuple extraction is assigned to a single machine.
- Every machine computes an inverted index[**Lucene**: text search engine library]
    - We use Lucene because:
        - Given documents, Lucene can compute an inverted index for us
        - Lucene is *Free & Open source!*

Part II: TEXTRUNNER

Analysis

- Traditional IE system: O(R · D) where R: number of relations, D: number of documents
- TEXTRUNNER: tuple extraction in O(D) where & O(TlogT) time to sort, count and assess the set of T tuples found by the system
- TEXTRUNNER extracts facts at an average speed of 0.036 CPU seconds, by dependency parser is 3 sec.
  - TEXTRUNNER is more than 80 times faster
  - 18 sentences in one Web page ——> 0.65 CPU sec. per page
  - 9 million Web pages ——> less than 68 CPU hours
  - Divide the corpus into 20 chunks ——> less than 4 CPU hours
  - 5 additional CPU hours to merge and sort the tuples.

Part II: TEXTRUNNER

Experimental Results

- TEXTRUNNER VS KNOWITALL (extracting from 9 million Web pages)
  - TEXTRUNNER 's average error rate is 33% lower than KNOWITALL's
  - TEXTRUNNER: 85 CPU hours to perform all relations in the corpus at once
    - KNOWITALL: 6.3 hour per relation

(<proper noun>, acquired, <proper noun>)
(<proper noun>, graduated from, <proper noun>)
(<proper noun>, is author of, <proper noun>)
(<proper noun>, is based in, <proper noun>)
(<proper noun>, studied, <noun phrase>)
(<proper noun>, studied at, <proper noun>)
(<proper noun>, was developed by, <proper noun>)
(<proper noun>, was formed in, <year>)
(<proper noun>, was founded by, <proper noun>)
(<proper noun>, worked with, <proper noun>)

Figure 10

	Average Error rate	Correct Extractions
TEXTRUNNER	12%	11,476
KNOWITALL	18%	11,631

Figure 11

Part II: TEXTRUNNER

Experimental Results

- Global Statistics on Facts(tuples) Learned
  - restrict our analysis to a subset of tuples with high probability
    - the probability is at least 0.8
    - the tuple's relation is supported by at least 10 distinct sentences
    - top 0.1% relations will be not considered
  - our estimations(manually estimated):
    - correctness of facts
    - number of distinct facts

Part II: TEXTRUNNER

Experimental Results

- Estimating the Correctness of Facts
  - randomly select 400 filtered tuples.
  - judge whether the relation was well-formed
    - e.g. well formed: *located in*; not well formed: *of securing*
  - judge to see if the arguments were reasonable for the relation
    - e.g. well formed: (Shibuya, located in, Tokyo)
    - not well formed: (23, located in, Tokyo)
  - judge each concrete and abstract tuple as true or false
    - concrete: (Tesla, invented, coil transformer)
    - abstract: (Einstein, derived, theory)

Part II: TEXTRUNNER

Experimental Results

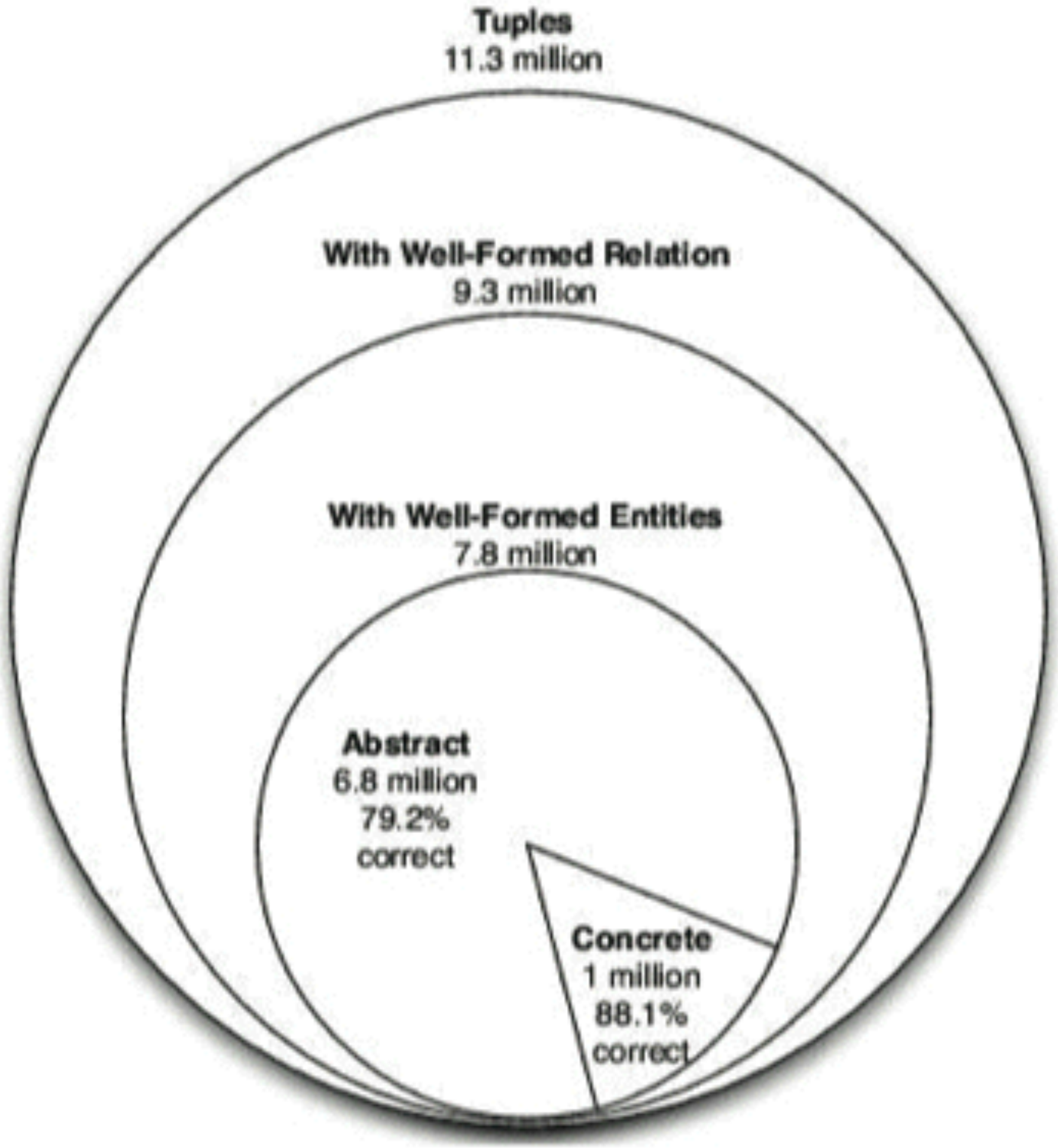

Figure 12

Part II: TEXTRUNNER

Experimental Results

- Estimating the Number of Distinct Facts
  - Further merge the relations (91%)
    - e.g. “invented” “was invented by”
  - Find clusters of concrete tuples,
    - e.g. cluster1: (A,{r1, r2,…, rn},B)
  - only one third tuples belongs to clusters
  - randomly sampled 100 cluster and manually determine how many distinct facts existed within each cluster ——> three quarter
  - ——> 2/3 + 1/3 ・3/4 , so almost 92% of the tuples are distinct

Part III: A Short Introduction to O-CRF

Motivation

- 95% extraction patterns can be grouped as shown in Table 1
  - —> relation-independent extraction is feasible

Table 1

Relative Frequency	Category	Simplified Lexico-Syntactic Pattern
37.8	Verb	E_1 Verb E_2 *X established Y*
22.8	Noun+Prep	E_1 NP Prep E_2 *X settlement with Y*
16.0	Verb+Prep	E_1 Verb Prep E_2 *X moved to Y*
9.4	Infinitive	E_1 to Verb E_2 *X plans to acquire Y*
5.2	Modifier	E_1 Verb E_2 Noun *X is Y winner*
1.8	Coordinate_n	E_1 (and|,|-|:) E_2 NP *X-Y deal*
1.0	Coordinate_v	E_1 (and|,) E_2 Verb *X , Y merge*
0.8	Appositive	E_1 NP (:|,)? E_2 *X hometown : Y*

- TEXTRUNNER uses Naive Bayes Classifier(NBC)
  - Predict relation of a single variable
- Graphical models such as Conditional Random Fields(CRF) can model multiple, interdependent variables
- So we use CRF instead of NBC

Part III: A Short Introduction to O-CRF

Training

- Apply a phrase chunker to documents to get noun phrases candidates ei
- if ei - ej < maxDistance then pij = pair(ei, ej)
    - ENT: entity
- Tokens in the context are treated as possible relations
    - B-REL: start of a relation
    - I-REL: continuation of a relation
    - O: not believed to be part of a relation

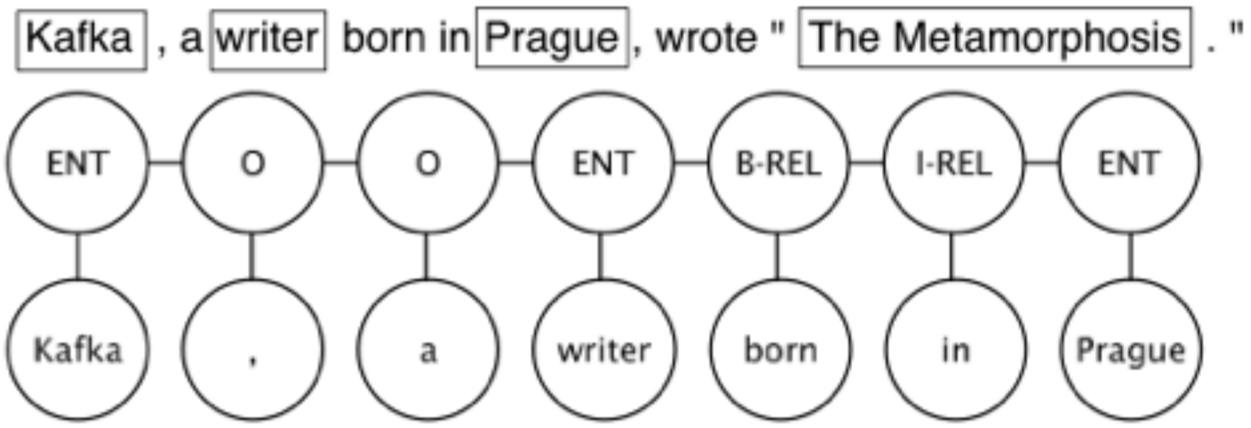

Figure 13

Part III: A Short Introduction to O-CRF

Extraction

- Perform entity identification using a phrase chunker
- Use CRF to label relations
- Apply RESOLVER Algorithm[Yates and Etzioni, 2007] to find relation synonyms

Part III: A Short Introduction to O-CRF

Experimental Results

- O-CRF achieves both double the recall and increased precision relative to O-NB

Category	O-CRF P	O-CRF R	O-CRF F1	O-NB P	O-NB R	O-NB F1
Verb	93.9	65.1	76.9	100	38.6	55.7
Noun+Prep	89.1	36.0	51.3	100	9.7	55.7
Verb+Prep	95.2	50.0	65.6	95.2	25.3	40.0
Infinitive	95.7	46.8	62.9	100	25.5	40.6
Other	0	0	0	0	0	0
All	**88.3**	**45.2**	**59.8**	86.6	23.2	36.6

Table 2

Reference

- Etzioni, O., Cafarella, M., Downey, D., Popescu, A., Shaked, T., Soderland, S., Weld, D.S. and Yates, A. (2004) ***Unsupervised named-entity extraction from the Web: An experimental study***
- Banko, M., Cafarella, M., Soderland, S., Broadhead, M. and Etzioni, O (2007) ***Open Information Extraction from the Web***
- Banko, M and Etzioni, O (2008) ***The Tradeoffs Between Open and Traditional Relation Extraction***

The End

Thanks for Attention

Questions ?

Appendix

Inverted Index

- a simple example of inverted index:
    - if we search “panda eat”, then {0, 2} ∩ {0, 1} = {0}

Tuple	***Ei***	***Ri,j***	***Ej***
T[0]	panda	eat	bamboo
T[1]	tiger	eat	meat
T[2]	child	like	panda

→

panda	0, 2
eat	0, 1
bamboo	0
tiger	1
meat	1
child	2
like	2

Appendix

Precision and Recall

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

- e.g. there’re 5 black and 5 white balls in a box
- Task: take out all the black ones
- if I have taken out 4 black and 4 white
- Precision: 4/8 = 0.5
- recall: 4/5 = 0.8