Literature Survey: Entity and Relationship Extraction

Sagar Sontakke
IIT Bombay
sagarsb@cse.iitb.ac.in

The field of entity extraction has got an immense importance to process this kind of unstructured data. Extracting information from such data is a challenging task and it can be achieved through entity extraction. The way how different entities interact with each other is also an equally imporatant factor to build the knowledge base. The report explains and examines various methods to achieve this goal of extracting entities and their relationships from the automobile warranty data. It presents and evaluates a rule based system to perform this task.

1 Rule Based Entity-Relationship Extraction

A rule based system depends mainly on the rules that are hand-code or automatically learn from the data. A rule based system consists of two parts: a collection of rules and a set of policies to fire these rules. We need to consider sevaral apsects as described in the following sections.

1.1 Form and Representation of Rules

A basic rule have a form like this: "Contextual Pattern → Action". A contextual pattern consists of one or more labeled patterns capturing properties of the entities. A labeled pattern consists of a pattern that is usually a regular expression. The action part generally refers to the tagging actions with entity tokens.

Features of Tokens

A token can have a number of features like:

- String representing the token
- Orthography type of the token. It includes small or capital cases, mixed cases, special symbols, numbers, punctuations, etc.
- Part of Speech tag of the token
- The domain dictionary in which the token appears. For example, dictionary of city names, companies, people names, etc.
- Any annotations, or tags assigned by the preprocessing steps.

Rules for Identifying Single Entities

These rules can have three parts(Sarawagi, 2007) as below:

1. A pattern capturing the context before the entity, it is optional
2. A pattern capturing the entity
3. A pattern capturing the context after the entity, it is optional

Some example of rule formation are as below:

- A pattern for identifying the person names of the form *"Dr. Abdul Kalam"* which consists of a title ("Dr."), a dot, two capitalized words. The rule can be of the form:

  ({DictionaryLookeup=Titles} {String="."} {OrthographyType= Capitalized word}{2}) → Person Name

- The rule for the entity year can be given as below. Example sentence: *"Elections will happen by 2015."*

  ({String = "by" | String = "in"})({OrthographyType = Number}):y → Year=:y

- Similarly, the rule for identifying a company name like "The Google Corp.", can be given as:

({String = ”The”}) ({OthographyType = Capitalized Word})
({OrthographyType = Capitalized word, DictionaryType =
”Compnay end”}) $\rightarrow$ Company Name

Rules to Mark Entity Boundaries

Certain entities consists of multiple words. Such type of entities need to mark start and end boundaries in order to do the extraction effectively. We insert the $< start >$ and $< end >$ markers before and after the entity.

Below is an example to illustrate this:

- The entity like book or journal names cosists of multiple words. Example sentence: *This concept is explained in The Design of Algorithms book.* The rule for such entities can be given as follows:

  ({String = ”explained”} {String = ”in”}):book_start
  ({OrthographyType = Capitalized word}{2-5}) $\rightarrow$
  insert $< book\_start >$ after:book_start.

1.2 Organizing Collection of Rules

A typical rule based system consists of a large number of rules. These rules are fires in some predspecified order. However, many of the rules can have overlapping text regions and may result in conflicting actions. Rules can be organized as explained in the subsequent text.

1.3 Unordered rules with policies to resolve conflicts

In this approach, all rules are treated as an unordered list and each rule is fired independently. A conflict may occur when the spans covered by two different rules overlap. In such cases we can do:

- Prefer the rule that marks larger span
- Merge the spans of text that overlap

1.4 Rules arranged as an ordered set

A complete priority is defined over rules which defines the order in which the rules should be fired. In learning based systems, the rules priorities can be determined as a function of the precicion or recall of the rule on the training data.

Defining ordering or priority over rules can be useful for the later rules that can be benefited from the previous rules. This is particularly useful for fixing the errors of the unmatched tags. Below is an example where the rule with lower pririty gets benefited from the rule with higher priority.

- The rule $< start\_book >$ (**rule1**) has higher priority than the rule $< /end\_book >$ (**rule2**). The rule for $< /end\_book >$ can be fired if the rule $< start\_book$ is already successful.

  Rule1:
  ({String = ”explained”} {String = ”in”}):book_start
  ({OrthographyType = Capitalized word}{2-5}) $\rightarrow$
  insert $< book\_start >$ after:book_start.

  Rule2:
  tag = $< start\_book >$) ({OrthographyType = word}+):book_end
  {String = ”chapter”} $\rightarrow$ insert $< /book\_end >$ after:$< book\_end >$

1.5 Rule Learning Algorithms

Rules can be prepared manuall by a domain expert or can be learnt from the data. In this section we look at the methods of learning the rules automatically.

Primarily, there are two major methods of leaning the rules:

1. Bottom-up Rule Fomation
2. Top-down Rule Fomation

Below are some terms used in these two algorithms:

- **Training data:**
  $D = x_1, x_2, ...x_N$, set of N documents where occurences of entities are marked correcly.
- **Rules**
  $R_1, R_2, ..., R_k$ set of k rules to be learnt from training data.

- **Coverage of a rule**
  The fraction of the text $S(R)$ to which the body of rule R matches.

- **Precision of a rule**
  Out of the $S(R)$ segments matched by rule R, say $S'(R)$ are correct. The fraction $S'(R)/S(R)$ is called as the precision of the rule R.

In rule learning, our aim is to cover all segments that are matched by one or more rules. The precision of each rule must be high. Ultimately the set of minimal rules that have better recall and precicion over the training data is considered as the final answer. One measure to cover every entity of the document (100%) recall and correctly (100%) precision is to have all the rules specific to the entities. But it does not guarantee generalizability and minimal number of rules. So, the goal is to find out a set of minimal number of rules that can ensure a better recall with some loss of precision.

Findig such a size optimal set of rules is intractable. So, rule learning algorithms follow a greedy hill climbing strategy for learning one rule at a time under the following general framework:

Algorithm 1 General Algorithm for Rule Learning

```
Require: Entity tagged training data
Rset ← set of rules, initially empty
while there exists an entity x ∈ D and not covered by rule in Rset do
    Form a new rule around x
    Add this rule to Rset
end while
Post process rules to prune away redundant rules
```

Below we look over the Bottom up rule learning algorithm.

1.6 Bottom-up Rule Learning Method

In bottom-up rule learning method, the starting rule is very specific. It has a minimal coverage but 100% precision and it is guaranteed to be non-redundant. This rule is gradually made more general so that the coverage increases with some possible loss of precision.

The outline of the algorithm is shown below:

Algorithm 2 Bottom-up Algorithm for Rule Learning

```
Require: Entity tagged training data
Creation of seed rule from uncovered instances
Generalization of seed rule
Removal of instances that are covered by new rules
```

Creation of seed rule

- A seed rule is created from an instance x that is not already covered by existing rules.
- A seed rule is a snippet of w tokens to the left and right of T in x fiving rise to very specific rule of the form: $x_{i-w}..x_{i-1}..x_{i+w}T$ where T appears at position x_i
- Example of the seed rule creation for a *person name* for the sentence:

  "According to Abdul Kalam, India will be the most powerful country by 2020."

  Seed rule:

  ({String = "According"} {String = "to"}):person_start
  {String = "Abdul"} {String = "Kalam"} → insert $< person >$ at:person_start

Generalizing seed rule

- The seed rule is generalized either by dropping the tokens or by replacing the tokens with the more general feature of the token.
- For example, consider the seed rule created for the above sentence. It can be generalized as below:

  Generalized rule

  ({String = "According"} {String = "to"}):person_start
  {OrthographicType = Capitalized word}
  {OrthographicType = Capitalized word} → insert $< person >$ after:person_start

Pruning away redundant rules

- We need to get a minimal set of rules that have almost 100% coverage and better precision

- Some of the rules might be repeated and are redundant. Such rules need to be pruned away.

1.7 Case Study: Rule Based System(Berland and Charniak, 1999)

The work by Matthew Berland, Eugene Charniak(Berland and Charniak, 1999) is to find parts in a very large corpora. This is one of the earlier rule based system that focus to extract parts of an object from whole.

According to this work, part is not necessary a physical object, it can be a concept as well. For example, ”odometer” is a part of ”car” while ”plot” is a part of a ”novel”. The implemantation of this system involves manual creation of the patterns for the parts by some domain expert. To create such patterns, they made use of the sentence structures. Some of the sentences are as shown in figure1

... the **basement** of the **building**.
... the **basement** in question is
in a four-story apartment **building** ...
... the **basement** of the apartment **building**.
From the **building's basement** ...
... the **basement** of a **building** ...
... the **basements** of **buildings** ...

Figure 1: Observed Sentences

The patterns are created in terms of the part of speech tags. These are much like regular expressions. The example patterns are as shown in figure2

A. *whole* NN[-PL] 's POS *part* NN[-PL]
...building's basement ...

B. *part* NN[-PL] of PREP {the|a} DET
mods [JJ|NN]* *whole* NN
...basement of a building ...

C. *part* NN in PREP {the|a} DET
mods [JJ|NN]* *whole* NN
...basement in a building ...

D. *parts* NN-PL of PREP *wholes* NN-PL
...basements of buildings ...

E. *parts* NN-PL in PREP *wholes* NN-PL
...basements in buildings ...

Figure 2: Patterns Based on PoS Tags

These patterns are then applied to the test data and results are postprocessed. The postprocessing of results involves filtering out words ending with ”ing”, ”ness”, ”ity”, since though these words are nouns, they do not generally represent a part.

Problems Faced

- Idiomatic phrases could not be pruned away.For example, some idioms like ”Son of a gun”.
- Part of Speech Tagger mistakes. Mistakes at the PoS tagging level are percolated to further steps. For example, ”the re-enactment of the car crash” $\rightarrow$ ”re-enactment” will be a part if ”crash” is tagged as a verb.

Results

This system achieved an accuracy of about 55%. The main reasons for the lower accuracy are because of the problems faced. If the mistakes occur at PoS tagger level, they are cascaded to the next level. Sometimes, the rules or patterns created are too brittle to capture all kinds of entities.

2 Statistical Entity-Relationship Extraction

In statistical approach(Wróblewska and Sydow, 2012), and (Roth and Yih, 2002) the main goal is to design a decomposition of the unstructured text and then labeling various parts of the decomposition, either jointly or independently.

The decomposition can be performed in following ways:

- Labeling tokens, word chunks, segments
- Grammar based methods
- Relationship Extraction with feature based methods

2.1 Labeling tokens, word chunks, segments

The unstructured text is considered as the sequence of tokens and the extraction problem is to assign the an entity label to every token.

Labeling Tokens

- We denote the sequence of tokens as $x = x_1, x_2, ..., x_n$.
- At the time of extraction, each token x_i from the sentence has to be classified into one of a set Z of labels.

- This gives rise to a tag sequence $y = y_1, y_2, ..., y_n$
- The set Z comprises of teh entity types E and a special label "other" for the tokens that do not belong to any other entity types.
- The BCOE encoding style (Begin, Continue, Other, End) can be adopted.

Segment Level Models

In this model, features are defined over segments comprising of multiple tokens forming an entire entity string. In this way features can capture joint properties over all the tokens forming part of an entity.

For segment $S_j = (Y_j, L_j, U_j$, the feature is of the form:

$$F(Y_j, Y_{j-1}, X, L_j, U_j)$$

Entity Level Featues

- **Similarity to an entity in the database**

  For example, if we have a database of company names, and a segment of our inputs matches one of the entries entirely, then this gives a strong clue to mark the segment as a book name. In a sequence model, we cannot enforce arbitrary segment of text to take the same label. Thus, we can define a segment-level feature of the form:

  $f(y_i, y_{i-1}, x, 3, 5)$
  $= [[x_3, x_4, x_5 appear sinalist of books]]$
  $= .[[y_i = book]]$

  In general, since unstructured data is noisy, we define real-valued features that quantify the similarity between them instead of boolean features that measure exact match. It is more useful to define.

  For example, the feature below captures the maximum TF-IDF similarity between the text span $x_3x_4x_5$ and an entry in a list of book names.

  $f(y_i, y_{i-1}, x, 3, 5)$
  $= max_{J \in books} TFIDF similarity(x_3x_4x_5, J)]]$
  $= .[[y_i = book]]$

- **Length of the Entity Segment**

  The length of a segment can be used as a typical feature that captures the length distributuons of the entities. Thus, we can define a feature as a legth.

  The length of a book title can be of 3 words. It can be given as:

  $f(y_i, y_{i-1}, x, l, u) = [[u - l = 5]].[[y_i = title]]$

2.2 Grammar based methods

Some entity extraction systems require a better interpretation of the structure of the source. A grammar-based model uses a set of production rules, like in CFG, to express the global structure of the entity.

For example, to capture the style homogeneity amongst author names in a citation we can define a set of production rules as shown in figure[]

R: S → AuthorsLF | AuthorsFL
R0: AuthorsLF → NameLF_Separator AuthorsLF
R1: AuthorsFL → NameFL_Separator AuthorsFL
R2: AuthorsFL → NameFL
R3: AuthorsLF → NameLF
R4: NameLF_Separator → NameLF Punctuation
R5: NameFL_Separator → NameFL Punctuation
R6: NameLF → LastName First_Middle
R7: NameFL → First_Middle LastName

Figure 3: Context Free Grammar Rules

Each production rule is of the form:

$$R \rightarrow R_1 R_2$$

It is scored a follows:

$$S(R) = S(R_1) + S(R_2) + w * f(R, R_1, R_2, x, l_1, r_1, r_2)$$

where
(l_1, r_1) and $(l_1 + 1, r_2)$ are text spans in x that R_1 and R_2 cover, respectively
w is the weighting function (eg. proper name, verb, etc.)

The score of a node depends on the production used at the node and the text spans that its children

$R1$ and $R2$ cover. This method of scoring makes it possible to find the highest scoring parse tree in polynomial time. Here is an example with scoring.

Peter Haas, George John

One of the many possible parses of the string is:

$R_0 \rightarrow R_4 R_3$
$R_4 \rightarrow R_6 x_3$
$R_3 \rightarrow R_6$
$R_6 \rightarrow x_1 x_2$
$R_6 \rightarrow x_4 x_5$

The total score of this tree is:

$$
\begin{aligned}
&w * f(R_0, R_4, R_3, x, 1, 3, 5)\\
&+w * f(R_4, R_6, Punctuation, x, 1, 2, 3)\\
&+w * f(R_3, R_6, \varphi, x, 1, 2, 2)\\
&+w * f(R_6, x, 1, 2) + w * f(R_6, x, 3, 4)
\end{aligned}
$$

2.3 Relationship Extraction: Feature based methods

Two or more entities are related with each other with some predefined relation like "is_employee_of" is a relationship between a *person* entity and an *organization* entity. Relationships can be binary (between two entities) or they can be multiway (between multiple entities). Binary relations can be expressed as a triplet of the form $< subject, predicate, object >$. Most common types of resources that are useful for relationship extractions are as below.

- **Surface Tokens**
  - The tokens around and in-between the two entities often hold strong clues for relationship extraction
  - For example, the relation *situated_in*, for the following sentence:

    [Company] Symantec [/Company] **is_located_in** the [Location] Pune [/Location]
- **Parts of Speech tags**
  - Part of speech (POS) tags play a more central role in relationship extraction than in entity extraction.
  - Verbs play a crucial role in the relationship extraction.
  - For example, the relation **held_in**, following PoS tagged sentence:

    The/DT University/NNP of/IN Helsinki/NNP **hosts/VBZ** ICML/NNP this/DT year/NN
- **Syntactic Parse Tree Structure**
  - A parse tree groups words in a sentence into prominent phrase types such as noun phrases, prepositional phrases and verb phrases
  - It is more elaborative than PoS tags
- **Dependency Parse Tree**

  Dependency graph structure(Choi and Choi, 2008) links each word in the sentence to other words that depend on it. For example, consider the sentence:

  Haifa located 53 miles from Tel Aviv will host ICML in 2010

  The dependency graph for this sentence can be given as shown in figure[]. The arrows shows the dependency (binary relation) between the words that it connects.

Figure 4: Dependency Graph for the example sentence

Feature Based Methods for Relationship Extraction

Many attempts are made to convert the features that are in form of tree, graph, etc. to flat structures which can then be utilized by classifiers.

Let x denotes sentence and $x_1, x_2, ..., x_n$ denote words of the sentence at index 1,2,...n. Suppose $E1$ and $E2$ denote the segments in sentence x corresponding to the two entities for which we wish to know the relationship. Each word x_i is associated with a set of properties $p_1, p_2, ..., p_k$. These are the features like PoS tags, orthography type, class of word in given ontology, entity type, etc. The first set of features are obtained by taking all possible conjunctions of the properties of the two tokens representing the two entities $E1$ and $E2$. Examples of such features are given below:

- For "resides_in" relation, the feature can be given as:

  $[[Entitylabelof E1 = Person, Entitylabelof E2 = Location]]$

- For "acquired_by" relation, the feature can be given as:

  $[[Entitylabelof E1 = Company, Entitylabelof E2 = Company]]$

2.4 Case Study: Joint entity and relation extraction using card-pyramid parsing

This a work by Raymond J. Mooney and Rohit J. Kate(Kate and Mooney, 2010). It follows a joint approach for entity and relationship extraction.

Why Joint Approach?

The traditional way of entity and relationship extraction follows a pipelined approach in which first entities are extracted followed by extraction of relations. If entities are incorrectly extracted, these inaccuracies are percolated and it also induces inaccuracies in the relationship extraction.

The joint approach to entity-relationship extraction avoids the drawback of pipelined approach, so this approach is preferred.

For example, if "works_for" is a relation identified by a relation extraction, then it can enforce identifying its arguments as *Person* and *Organization*, about which the entity extractor might not have been confident.

Goal

This approach focus on joint extraction of entities and relations to improve the accuracies. Use of "card-pyramid" i.e. a graph that compactly encodes all possible entities and relations in a sentence.

Approach

Card Pyramid Structure

It is a tree like graph with one root, internal nodes and leaf nodes. If there are n leaves, there are

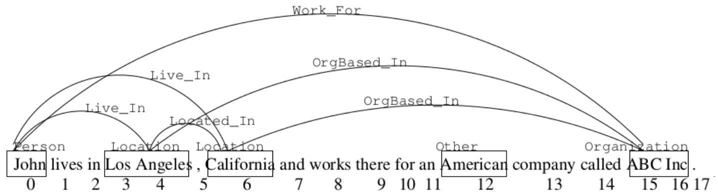

Figure 5: A sentence shown with entities and relations

exactly n number of levels. There are decreasing number of nodes from bottom to top. Leaves are at lowest level (0) and root is at highest level $(n-1)$. Every non-leaf node at position i in level l is parent of exactly two nodes i and $(i+1)$ at level $(l-1)$.

A typical card pyramid structure looks as shown in figure6

Figure 6: A Typical Card Pyramid

The card pyramid strucure for the given sentence is shown in figure7

Card Pyramid parsing

The process of jointly labeling the nodes of a card-pyramid which has all the candidate entities (i.e. entity boundaries) of the sentence as its leaves. It requires grammar for card pyramid in terms of entities and relations.

Entity Productions (leaf nodes):

Entity label $\rightarrow ce$

Relation Productions (non-leaf nodes):

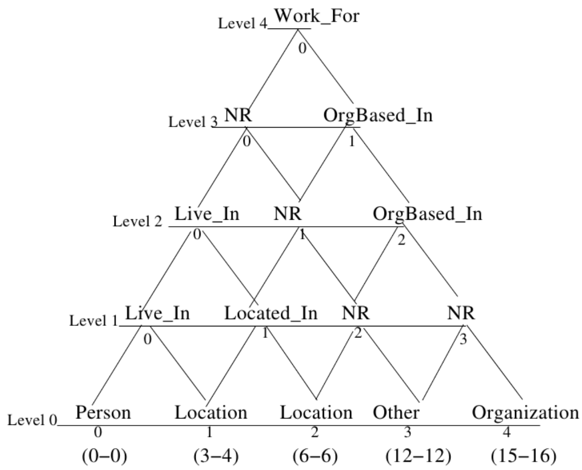

Figure 7: A Card Pyramid structure for given sentence

Relation label $\rightarrow$ $entityLabel1$ $entityLabel2$

Examples:
work_for $\rightarrow$ person organization
NR $\rightarrow$ person other
OrgBasedin $\rightarrow$ Location Organization
Person $\rightarrow$ ce
Location $\rightarrow$ ce

Classifiers Used

Use of two classifiers to extract entities and to extract relations . Classifier for every entity production which gives the probability of a candidate entity being of the type given in the production's LHS. A classifier for every relation production gives the probability that its two RHS entities are related by its LHS relation.

Beam Search

Parsing does a beam search and maintains a beam at every node. The beam at each node is a queue of items we call beam elements. At leaf nodes, a beam element simply stores a possible entity label with its corresponding probability. At non-leaf nodes, a beam element contains a possible joint assignment of labels to all the nodes in the sub-card-pyramid rooted at that node with its probability.

Relation productions for which left-most leaf of the left child and right-most leaf of the right chil are RHS non-terminals. For every such production in the grammar, 2 the probability of the relation is determined using the relation classifier. This probability is then multiplied by the probabilities of the children sub-card-pyramids. Finally, the estimated most-probable labeling is obtained from the top beam element of the root node. This algorithm works in polynomial time.

A support vector machine (SVM) classifier for each of the entity productions in the grammar. It outputs the probability that the candidate entity is of the respective entity type. Probabilities for the SVM outputs are computed using the method by Platt (use all possible word subsequences of the candidate entity words as implicit features using a *word-subsequence kernel*)(Mooney and Bunescu, 2005). In addition to it, PoS tags, actual entity word, context words are used.

References

Matthew Berland and Eugene Charniak. 1999. Finding parts in very large corpora. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 57–64. Association for Computational Linguistics.

D Choi and K Choi. 2008. Automatic relation triple extraction by dependency parse tree traversing. *Poster and Demo*, page 23.

Rohit J Kate and Raymond J Mooney. 2010. Joint entity and relation extraction using card-pyramid parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 203–212. Association for Computational Linguistics.

Raymond J Mooney and Razvan C Bunescu. 2005. Subsequence kernels for relation extraction. In *Advances in neural information processing systems*, pages 171–178.

Dan Roth and Wen-tau Yih. 2002. Probabilistic reasoning for entity & relation recognition. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

Sunita Sarawagi. 2007. Information extraction survey. *Foundations and Trends in Databases*, 1(3):261–377.

Alina Wróblewska and Marcin Sydow. 2012. Dependency-based extraction of entity-relationship triples from polish open-domain texts.